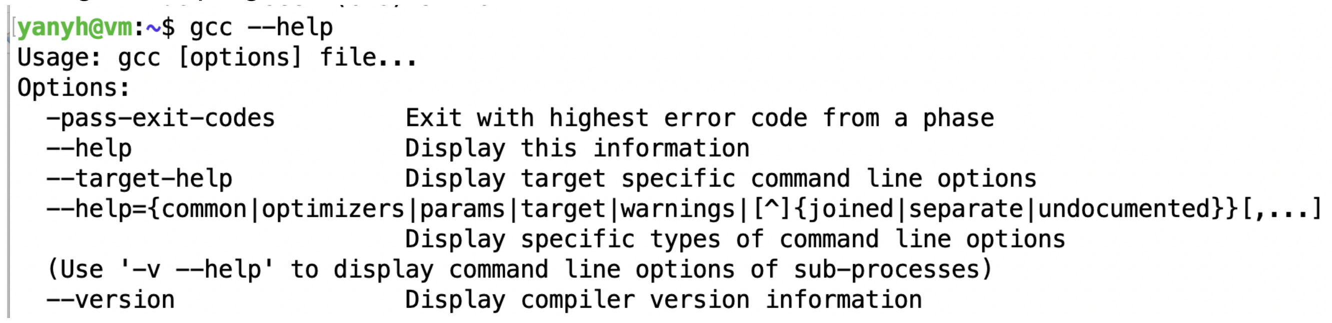
Class Exercise Activity for ITSC 2181 Introduction to Computer Systems
Module 6 - Unit 1: Instruction Set Architecture and RISC-V Assembly Programming

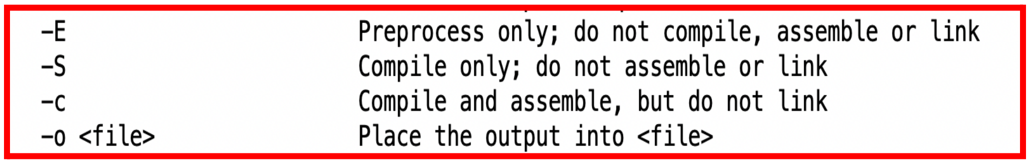
Compiling a C program to produce its assembly output.

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. You can use the virtual machine for this class to do the same thing.
2. Try the "gcc -help" and "man gcc" commands, which show how to use the gcc compiler to compile high-level programs to machine code.



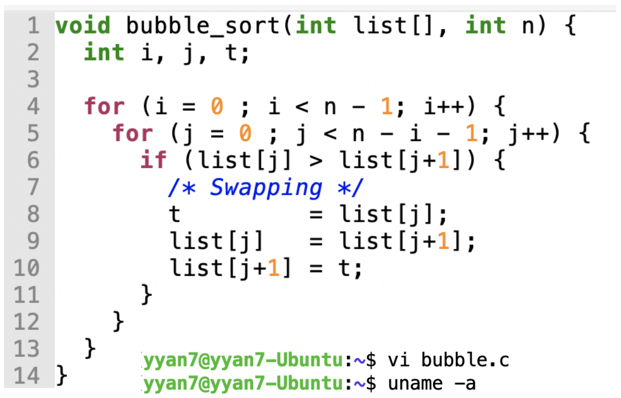
```
yanyh@vm:~$ gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...]
                           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>
```

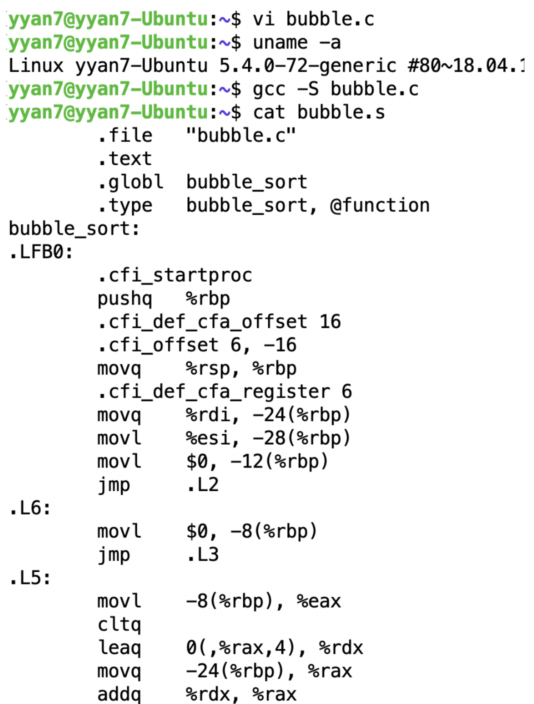
3. Create bubble.c file and type in the content:



```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;
      }
    }
  }
}
```

4. Compile the bubble.c file with -S option, which generates 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.**



```
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
```

5. Explore other ISA assembly programs of bubble.c from Compiler Explorer at https://godbolt.org/, and count the number of instructions used for each ISA. You need to count the number of instructions for the specified compiler and ISAs for the bubble.c program. 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.

Submission: Submit your work by writing in the textbox the instruction count of each ISA, one line for each ISA. To do that, copy the following bold content in the text box of the submission, and replace xxx with the number you count for each ISA.

Compiler and ISA,	**Number of instructions**
RISC-V rv32gc gcc latest:	**xxx**
MIPS64 gcc 5.4:	**xxx**
x86-64 clang 12.0.0:	**xxx**
ARM gcc 8.2:	**xxx**
RISC-V rv32gc clang(trunk):	**xxx**
RISC-V rv64gc clang(trunk):	**xxx**

The study shows that for the same high-level program, the instruction sequence for different compilers 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.