Assignment 3

CS3310 Compiler Design Lab

due October 5, 2015 at 23:55 on moodle

Code generation for a reasonably ~~large~~ small subset of C. Given a C program, generate assembly code (.s file). Use the x86 assembler to generate the executable code and run it.

Instructions

1. Download CS00B000.tar.gz to a directory, go to the directory in a terminal
2. tar xvzf CS00B000.tar.gz.
3. mv CS00B000 CSxxB0xx # use your roll number
4. cd CSxxB0xx
5. make # compile given demo program
6. make test # run it on testcases
7. make tar # generate .tar.gz for submission

Details

1. Use C language only, and lex-yacc tools.
2. The language constructs include declarations, assignments, function calls and comments. Data type supported is int (variables and constants). **No** if, for, while, int array, char, char *, float, nested blocks etc. The only non-integer would be the format string of printf. The function with arguments would be printf.
3. Declaration for all variables will be done together, before any assignment. Declaration and assignment will be done in separate instructions (int x = 10 is disallowed. Instead, it would be int x; x = 10). There can be local variables inside a function.
4. There will be no “#include <>”, no global variables, no static variables.
5. Following operators should be supported: + and -. Although we don't support unary minus (-5), the value of an expression may be negative (4-9).
6. There will be no function declaration (void fun();) but there would be function definitions. All the functions will be defined before main. If f1 calls f2, f2 would be defined before f1. This means recursive programs are not supported. Return type of every function and main will be void. Parameters will not be passed to functions, except printf.
7. The format string would not contain double quotes “ as a valid character. But it may contain \n and \t.
8. Comments start with // and end with newline.
9. There can be maximum 1000 variables in the program but there could be unknown number of temporaries that may get generated.
10. We will follow scope rules for C. Variables with the same name may be redeclared in a different scope (which is essentially a different function. No globals are supported). If they are declared in the same scope, it would be ERROR.
11. The language would be strictly typed (function cannot be assigned to int, for instance).
12. There are ten public testcases along with expected answers. You are expected to

output only “ERROR” (as a substring). Check your program on the testcases using: make test. Also a “.s” assembly code file is provided for your reference. You can generate the same using '-S' option in gcc for your C program (gcc -S 1.c). Fifteen more private testcases would be used for evaluation.

13. Your code should be able to run on some DCF machine. You are supposed to take appropriate measures. It will be evaluated on the DCF machine. We will not entertain gcc version mismatches or OS version mismatches.
14. No optimizations need to be done.
15. The evaluation would run a makefile to run your lex-yacc parser a.out. a.out will receive a testcase C program (as in previous assignments) which would produce x86 assembly code on stdout. We will redirect this output to a .s file, run assembler to generate testcase b.out, and run it. The output of b.out should **exactly** match the expected output. The expected output can be obtained by running gcc on the testcase C program to generate c.out and running it. You don't have to worry about which file to name what -- we will provide you with a3.l, a3.y and a makefile. Simply print assembly to stdout.
16. **Assembly learning:** You will need to learn about movl, addl, subl variants for expressions and assignments. Variables can be allocated with adding (or subtracting) an offset to (from) the stack pointer. Function arguments can be pushed using pushl instruction. A call instruction would be needed for function calls. Format strings need to be defined specially as LC0, LC1, etc. There would be a standard header + footer you will have to print to create the assembly code in the right format.
17. **Output:** x86 assembly code (on stdout, not in .s file). Generate temporaries as you generate instructions and allocate all of them on stack at once by updating %esp at the start of the function (along with other local named variables). This way you don't have to worry about data registers.

Submission

Submit a single tar file as ROLLNO.tar.gz which extracts to directory ROLLNO which contains all the relevant lex-yacc files, and Makefile . You can use the following command to create it: *make tar*
After the submission, it is your responsibility to download your own .tar.gz from moodle and see if it extracts properly. If there are issues, resubmit on moodle. If for some reason, there are issues with moodle, you have to make sure your send your assignment to either the instructor or any of the TAs before 23:55 on the submission date. We will not entertain requests such as “Sir, I uploaded wrong file” or “I forgot to include one file in the archive”.

**Please follow the instructions properly otherwise your marks will be deducted.
~~Also, this is a relatively larger assignment, so you may want to start early.~~ You are already late.

Notes

- **Declarations:** type varlist;
- **Assignments:** id = expr; // need not support id1 = id2 = id3;
- **Function call:** id(); or printf(...);
- **Function definition:** void id() block
- **Comment:** //...