CSE 378 Winter 1998

Machine Organization and Assembly Language Programming

Problem Set #3

Due: Wednesday, January 28

In this assignment you'll have to write some simple subroutines in MIPS assembly language using SPIM to debug and test your programs.

Instructions on how to **turnin** your program as well as test input examples and how to use them will be given to you shortly in a separate handout.

Recall that it is **imperative** that your programs be **commented**. Programs without comments might not be looked at, even if correct!

The first two problems are to get you familiar with assembly language. The third one is to prepare you for the next assignment.

PART I. Write a procedure in MIPS assembly language that finds the number of strictly positive elements and the number of strictly negative elements in an array of integers (there could be some zero elements which should not be part of either count).

Your procedure will take two arguments: a pointer to the array of integers that will be passed in register $a0 and the number of elements in the array that will be passed in register $a1. Your procedure will return the number of strictly positive elements in $vo and the number of strictly negative elements in $v1.

Please name your procedure **problem1** so that it can be tested easily.

PART II. Write a procedure in MIPS assembly language that converts a character string of ASCII digits into the integer value represented by the string. The character string is null terminated and a pointer to it will be given as the argument in register $a0. The ASCII representation of characters is in your book page 142 (with the NULL character having the representation 0). Thus the ASCII string "+24" is represented by the 4 bytes (where the value of each byte is given in decimal notation) "43", "50", "52", "0".

Your procedure named **problem2** should return in register $v0 the value of the positive, negative or zero representation of the string. If the string is valid, i.e., is a sign followed by a series of digits, or a series of digits, register $v1 should contain 0. Otherwise, if the string is not valid (e.g., contains a letter, or a digit followed by a sign etc.) the value returned in $v0 is immaterial but register $v1 should contain a 1.

PART III. In this problem we are setting the stage for the next programming assignment.

Many desk calculators and some machine-independent representations of high-level programming languages, such as the Java Virtual machine (JVM) for the very popular Java language, are based on the concept of stack architectures (recall Problem Set #2).

The JVM is an abstract computing machine that is a stack machine whose instructions are called *bytecodes*. A small subset of the JVM instruction set is in the table below:

Name	Bcode	Instr. format	Stack operations	Description
IADD	0x60	IADD	POP v1, POP v2, PUSH res	Integer add
ISUB	0x64	ISUB	POP v1, POP v2, PUSH res	Integer sub
IMUL	0x68	IMUL	POP v1, POP v2, PUSH res	Integer mul
ILOAD	0x15	ILOAD *index*	PUSH res	Load from local var. at index
ISTORE	0x36	ISTORE *index*	POP v1	Store to local var. at index
BIPUSH	0x10	BIPUSH *immed*	PUSH res	Byte immediate push
SIPUSH	0x17	SIPUSH *immed1 immed2*	PUSH res	Short immediate push
POP	0x57	POP	POP v1	POP value, discard
IFEQ	0x99	IFEQ *offset1 offset2*	POP v1	Branch on v1 == 0
IFNE	0x9a	IFNE *offset1 offset2*	POP v1	Branch on v1 !=0
IFLT	0x9b	IFLT *offset1 offset2*	POP v1	Branch on v1 <0
IFLE	0x9e	IFLE *offset1 offset2*	POP v1	Branch on v1 <=0
IFGT	0x9d	IFGT *offset1 offset2*	POP v1	Branch on v1 >0
IFGE	0x9c	IFGE *offset1 offset2*	POP v1	Branch on v1 >=0
GOTO	0xa7	GOTO *offset1 offset2*	none	unconditional branch
IRETURN	0xac	IRETURN	POP v1	Integer return

Each byte code is 1 byte long. An *index* (as in ILOAD *index*) is also 1 byte long (some explanation of what it refers to will be given in the next assignment). *immed*, *immed1* and *immed2* are all immediate values of 1 byte. Thus BIPUSH is followed by 1 byte of immediate while SIPUSH is followed by 2 bytes of immediate. *offfset1* and *offset2* are also 1 byte long meaning that each branch instruction is followed by a 16-bit offset (how this is done will be explained in the next assignment).

In the next assignment you will simulate the JVM. The overall process will be:

1. Fetch the next bytecode
2. Decode it
3. Fetch the operands
4. Execute the operation
5. Store the results

This week, you will start implementing the first two steps, i.e., your "only" task is to take as input a sample string of *bytecodes* and "decode" that string, that is write out the sequence of instructions that the string meant to execute.

For example, the string of bytecodes:
0x15, 0x12, 0x15, 0x06, 0x60, 0x10, 0xa1, 0x64, 0x36, 0x01
should result in:

```
ILOAD 18
ILOAD  6
IADD
BIPUSH 161
ISUB
ISTORE 1
```

You should write your code with the whole sequence of 5 steps above in mind, i.e., your main program should call one or more subroutines, for example for decoding (needed in this part of the assignment), for executing (not needed in this part of the assignment) etc.

You should also have some registers dedicated to some functions of the JVM you are simulating, for example one for the program counter of the JVM (needed in this part of the assignment) one for the stack pointer for the JVM stack (not needed in this part of the assignment and different of course of the $sp of the SPIM machine) etc.