Name: ___________________________ Date: ____________

CoE 0147: Spring 2014 Lab 2

Lab 2: Immediate Values, Memory, and System Calls

Each of you should submit your own solution, according to the instructions at your TA's website. Each person must turn in their own copies of the lab. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab.

The lab due date is posted on the TA's website. Submission timestamps will be checked. Late submissions will not be accepted.

For each of the three parts in this lab, you should write a separate, new program (a new file). You must name your files in a specific format. Each part will tell you the name to use. Follow instructions so that your program produces output in the correct format. In part 1, you will be given some starting code that you will modify.

Part 1: Immediate Values

Immediates are constant numbers that can be loaded into registers, used in arithmetic operations, or used in memory operations. They can often make our code a little clearer and self-explanatory by letting us use constant numbers directly in our instructions. Sometimes, they are absolutely necessary.

An example of using immediate values is seen in the following:

```
addi  $a0,  $zero,  1492  #1492 is the immediate
#$a0 contains the year that Columbus first sailed for the Americas.

addi  $a1,  $a0,  10      #10 is the immediate
#$a1 contains the year of Columbus's last voyage
#(10 years after his original voyage).
```

Due to MIPS instruction set limitations, immediates are limited to being 16 bits large. Immediates larger than 16 bits literally cannot “fit” in a single instruction (remember that instructions are always 32 bits). We often want or need to handle larger immediates (e.g., 24 bits, 32 bits). With some smart instructions manipulation, we can handle immediates of any desired size (e.g., a 32 bit immediate).

For example:

```
#1492 and 10 both easily fit within 16 bits.
#Put the population of New York City into $a2.
addi  $a2,  $zero,  8175133    #8175133=0x7CBEiD

#Problem! The population of New York City does not fit in 16 bits!
#Therefore, the above instruction is invalid.
#There is no single valid MARS instruction that can do the above.
```

MARS will let you use immediates larger than 16 bits, even though such an immediate cannot fit within a single instruction! MARS is “smart enough” to automatically break your instruction that requires a 17 bit or larger immediate into multiple instructions. By running those multiple instructions, one after the other, the result will be as if MIPS could use immediates that require more than 16 bits.

Now you try:

Your goal is to write the distinct sequences of instructions that will put the 32 bit number 0xFACEBEEF into the register $t4. Your solutions can't load any data from memory. This will require more than 1 instruction. Below, you are given template code to help you. You will then answer a few questions about your solutions.

Tip: To figure out the sequence of instructions that will let you put that 32 bit number into $t4, consider looking ad how the load immediate pseudoinstruction (“li”) works. First use that pseudoinstruction to load the number 0xFACEBEEF into $t4. Assemble and examine/test/step through the resulting program. Notice which instructions are *actually* run (pay attention to the “Basic” column). Use those instructions to put 0xFACEBEEF into $t4 and get rid of your “li” instruction.

There are several ways to put 0xFACEBEEF into $t4. The best way to do so will not unnecessarily disturb any registers other than $t4.

Run the program to completion and verify that it works. Use the following template code and insert your answers to the following questions in the template code.

Question 1: What is the machine code (in hexadecimal) of these instructions?
Question 2: What instruction format are these instructions (R, I, or J)?
Question 3: What are the values (in hexadecimal) of the immediate field in each instruction?

The template code you must use is available at:
http://www.pitt.edu/~dbd12/teaching/files/lab2part1.asm

Your book's green sheet will be useful here

Part 2: Memory

Consider the following definition of variables in memory:

```
.data
x:    .word  15
y:    .word  6
z:    .word  0
.text
#instructions go here
```

That above program tells MARS to set aside three words of memory. It initializes each word to a specific value. Each word can be loaded from or stored to by referring to each word's name (“x”, “y”, and “z”). These words of data will be set aside in an area separate from the program's instructions.

a) Write a MIPS program that adds x and y and stores the result in z (z = x + y). Name this program lab2part2.asm. Your program should load x and y from memory into two different registers, perform the operation, and then store the result back into z's memory location.

Hint: consider using the “la” (load address) pseudoinstruction as well as the “lw” (load word) instruction.

Something to think about: Can you find x, y, and z in MARS's data segment view? What are their addresses and what arrangement are they in (i.e., which comes after the other).

b) Take your part a program and modify it so that after z has been stored to, both x and y are overwritten with z's value. **Use constant offsets in your store instructions to do this.** An example of using a constant offset is something like this: `lw $t0, -8($t1)`. In this case, the constant offset is -8. The address stored in $t1 will have -8 added to it, and then the word stored in that location (i.e., the word at address $t1 – 8) will be loaded by the processor and put into $t0.

c) Take your part b program and modify it so that the variables are now bytes. Use the following definition of variables in memory. Modify your program to use the proper load/store byte instructions. Use the following definition of variables in memory:

```
.data
x:    .byte  15
y:    .byte  6
z:    .byte  0
```

Something to think about: Can you find x, y, and z, in MARS's data segment view? What are their addresses?

d) Take your part c program and modify it so that the variables are halfwords. Modify your program to use the proper load/store halfword instructions instead of the load/store byte instruction you were using:

```
.data
x:    .half  15
y:    .half  6
z:    .half  0
```

Something to think about: Can you find x, y, and z, in MARS's data segment view? What are their addresses?

Question 4: Submit your lab2part2.asm program form part2.d. You do not need to submit a program for the other sections/steps (a through c) in part 2.

Part 3: System Calls

Write a MIPS program that prompts the user for two values then prints their difference in the following format: "The difference of X and Y ix Z", there X and Y are values read and Z is their difference.

Sample output:

```
What is the first value?
7  <--- this is input from the user (do not print)
What is the second value?
3  <--- this is input from the user (do not print)
The difference of 7 and 3 is 4
```

Tip: Use multiple system calls of different kinds to create the output. For example, use a print string syscall to print the first part of the output ("The difference of "), then a print integer syscall, then another print string syscall (" and "), etc. Remember that "\n" is a new line (blank line). Be sure that your program's output exactly matches the format of the above sample output.

Mips1.asm from the lecture examples contains an example of using a syscall. You may find it useful to refer to that sample program.

Useful MARS system calls list:
http://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html

Question 5: Submit your part 3 program. Be sure to name the file lab2part3.asm.