CS 577 Homework 2 – Extending a MIPS Machine Code generator – 2pm, Wednesday, May 29, 2002

On the course web page, you'll find a program `compile.java`, which generates MIPS assembler code for a small subset of JVM code. The Java subset supported includes integer arithmetic, static methods, and a minimal set of output facilities. The program uses the "Version 2" code generation scheme: local variables and stack slots (up to 10 of each are allowed) are assigned to fixed register numbers. The program uses the BCEL library to handle the JVM code. Also on the course web page, you'll find a pointer to a MIPS simulator called SPIM, which is installed on our Solaris network and can be downloaded and installed on a Unix (or, supposedly, Windows) machine of your choice. You can use the simulator to run (and debug) the generated MIPS assembler code.

For example, to compile and run a Java file `foo.java` containing a suitable `main` method as usual, one can execute:

```
javac foo.java
java compile0 foo.class > foo.s
spim -file foo.s
```

Your assignment is to extend `compile` to deal with integer arrays. This will involve adding support for about twelve new JVM instructions, and testing your modifications.

Details

The subset handled by the existing generator should be sufficient to compile simple integer programs involving `static` methods. (I may have left out one or two rarely used instructions; if so, feel free to implement them!) The only way to do output (supported by a nasty special-case hack) is via `System.out.print(`x`)` where x is an integer or a string. Still, this is enough to write test cases that display their results, and will run under the ordinary JVM as well as via generator+simulator. The generator generally issues an "unimplemented" message about any instruction it can't cope with. It deliberately ignores constructor methods (called `<init>`) since `javac` always produces one of these for every class, even if the class contains only static methods.

To handle arrays, you'll need to add cases to handle twelve additional instructions: `ACONST_NULL`, `ALOAD`, `ARETURN`, `ARRAYLENGTH`, `ASTORE`, `IALOAD`, `IASTORE`, `IF_ACMPEQ`, `IF_ACMPNE`, `IFNONNULL`, `IFNULL`, `NEWARRAY`. Most of the necessary code can be copied from the existing support for integers. The interesting cases are `NEWARRAY`, `ARRAYLENGTH`, `IALOAD`, and `IASTORE`. To see how these instructions are used by real Java programs, write `.java` testcases and use `javap -c` to examine the corresponding bytecode. Make sure you write test cases that exercise all these instructions!

You need to arrange to "raise" the following built-in exceptions when appropriate:

```
NegativeArraySize
NullPointer
ArrayIndexOutOfBounds
```

If one of these exceptional conditions occurs, your generated code should simply print out a message "Uncaught Exception *name*" where *name* is one of the above, and exit immediately. Don't try to implement real exceptions or exception handling!

Represent arrays using the format suggested in lecture: i.e., an n-element integer array is represented by a $n+1$-word heap block, with the length stored in the first word. To allocate a heap block, use the `sbrk` system call (it behaves much like `malloc` and also zeroes the returned memory). Don't worry about deallocation or garbage collection!

The register conventions used by the generator are explained in a source-code comment. You should find them straightforward. You may find the implmentation of INVOKE and RETURN informative, although it is not directly relevant to this assignment.

SPIM is straightforward to use. You'll definitely want to examine the documentation carefully; it gives full details on the MIPS instruction set, assembler format, and SPIM options. (Note: I've had difficulty making vanilla `spim` run in interactive mode from a linux shell; it just sits there eating input and does nothing. The batch mode `spim -file` seems to work fine, as does `xspim` – except for the odd core dump.) In particular, you'll need to learn about the `sbrk`, `print_string` and `exit` system calls; the examples of system calls in the existing code should help.

How to submit your homework.

Submit the homework *on paper* at the beginning of class on the due date. You should submit:

- Your modified version of `compile.java`
- A README and/or `makefile` describing how to compile, build, and run your program, unless this is blindingly obvious.
- Evidence that your program works, in the form of a set of test programs that exercise integer arrays (including exception-causing cases) together with the `.s` output and simulator run results for each test.

Extra Credit

Implement more of the JVM instructions. Or (more fun), implement the "Version 3" generation scheme.