In this last part of the assignment, you will generate 32-bit x86 code for `iplC` programs. For initial ideas regarding code generation, you can read section 8.6 of the main text book. This is titled *A simple code generator*. The primary requirement of your code generator is correctness. The next requirement is completeness—all features should be implemented in a general sense. For instance, your code generator should not fail because an expression was too large and you could not find enough registers to hold intermediate values during its evaluation. Going beyond these may earn you bonus points. If you can argue or, even better, demonstrate that your code generator produces more efficient code, that will earn you bonus points. I shall later inform you regarding the criteria for awarding bonus marks.

After having studied intermediate code generation and runtime environments, you now know enough to generate simple but *correct* code for `iplC` programs. The lectures on code-generation will focus on generating *efficient* code. Please start off as soon as possible. As usual, I may keep adding to the assignment description for further clarifications.

Following are the resources that you may want to use for code-generation:

1. Quickly check out whether the gcc compiler on coffre can seamlessly produce 32-bit executables with the `-m32` option.
2. Use the `mygcc` alias provided to you generate clean and compact assembly code.
3. Use `.gdbinit` from `https://github.com/cyrus-and/gdb-dashboard` to configure your `gdb`, so that you can single-step through the program.
4. You should get your ideas regarding the code to be generated by compiling gcc programs and observing the assembly output. However, if you want to refer to a document, here are some:
   (a) A Tiny Guide to Programming in 32-bit x86 Assembly Language:
   `https://www.cs.dartmouth.edu/ sergey/cs258/tiny-guide-to-x86-assembly.pdf`
   (b) A larger x86 Assembly Language Reference Manual:
   `https://docs.oracle.com/cd/E19641-01/802-1948/802-1948.pdf`
   (c) A document by your friendly TA, Atul Sudharshan Sainath, on floating point instructions and their usage.