

OPERATING SYSTEMS

ASSIGNMENT 0.2 COMBINING C AND ASSEMBLY LANGUAGE PROGRAMS

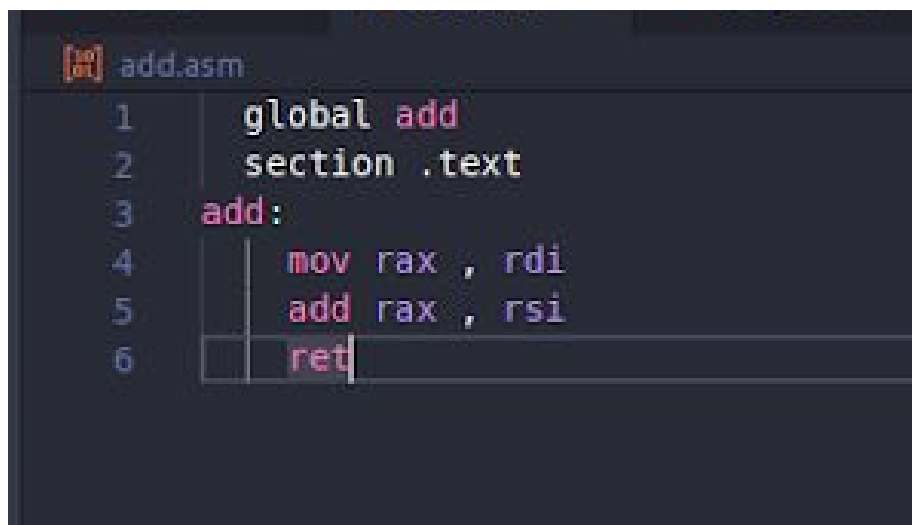
SUBMITTED BY
DOLLY SIDAR (2019304)

THE STEPS

WORKING

C and Assembly can be used together to provide extra flexibility.

The code starts with the user input first integer and a second integer. Then the program calls the function `add()` to perform the addition of two integers. The function takes two parameters with return type `int64_t`. And is written in assembly language with file name `add.asm`; the two arguments are passed in registers reading a C argument from left-right. The arguments passed in a specific register, i.e., first arguments will pass in **rdi**, second will be in **rsi**.



```
[A] add.asm
1  global add
2  section .text
3  add:
4      mov rax , rdi
5      add rax , rsi
6      ret
```

As we can see above, first it moves the value stored in **rdi** register to destination register **rax** by using opcode `mov` and then adding with the help of **add** opcode to add number store in **rax** and **rdi** registers. The result will be stored in **rax** because integers are returned in **rax** and **rbx**, and also, while adding **rax** is our destination register. At last, we have to return the value in **rax**. Then the main function of `prog-add.c` will print this sum that has been computed by the `add` function in assembly language.

COMPILING

Compiling using make file.

1. The first step is to compile the `.asm` file using target `compile_nasm` in Makefile, followed by the commands to compile `.asm` file using NASM, an assembler for the Intel x86 architecture.

```
nasm -f elf64 add.asm -o add.o
```

This command compiles the assembly source to an object file (written in binary) add.o. The -f elf64 option instructs NASM to assemble the code for a 64-bit target(x86_64) and automatically stop the process.

2. The second step is compiling the .c file using target compile_C in the make file, which contains the commands to compile .c file. It simply follows the compiling stage, which I have mentioned in assignment 0.1 and then creates an object file with .o extension and stops the process by using the -c flag.

```
gcc -c prog-add.c -o prog-add.o
```

3. The third step is linking, which links the assembly and C code with the GCC command. Linking will create the final executable file. Here linking plays a vital role, connecting all the source files, all the other object codes in the project. Like in this program, we first compile the assembly code and then get an object code after compilation; similarly, we did it for C code. But we have to make one single program, and linking will link the add.o and prog-add.o to one single executable code.

```
gcc add.o prog-add.o -o final
```

4. The last step is to run the executable file using command ./final. final is the file which is created in the above command. Using an output target in makefile to run this command.

```
./final
```