

CS126: Introduction to Compilers

Assignment 10: ASMGGEN

Out: 11/21/03

Due: 12/5/03

1.0 Introduction

The final stage of the compiler involves generating assembler code from the optimized and register-assigned intermediate code. Here we assume that the optimizer has been run and the intermediate code has been updated accordingly. We also assume that register allocation has been done, both locally and globally.

2.0 Interfaces

The basic interface that you have to implement for generating assembly code is that defined by `DecafAsmGenerator.generateCode`, an abstract routine that you should implement in your own `AsmGenerator` class.

Your class should extend `spr.decaf.simple.SimpleAsmGenerator`. It should use the `createAsmFile` method to create an `AsmFile` class that provides easy access to writing the assembler file. The routine should do four things:

- First, you should generate a simple header for the asm file. This should include the information that goes at the start of the file (comments, `.file` and `.version` directives, etc.). Look at asm files generated from C or C++ as examples.
- Second, you should generate code for each routine in the program. You can get the list of routines from the `DecafIcode.Icode` instance available as `icode_root`. This is where the bulk of the work lies.
- Third, you should generate a `vtable` for each class. This should go in the `.rodata` section.
- Finally, you should generate the trailer. This includes all constant strings and any trailing comments.

The results of register allocation have been saved as part of the intermediate code structures before the various `generate` calls are made. In particular, for each routine `getTotalOffset()` returns the size of the area needed for temporary storage (for spilling registers); and for each temporary `getRegister()` returns the register assigned to that temporary while `getAddress()` returns the memory address assigned to that temporary. Memory addresses are assigned relative to the frame pointer (register `ebp`).

Once you get all this working, you can then invoke the assembler (as) on the resultant file. If you want to try running your program, you can bind with the library in decafruntime.o. This contains basic implementations of the predefined classes, the routine that creates a string from a C-string, and array allocation routines.