CSE 378 Autumn 1999

Machine Organization and Assembly Language Programming

Problem Set / Programming Assignment #4

Due: Thursday, October 28

This assignment has two components: (1) Problems from the text relative to the Single and multiple cycle implementation of the MIPS ISA and (2) a SPIM program which will be extended in the next assignment (this program is a little tedious but such is life ....)

By now, you should be familiar with the material covered in Chapter 5 (Sections 5.1 through 5.4).

Part I

1. Problems 5.5 and 5.15
2. Problems 5.10

PART II. In this problem we are setting the stage for the forthcoming programming assignment #5.

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:

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 assignment #5; at this point it is sufficient for you to know that *index* is an unsigned integer). *imm* is a signed immediate value of 1 byte, *imm1* and *imm2* are both 1 byte long and when concatenated form a signed immediate value of 2 bytes. 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 assignment #5).

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
IDIV	0x6b	IDIV	POP v1, POP v2, PUSH res	Integer div (quotient)
ILOAD	0x15	ILOAD *index*	PUSH res	Ld from loc. var. at index
ISTORE	0x36	ISTORE *index*	POP v1	St to loc. var. at index
IALOAD	0x2e	IALOAD	see text in Ass 5	Ld from local
IASTORE	0x4f	IASTORE	see text in Ass 5	St to local and pop
BIPUSH	0x10	BIPUSH *imm*	PUSH res	Byte immediate push
SIPUSH	0x17	SIPUSH *imm1 imm2*	PUSH res	Short immediate push
DUP	0x59	DUP		Duplicate top of stack
DUP2	0x5c	DUP2		Dupl. top 2 st. entries
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

In assignment #5 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, 0x59, 0x10, 0xa1, 0x64, 0x36, 0x01
should result in:

```
ILOAD 18
ILOAD  6
IADD
DUP
BIPUSH -97
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.