The Assembler and Linker: Dynamic Duo

In this document I will be referrring to the assembler in regards to the entity that ingests files written in assembly and is then assembled into an object file. Our assembler is the NASM assembler, which is the Unix standard assembler. When using a Windows based operating system, there is a similiar tool called MASM. These two focus on accomplishing the same goal but on different systems. If you have a Windows computer, it is recommended that you either run Linux in an isolated Virtual Machine or configure your Windows installation to have a Ubuntu installation run parallel wiht your kernel. <— fix this because it sounds complicated as fuck.

This object file is useless unless linked by the linker. When we examine what exactly an object file is, the file command tells us this:

```
$ file main.o
main.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),
    not stripped
```

We can see that it is an 64-bit execuatable that can be relocated in memory and is unstripped. This binary (locally compiled code) contains symbols that can be interpreted by debuggers such as GDB to step through program flow. Such symbols may be ommitted from code that does not require debugging, for example production code.

Step 1: Compiling

The first step is using NASM to do the assembling, which in our examples looks something like this:

```
nasm -g -felf64 file.asm -l file_listing.lst -o file.o
```

Revisiting the topics of symbols, our -g flag tells NASM to include debuggin symbols. If ommitted, GDB would be unable to debug. The flag `-felf64` indicates that we wish to produce an ELF 64-bit executable, which can be see in the examniation of an object file section. `-l` means that we create a separate text file that contains a memory map of the program, using predefined memory addresses. Lastly, `-o` sends the output to a specifed file, generally of a .o extension.

Step 2: Linking

There are two different linkers we can use; gcc or ld. If your code requires external C functions, please use GCC. In fact, you can even resort to using GCC as the assembler, linker and executioner in one command. However, this will not

be covered until later sections. For us, we will use ld and it gets the job done. This part is simple and requires no explanation

```
ld file.o -o file_exectuable
```

Remember that each time you want to run your program, you must run these commands in order. I have provided a shell script that does this task for you and is available here. I will assume no prior knowledge of Bash scripting so I will include a separate section covering this topics.