University of Arizona, Department of Computer Science

CSc 553 — Assignment 3 — Due noon, Thu April 7 — 15%

Christian Collberg
January 28, 2011

1 Introduction

This assignment consists of three tasks:

1. Install the `libjit` just-in-time library.
2. Rewrite your interpreter from the previous assignment to generate machine code at runtime.
3. Write one or more benchmark programs to compare the speed of interpretated and jitted code.

Note the following:

1. You don't have to implement the object-oriented features of Luca nor the garbage collector.
2. You should work in teams of two students. Teams of three will have to do more work.

2 Install libjit

1. Install the `git` version control system:
   - Mac OS X: `http://code.google.com/p/git-osx-installer`
   - Linux:

     ```
     > yum install git-core
     > apt-get install git-core
     ```
2. Use `git` to download the latest version of `dotgnu` (`http://dotgnu.org/pnet-git.html`):

   ```
   > git clone git://git.sv.gnu.org/dotgnu-pnet.git
   > cd dotgnu-pnet
   > git submodule update --init
   > git checkout master
   > git pull
   > git submodule update
   ```
3. Build and install `libjit`:

   ```
   > cd libjit
   > auto_gen.sh
   > ./configure
   ```

```
> make
> make check
> sudo make install
```

On my machine, the library gets installed in `/usr/local/include/jit` and `/usr/local/lib/x86_64`.

4. Build documentation:

```
> cd doc
> ./mkpdf.sh
```

5. Learn from the examples:

```
> cd tutorial
> cat *.c
```

6. Try it out:

```
> cd samples
> gcc hellovm.c -L/usr/local/lib/x86_64/ -ljit -o hellovm
> cat > gen.pl
#!/usr/bin/perl
open TEST, ">test";
$bytecode="AHello World!\n\x00BC";
print TEST "HelloVM\x00", pack('i',length($bytecode)), $bytecode;
<CTRL-D>
> perl gen.pl
> hellovm test
Segementation fault    <=== OOPS - Fails on Mac OS X! ):
```

3 Write a bytecode jitter! [90 points]

1. Extend `lucax` from the previous assignment to translate the bytecodes to binary code at runtime.
2. Add an option `-j/--jit` to `lucax` to turn jitting on.
3. Add an option `-d/--debug` to `lucax` to turn on debugging output. At the very least, print out whenever an interpreted function gets called and when it gets jitted.
4. You can implement any jitting strategy you want, but some are clearly better than others. In order from best (most points!) to worst:
   (a) Compile functions on demand, i.e. don't compile a function until it's called. [90 points]
   (b) Compile the entire program before execution starts. [80 points]
   (c) Only jit *leaf routines* (functions that don't call other functions). [70 points]
   (d) Only jit programs without functions (i.e. only compile `MAIN`). [60 points]

   Variants of these strategies are, of course, possible. For example, the first time a function is called, interpret it. If the function is called a second time (indicating that, maybe, it's important and might be called again!), jit it and execute the resulting binary code. Indicate in your documentation which strategy you're using.

5. You **must not** simply jit each bytecode instruction by itself, having it push and pop its arguments and results on a stack. Rather, you must compile an entire bytecode function at a time, effectively using registers to transfer arguments between instructions. In other words, you should compile away the evaluation stack that the interpreter uses.

4 Write your own test case! [10 points]

Write a computation-intensive benchmark function. It should do something vaguely useful, consist of multiple functions, and run for a significant amount of time (at least 10 seconds, or so) so that we can accurately measure execution time. You are allowed to share your benchmark program with the other groups in the class so that you can see how your code compares to theirs!

5 Submission and Assessment

The deadline for this assignment is noon, Thu April 7. It is worth 15% of your final grade.

You should submit the assignment to `d2l.arizona.edu`.

You should submit *one* file, ass3.zip, containing all the files necessary to build the compiler and interpreter. Modify the `makefile` so that the grader can build the project by simply typing `make`, and nothing else.

Don't show your code to anyone, don't read anyone else's code, don't discuss the details of your code with anyone. If you need help with the assignment see the instructor or the TA.