CSC435/535: Assignment 4
(Due: 4 December 2012)

Assignment Description

This assignment asks you to complete your `Cb` compiler, generating ARM assembly language.

Unlike a normal production compiler, assembly language code will be generated directly from the AST. This is reasonable when the target is the ARM architecture because ARM is a RISC architecture implying that the compiler is unlikely to run out of registers and because ARM instructions are similar to IR instructions.

To keep the assignment reasonably small in scope, some shortcuts are appropriate. These are:

- All local variables are held in memory. A local variable must be loaded into a register before it can be used in an expression. Registers r4 to r11 are used for this purpose. At the end of a statement, none of these registers will be in use any longer.
- Parameters of methods will also be held in memory, pushed onto the stack by the caller. (The normal ARM coding conventions use registers `r0`-`r3` for the first 4 parameters.)
- Methods need not accept parameters or return a result with a `struct` type. These are a major complication because `struct` values have arbitrary sizes. All other values in `Cb` are one word (32 bits) in size – they are either `int` values or references to composite data items.
- Register `r12` will have a dedicated use as a frame pointer; it provides access to local variables. (The normal ARM coding conventions do not use a frame pointer.)
- We will not worry about code quality ... it is far more important that the generated code work than that it be efficient.

Notes on the Grading Criteria

Implementing all of the above so that the generated code works in all normal circumstances is worth an A grade. The difference between A and A+ (which is supposed to mean exceptional performance) comes from handling `struct` types in method calls or from implementing a fairly complete set of static speephole optimizations or from avoiding the use of memory for parameters and local variables (combined with a reasonable register allocation strategy). You don't have time to attempt more than one of those! And there's no credit for this extra work unless the compiler is generating correct code. Fast incorrect code is not at all useful.

The Provided Materials

- You have to continue from your Assignment 3 code. The files listed below should be incorporated into your compiler.
- The supplied source code files are listed in the table below.

File	Description
`cbc.cs`	The main program which invokes everything else. It has been updated to invoke the code generation and assembly language output.
`GenCode.cs`	A class which traverses the AST to generated ARM assembler code. It contains methods for traversing different parts of the AST; these methods contain switch statements which should have cases for all node tags that can appear.
`Location.cs`	Defines a `Loc` class which defines how a memory location is accessed in the generated ARM code.
`ARMAssemblerCode.cs`	Defines a class used for building up the ARM code in memory. These leaves open the possibility to sweep over the code performing peephole optimization before the code is output.
`CbRuntime.s`	An ARM assembly language file which contains various utility functions for I/O, integer division, etc., which the C♭ program may need to invoke. [**NOTE**: This file is not ready yet. Coming soon!]

- Additional documentation:

`CFlatCodePatters.pdf`	Contains suggestions about the code patterns to generate.
`ARMReference.pdf`	A summary of the ARM instruction set

- Testing/running ARM assembly language programs. These run on Windows. They are not yet uploaded to conneX but will be there soon.

`as.exe`	An assembler for ARM code; it can be used for verifying that the code generated by a C♭ compiler is syntactically correct and links properly with the supplied `CbRuntime.s` file.
`ARMSim#`	An ARM simulator developed in our department.

Submission Requirements

These are much the same as for assignment 3!

1. You must provide exactly one file. It must be a zip file or gzipped tar file which contains all the source code files needed to build your program. If you added more C# files to your project, include those too.
2. Also include a file named `README.txt` which identifies the team members. If you have any comments you want to share about problems with the assignment, this is an appropriate place to supply the comments.
3. Important: do *not* include any files generated by gplex or gppg in your submission, we want the `.lex` and the `.y` files to be submitted.
4. The project is to be completed in teams of either 2 or 3 persons. The ideal size is 2 people. The teams do not have to contain the same members as for Assignment 3. (But given that assignment 4 is built on assignment 3, it would probably be difficult to switch members now.)

Final Note

Same as for assignment 3!

No guarantee is provided (or can be provided) that the supplied code is 100% correct. You become responsible for all the code which you include in your compiler project.

If you find any errors or deficiencies in the supplied code, please report them and they will be fixed as soon as possible. A separate document named `Corrections.txt` will then appear in the assignment 4 folder on conneX which describes the problem and its correction. That file will grow as needed. No special announcement will be made unless it is a serious problem that you would have trouble fixing without assistance.