# CSE231: Operating Systems

Assignment 0.2

**Name:** Mihir Chaturvedi
**Roll no.:** 2019061
**Branch:** CSE

---

# nasm Command Line Options

1. **`-E` flag:** Stop after preprocessing

   This flag halts the assembly process after the preprocessing of the assembly file. The preprocessed file output is either sent to the standard output (`stdout`), or the the output file, specified in the command using the `-o` flag.

2. **`-f/--format` flag:** Specify the format of output file

   This flag specifies the output file format we require. The file format may be different depending on the system architecture and operating system.

3. **`-elf64`**

   This is the format of the output file we have specified using the -f/--format flag. As mentioned in the documentation of nasm, -elf64 refers to ELF64 (x86_64) object files (e.g. Linux). “elf” stands for “Executable and Linkable format”, and the “64” specifies the word-size architecture (in this case, 64-bit architecture).

4. **`-o` *file*:** Place output in *file*

   This flag is used to specify the output file of the command, and whatever the command produces as output. In some cases, the primary output stream is the standard output (stdout) which prints out the contents in the terminal itself. We can route the output into a file using this command.

# Output File Descriptions

There are eight files that we need to deal with:

| | |
|---|---|
| prog-add.c | User-written C code |
| add.asm | User-written ASM code |
| prog-add.i | Intermediate/preprocessed C code |
| add.i | Intermediate/preprocessed “pure” ASM code |
| prog-add.s | Compiles assembly code |
| prog-add.o | Assembled object file of prog-add.c |
| add.o | Assembled object file of add.asm |
| prog-add | Final executable after linking C and ASM objects |

In the following, I’ll be discussing the stages relevant only to user-written assembly code, i.e., add.asm. For prog-add.c, I have written in depth about it in Assignment 0.1.

1. Preprocessing stage

   As with the gcc compiler preprocessing C code, the nasm compiler preprocesses the assembly code by removing comments “purifying” it further. Assembling with nasm supports “macros” which are processed using the preprocessor. A lot of the details are mentioned here: Chapter 4. The NASM Preprocessor

2. Assembling Stage

   The preprocessed assembly code needs to be converted into “machine code”, which is a binary that only machines understand (doesn’t have standard text encoding). The assembly files are assembled and an “object” file is created.

   Conventionally, and by default, these object files have the extension ‘.o’.

3. Linking Stage

   This is the final step of the entire compilation process. Here, we link the object files received from the gcc compilation and nasm assembler. Multiple object files, that might contain references to each other, need to be “linked” to produce one, united file that can be readily and immediately executed by the system.

# Explanation

- In the user-written assembly file, add.asm, we create a subroutine “`add`” which will be referenced by the C code as a function with a name as `add()`.
- The file-format we choose to assemble uses 64-bit architecture, and the registers we receive return and receive in the subroutine ‘add’ are both 64-bits long.
- In the C code, the function ‘add()’ has the signature:

  ```
  int64_t add(int64_t, int64_t)
  ```

  This indicates that the return type of the function will be a “long int”, and the function takes two parameters that are too, both of “long int” types. These are in parity with the lengths of the rax, rdi and rsi registers in the assembly code.
- The linking of the object files enforces a connection between these two functions.