Name: _____

Lab for ITSC 3181 Introduction to Computer Architecture, Spring 2023

Lab #01: Compiling a C program to produce its assembly output. Due Wednesday 1/18 and Total points: 1% of the final accumulated percentage grade.

The development and lab environment https://passlab.github.io/ITSC3181/resources/devenv.html

1. Open https://repl.it/languages/c from a web browser. The left side of the interface is for you to type in your program. The Linux terminal environment is on the right side of the web interface, and get familiar with yourself using Linux commands from command lines (https://passlab.github.io/ITSC3181/notes/lecture00_LinuxBasicsCompilation.pdf). Inspect the output of the commands you type for execution.
2. Try "gcc –help" and "man gcc" command, which show how to use the gcc compiler to compile high-level program to machine code.

```
yanyh@vm:~$ gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...]
                           Display specific types of command line options
  (Use '-v --help' to display command line options of sub-processes)
  --version                Display compiler version information
...
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
```

3. Create hello.c file from either the terminal (using vim editor or other editor that you are comfortable with) or from the https://replit.com/languages/c web interface, and type in the content of the file:

```c
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv) {
  printf("Hello World\n");
  return 0;
}
```

4. Compile and execute hello.c program.

   gcc hello.c -o hello
   ./hello

5. Create bubble.c file and type in the content:

```c
void bubble_sort(int list[], int n) {
  int i, j, t;

  for (i = 0 ; i < n - 1; i++) {
    for (j = 0 ; j < n - i - 1; j++) {
      if (list[j] > list[j+1]) {
        /* Swapping */
        t          = list[j];
        list[j]    = list[j+1];
        list[j+1] = t;
      }
    }
  }
}
```

6. Compile the bubble.c file with -S option, which generate a bubble.s file for the ISA architecture of the machine (x86). Check the content of the bubble.s file. **Note: those lines with symbols that start with . (e.g. .file, .text, .cfi_startproc), the label lines (those that end with :, e.g. bubble:, .LFB0:), or those lines for function signature or comments are NOT instructions.**

7. Explore other ISA assembly from Compiler Explorer at https://godbolt.org/ for the bubble.c example.

```
yyan7@yyan7-Ubuntu:~$ vi bubble.c
yyan7@yyan7-Ubuntu:~$ uname -a
Linux yyan7-Ubuntu 5.4.0-72-generic #80~18.04.1
yyan7@yyan7-Ubuntu:~$ gcc -S bubble.c
yyan7@yyan7-Ubuntu:~$ cat bubble.s
        .file   "bubble.c"
        .text
        .globl  bubble_sort
        .type   bubble_sort, @function
bubble_sort:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movl    %esi, -28(%rbp)
        movl    $0, -12(%rbp)
        jmp     .L2
.L6:
        movl    $0, -8(%rbp)
        jmp     .L3
.L5:
        movl    -8(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %edx
        movl    -8(%rbp), %eax
        cltq
        addq    $1, %rax
        leaq    0(,%rax,4), %rcx
```

Submission: Submit your work using the submission page that you can find from canvas for this lab. The submission page has the following table. You need count the number of instructions for the specified compiler and ISAs for bubble.c program and input the counts in the table. Labels (those ends with :) and directive (those starts with .) are not instructions, so do not count them. Instructions are normally highlighted with color and you can apply Filter in the right-side Pan of the web interface to sort out so only instructions show. You should count manually the number of instructions from the output when selecting the specified compilers from https://godbolt.org.

	Compiler and ISA	Number of instructions
https://godbolt.org/	RISC-V rv32gc gcc latest	
https://godbolt.org/	MIPS64 gcc 5.4	
https://godbolt.org/	x86-64 clang 12.0.0	
https://godbolt.org/	ARM gcc 8.2	
https://godbolt.org/	RISC-V rv32gc clang(trunk)	
https://godbolt.org/	RISC-V rv64gc clang(trunk)	

The study shows that for the same high-level program, the instruction sequence for different compiler and different machine architecture (represented by its ISA) are very different. Even for the same ISA, the instruction sequences vary from different compilers. Instruction sequences also vary between 32-bit and 64-bit machines of the same ISA and of the same compiler.