Project 2

CMCS 22620/32620, Spring 2004

Assigned: April 21, 2004
Due: May 3, 2004

1 Introduction

The purpose of this project is to write the instruction selection phase for our Minijava compiler. The input consists of a *trace* (see `tracetree.sml`), the output is a sequence of machine instructions (see `asm.sml`). At this point we still assume an infinitely large register file, i.e., we continue to use *temporaries* (see `temp.sml`).

2 Instructions

2.1 Files

Download the file `mj-project2.tgz` from the course web page. This is a compressed tarball containing the Minijava compiler's frontend, the translation to trees, the tree linearizer, the conversion to basic blocks, and the trace scheduler. There is also a new module `frame.sml` which deals with various aspects of the PowerPC stack frame as well as its register file. It will be important for you to read the comments in this file and understand the purpose of each component.

Notice that I have improved the frontend of the compiler to the point that the following things are now handled:

- checking for null pointers where necessary
- checking array bounds
- checking for violations of the subtyping invariant in array assignments
- support for **static** instance variables

As a result, there are some subtle changes to the interface of (among others) the *translate* module. Keep this in mind when comparing it to your solution to Project 1.

2.2 SML/NJ

To get started, the first step is to see whether you have access to a working SML/NJ installation. After downloading, uncompressing, and untaring said tarball, you should end up with a directory named `minijava` containing the source files. Go to that directory and fire up SML/NJ by typing `sml` at the shell prompt. You should see a greeting from SML/NJ and a new input prompt. At this prompt, type `CM.make "minijava.cm";`. The program should compile.

As you make modifications to `cg.sml`, you can re-issue the `CM.make` command (without quitting the `sml` session in between). The SML/NJ compilation manager will take care of recompiling only what's necessary.

2.3 The test driver

Among other things the tarball contains two new files, `compile.sml` and `main.sml`, which implement a test driver for the Minijava compiler. In particular, you can now build a stand-alone version and invoke it in a way reminiscent of invoking a C compiler. However, given that we do not yet have a register allocator, it is currently *required* to specify the `-S` command line option, causing compilation to stop after the generation of assembly code.

To build the standalone version, run the following command from your shell prompt:

```
ml-build minijava.cm Main.main mjc
```

This will create a "heap image" named `mjc.ppc-darwin`. The Minijava compiler can then be invoked using

```
sml @SMLload=mjc -S my-program.mj
```

A successful run of the Minijava compiler will leave a file called `my-program.s` containing the assembly code corresponding to the given Minijava program. Currently the output will not really be legal assembly code because there is no register allocator and names of temporaries appear in place of register names.

3 Handing it in

You should only have to make changes to file `cg.sml`. To hand in your solution, send this file as an e-mail attachment to both the instructor and the TA using the following e-mail addresses:

instructor	`blume (at) tti (hyphen) c (dot) org`
TA	`cysong (at) cs (dot) uchicago (dot) edu`

If you make other changes, then bundle all your files as a tarball and attach that to your e-mail.