CS 3432 – Computer Organization

Lab 3 - Expression Compiler Backend for RISC-V Author: Shirley Moore

**Instructions:**

Assignments must be submitted with the same person as in the previous lab, unless the other person drops the course, through Blackboard. Only assistance from your teammate, instructor, TA, or IA will be permitted. If working with some other team, only share ideas but not implementation because we are checking potential plagiarism.

**Introduction:**

A compiler typically has frontend and backend components. The front end parses a source code program and produces an abstract syntax tree (AST) representation of the program. The backend traverses the AST and outputs assembly or machine code instructions for a specific ISA. Other actions may also be carried out, such as transformations of the AST to implement optimizations such as strength reduction and common subexpression elimination.

For this assignment, you are given a build\_tree() function that takes a C R-value expression as an argument and returns a pointer to an AST. That is, you are given a front end for a mini compiler that handles a subset of C expressions. An R-value is an expression that can appear on the right-hand side of an assignment statement. Your job is to write a backend that traverses the AST and outputs RISC-V assembly code.

**Task:**

1. [10 pts]. You are given files main.c, build\_tree.c, build\_tree.h, backend.c. The file main.c contain the main() function. Construct a Makefile that:
2. Creates an executable when “make” is run.
3. Run the executable when target “run”

To run the code, type

./compile

and then enter an expression and press return. To stop entering expressions and quit the program,

type Ctrl-D for end-of-file. You can also redirect input from a file that contains expression strings one

per line, e.g.,

./compile <input1a.txt

1. [10 pts] Write a function that takes an unsigned integer as an argument and determines if that integer is a power of 2. If the integer is a power of 2, the function should return the logarithm to the base 2 of the integer, otherwise, the function should return 0. Write a driver program to test your function and test your function thoroughly to make sure it is correct.
2. [10 pts] Compile and run the code in build\_tree.c. Examine the code and explain how it works. Note how the postorder() function traverses the AST and outputs a postfix expression. Note that currently the code allows only single-letter variables using the first 10 lower-case letters. You may remove this restriction if you wish, but you are not required to do so.
3. (20 pts) Write a backend that traverses the AST and outputs a RISC-V translation of the expression. You may use register x0 as needed and you may use registers x5 through x31 to hold variable values and intermediate results. You should implement strength reduction by transforming multiplication by a power of two to use a shift instruction instead of a multiplication instruction (the function you wrote for part 2 above will be useful here). Note that you will need to convert the unary negation and NOT operators to binary operators since RISC-V does not have any instructions for unary operations.
4. (20 pts) Add code to the generate\_code() function to handle expressions that contain constants. For binary operations, the code you are given handles the case where both the left and right child of the operator node and REG type nodes. You will need to handle the following additional cases:

* The left child is REG type, and the right child is CONST type.
* The left child is CONST type, and the right child is REG type.
* The left child is CONST type, and the right child is CONST type.

For unary operations, the code you wrote for part a handle the case where the operand of a unary operator is a REG type node. You will need to add code that handles the case where the operand is a CONST type node.

1. [10 pts] Write code to handle immediate values that are larger than 12 bits.
2. [10 pts] Implement strength reduction for multiplication by a power of 2 by replacing the multiplication with the correct left shift instruction.
3. [5 pts] Add additional test cases to thoroughly test your code and show your testing results.