CMSC313 Project 4
Due 5/14 11:59pm

Design a compiler for a processor that supports the following Assembly instructions:
- ADD R0, R1: R0=R0+R1
- MOV R0, imm: R0 = imm
- MOV [rbp+offset], R0: Store R0 into memory location of rbp+offset
- MOV [rbp+offset], R1: Store R1 into memory location of rbp+offset
- MOV R0, [rbp+offset]: Restore R0 from memory location of rbp+offset
- MOV R1, [rbp+offset]: Restore R1 from memory location of rbp+offset

The compiler allows users to write code in C instead of Assembly and generates the corresponding Assembly code.
Create a compiler in C++ that reads a C file at cmsc313_proj4_input.c. Each line in the C file will be of the following format:
- <VAR> = <imm>;
- <VAR1> = <VAR1> + <VAR2>;

For example, the following instructions are possible:
- x = 3 ;
- y = 4 ;
- z = x + y ;
- tempVar = z + y ;

The compiler should output in Assembly to the screen, separating each Assembly instruction in its own line.

You can break down the compiler into the following operations:
- Read a line from input file
  - std::ifstream file("cmsc313_proj4_input.c");
  - while (std::getline(file, line)) { … }
- Identify the C instruction. For each C instruction, have a sequence of Assembly instructions that can be used to implement it. For example, "<VAR1> = <imm>" C instruction is implemented by "MOV R0, imm" Assembly instruction.
  - Check if line contains '+' symbol to identify the instruction:
    - if (line.find('+') != std::string::npos) {
- Identify the variables/immediate values used in the instruction
  - If words is a vector of string variables that splits each line, then words[0] is the destination register, words[2]/words[4] are the input registers for the addition instruction. words[2] is the immediate value for the immediate load instruction.
- Keep a unordered map (key/value pair) that stores the C variable name as key and stack offset location as the value.
- Keep a variable that points to the number of stack locations used (i.e., number of C variables). Assume that variable needs to be saved for the entirety of the program.
- For each C instruction, break it into the following sections:
  - Retrieve the input variables from stack memory into registers. Load immediate value into registers. Write out separate Assembly instructions to perform each of these operations.
  - Perform the sequence of Assembly instructions that is equivalent to the C instruction. Write out the sequence.
  - Store the output into stack memory. If the variable already exists (i.e., key exists in unordered map), then write back to that offset. If variable doesn't exist, create a new location (based on the variable that keeps track of number of stack locations used)

Some additional notes:

- Submit cmsc313_proj4.cpp file. Compile your code using the following command and confirm that it works as expected:

g++ -o cmsc313_proj4 cmsc313_proj4.cpp

- There are only 2 registers in this processor. If you have more than two variables, you have to offload some of the variables into stack memory (rbp+offset). After the register has been stored, the register is free to use for a different purpose. When the variable is needed again, read back from stack memory to retrieve its contents.
- For simplicity, assume that any stack offset>=1 is available for use by the compiler and each variable can be stored in one memory location (i.e., memory stack, variables, registers are all 8-bit wide).
- The compiler can either try to reduce the number of assembly instructions needed or it can be a fast compiler (i.e., reduce compiler complexity and save all variables to stack whenever it changes so that registers are always available). You can choose the fast option to simplify the project.