Name: Nick Petro **CoE 0147**

Lab 2, Spring 2016

Lab 2: Immediate Values, Memory, and System Calls

Important:

- While collaborations are allowed, the names of collaborators should be mentioned in the assignment. Each student should submit individual assignments for grading despite collaboration.
- **Paper copy** of the assignment is due in the class room, **before** the beginning of the next recitation.
- Solution codes (**ASM files**) for this week should be emailed to `Courseweb/CourseDocuments/Recitations/Lab2` before the next recitation, with "COE0147 Lab-2" in the subject line. Please zip the files together and submit them as a single entity.

Note: For each of the three parts of this assignment, you are expected to write separate programs (files), and name your files as instructed in each section. Also, include your name at the beginning of each file.
For example:

#Name: First_name Last_name
#Program starts from here.

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, 1939 # 1939 is the immediate
# $a0 contains the year in which world war 2 started

addi $a1, $a0, 6 #6 is the immediate
# $a1 contains the year in which world war 2 ended
```

Due to MIPS instruction set limitations, immediates are limited to being 16 bits large. Immediates larger than 16 bits cannot "fit" in a single instruction (remember that instructions are always 32 bits). We often 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:

```
# 1939 and 6 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 $t0. Your solutions can't load any data from memory. This will require more than 1 instruction. Below, you are given a template code to help you. Following this, you will answer a set of questions about your solutions.

Tip: To figure out the sequence of instructions that will let you put that 32 bit number into $t0, consider looking at how the load immediate pseudoinstruction ("li") works. First use that pseudoinstruction to load the number 0xFACEBEEF into $t0. 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 $t0 and get rid of your "li" instruction.

There are several ways to put 0xFACEBEEF into $t0. The best way to do so will not unnecessarily disturb any registers other than $t0.

Run the program to completion and verify. 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 is the format for 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 on the courseweb as File1.asm under Lab2.

Submit *lab2part1.asm* program for this section.

2 Memory

Consider the following definition of variables in memory:

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

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

a) Write a MIPS program that subtracts y from x 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 "lb" (load byte) 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 code from part (a) 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: lb $t0, -8($t1). In this case, the constant offset is -8. The address stored in $t1 will have -8 added to it, and then the byte stored in that location (i.e., the byte at the address $t1 - 8) will be loaded by the processor and put into $t0.

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

```
.data
x: .word 15
y: .word 6
z: .word 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 code from part (c) 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 word 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 *lab2part2.asm* program for part (d) only. You do not need to submit a program for the other parts (a through c) in section 2.

3 System Calls

Write a MIPS program (***lab2part3.asm***) that prompts the user for two values, then prints their sum and difference in the format provided.

Sample output:

What is the first value?
5 $<--$ *this is input from the user (do not print)*
What is the second value?
4 $<--$ *this is input from the user (do not print)*
The sum of 5 and 4 is 9 and their difference is 1

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 sum 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.

4 Practice Problems

1. (Q2.9) Translate the following C code to MIPS. Assume that the variables `f, g, h, i` and `j` are assigned to registers `$s0, $s1, $s2, $s3`, and `$s4`, respectively. Assume that the base address of arrays `A` and `B` are in registers `$s6` and `$s7`, respectively. Assume that the elements of the arrays `A` and `B` are 4-bytes words:

   ```
   B[8] = A[i] + A[j]
   ```

2. (Q2.10) Translate the following MIPS code to C. Assume that the variables `f, g, h, i` and `j` are assigned to registers `$s0, $s1, $s2, $s3`, and `$s4`, respectively. Assume that the base address of arrays `A` and `B` are in registers `$s6` and `$s7`, respectively.

```
addi $t0, $s6, 4
add  $t1, $s6, $0
sw   $t1, 0($t0)
lw   $t0, 0($t0)
add  $s0, $t1, $t0
```

3. (Q2.14) Provide the type and assembly language instruction for the following binary value:

   `0000 0010 0001 0000 1000 0000 0010 0000`

4. (Q2.15) Provide the type and the hexadecimal representation of following instruction:

   `sw $t1, 32($t2)`

5. (Q2.20) Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register `$t0` and uses the value of this field to replace bits 31 down to 26 in register `$t1` without changing the other 26 bits of register `$t1`.

Practice Problems:

1)
```
.text
lw $t0, $s3($s6)
lw $t1, $s4($s6)
add $t2, $t0, $t1
sw $t2, 8($s7)
```

2)
```
A[1] = A[0];
f = A[0] + A[1];
```

3)
```
R-Type
add $s0, $s0, $s0
```

4) 0xad490020

5)
```
.text
srl  $t2, $t0, 11
 sll  $t2, $t2, 26
 srl  $t1, $t1,  6
add $t2, $t2, $t1
```