Assembling, linking and debugging your assembly language program under Linux.

Assembly language programs are created using a text editor. The traditional naming convention is that assembly source files should have the file name extension ".s". Suppose you create a file called myprog.s. In order to turn this into an executable program, the following steps need to be taken:

as -g myprog.s -o myprog.o

where:

"as" is the GNU assembler program
"-g" tells the assembler to include debugging information in its output
"-o myprog.o" tells the assembler to name the output file "myprog.o"

If the assembler is successful, a file called myprog.o is created. This file is called an *object file*. Object files contain machine code (numeric) instructions, program and data labels as well as other information. The file is not quite yet a runnable program. In order to become runnable, numeric values need to be associated with program and data labels (so that jumps and reads/writes will actually work). These labels are fixed up using the linker. A program is linked in linux as follows:

ld myprog.o -o myprog

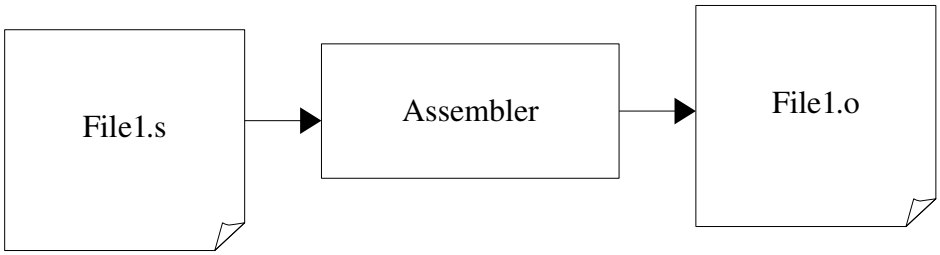
where

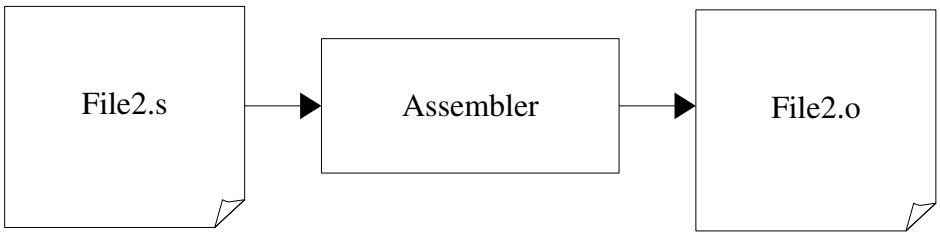
"ld" is the linker program
"-o myprog" tells the linker to name the resultant executable as "myprog"

If all goes well, the file "myprog" should be executable by simply typing "./myprog" (Note: "./" represents the name of the current directory).

Where does the linker program get its name?
The ld program is called a linker because it can be used to link or join together one or more components into a single executable file. The linking process can be thought of like this:

Assuming your program source is split across two source files, the assembly process is as shown below:





File1.o and File2.o will contain machine code, labels (non fixed-up) and indications of the size of code and data associated with each program module (file).
The object files might look like this:

<u>File1.o</u>

Data section
Data size 404 bytes
labels:
Array1 : offset 0 bytes
I : offset 400 bytes

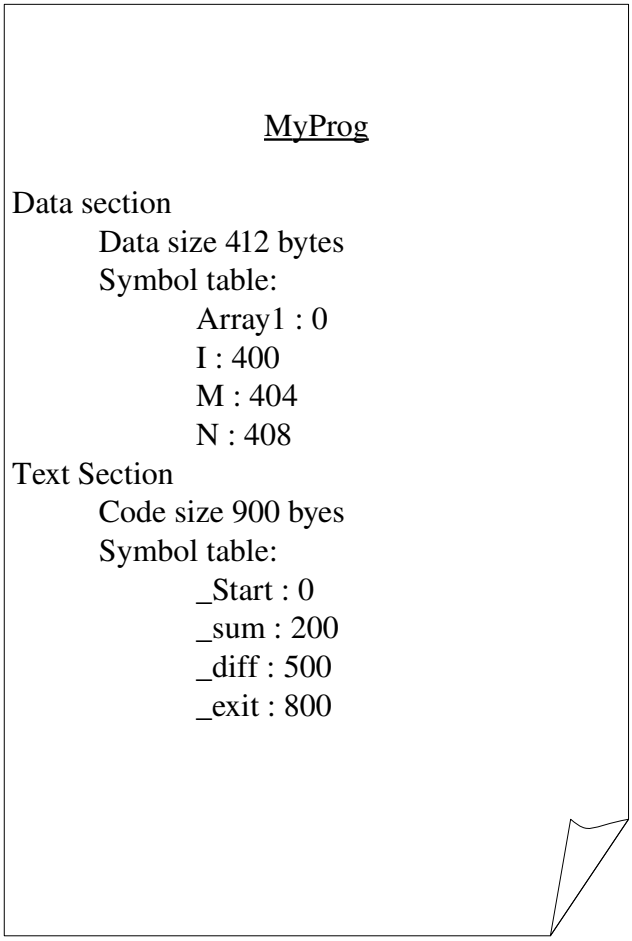
Text section
Code size 500 bytes
labels:
_Start : offset 0
_sum : offset 200

<u>File2.o</u>

Data section
Data size 8 bytes
labels:
M : offset 0 bytes
N: offset 4 bytes

Text section
Code size 400 bytes
labels:
_diff : offset 0
_exit : offset 300

The linker now joins those two files into a single executable whose structure is similar to this:



As can be seen, the linker, apart from simply merging the code in the two object files, also "fixes up" or associates numerical addresses with the symbols in the object files. Statements such as

```
call _sum
```

can now be replaced with

```
call 200
```

(note the "call" mnemonic was replaced with the appropriate numeric instruction by the assembler previously).

The assembler alone is unable to perform this symbol fix up as it does not know the size the data and code blocks within other program modules.

The above explanation is a little over simplified in places however it does illustrate the sorts of tasks the linker must perform.