CS/COE 0447 Fall 2009
Lab 2: Immediate values, memory, system calls and endianness
Due Date: September 24, 2009

To get credit on this lab, attend recitation on 9/18. Each of you should submit your own solution, according to these instructions: http://www.cs.pitt.edu/~sab104/teaching/cs447/submission.html. You may collaborate with your partner, but each person must turn in their own copy of the lab, with the name of their partner. The lab is due on 9/24 before midnight.

Part 1: Immediate Values

Consider the following two arithmetic/logic instructions:

```
addi $t1, $zero, -1
ori $t2, $zero, 0xFFFF
```

Question: What is the machine code (in hexadecimal) of these instructions? Is the immediate field (the last 16 bits) the same in both instructions?

Run the program and determine the contents of registers $t1 and $t2 after execution.

Question: What are the values of the registers? Why are they different?

Part 2: Memory

Consider the following definition of variables in memory:

```
.data
x:      .word     9
y:      .word    18
z:      .word
```

a) Write a MIPS program that adds x to y and stores the result in z.

b) Modify your program so that the variables are now halfwords. Use the following definition of variables in memory:

```
.data
x:      .half     9 0
y:      .half    18 0
```

```
z:      .half
```

c) Modify your program again so that the variables are bytes. Use the following definition of variables in memory:

```
.data
x:      .byte      9 0 0 0
y:      .byte     18 0 0 0
z:      .byte
```

Note that additional halfwords (or bytes) are defined so that the next labeled halfword (or byte) starts at a word boundary.

Part 3: System Calls

Write a MIPS program that promts the user for two values and then prints their sum in the following format: “The sum of X and Y is Z”, where X and Y are the values read and Z is their sum.

Go to http://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html for a list of available system calls in MARS.

Part 4: Endianness

The following MIPS code defines space for 4 bytes and initializes them to 0x01, 0x02, 0x03 and 0x04.

```
.data
a:    .byte     0x01 0x02 0x03 0x04
```

Copy the code to the simulator and assemble it.

Question: What is the address of the byte with value 0x04?

If our purpose is to use these 4 bytes as a word, we could have defined the previous label as follows:

```
a:    .word     0x01020304
```

Replace a's definition in the simulator and assemble the code again.

Question: What is now the address of the byte with value 0x04?

Question: Is the simulator little endian or big endian?