SFWR ENG 3F03 Summary

Author: Kemal Ahmed
Instructor: Dr. Ned[2]
Date: Winter 2014

Please join GitHub and contribute to this document. There is a guide on how to do this on my GitHub.

Terminal

Bash Files

%Command% -%option(s)% %target%
you can stack options, too
e.g. list files in directory: ls
options:

- r: reverse order
- l: display modified date and permissions
- t: sort by modified date
- a: hidden files
- more on the man page

all options:
ls –ltr Documents/Dropbox

%command% man: gives the man page, which explains what the command does and hopefully all the options and what they do.

$pwd: can replace the target directory with the current directory

mv: moves file, a.k.a. “cut”

If you push any data, you MUST pop it, since terminal does not have the garbage collection necessary to avoid that.

Remember SFWR ENG 3GA3? When calling a label (i.e. function), the address of the label will be pushed onto the stack and saved in RA (Return Address). “Return” will pop the return address from the stack.

Makefiles

Makefile is a utility that compiles a set of source code (usually a project), like a shell-script extension, using bash commands. This is especially useful for large projects. Use Makefiles to automate removing libraries from directory before using SVN. It is useful for any type of source that can be compiled with bash, but the most common application is compiling C/C++ source code.

If you have multiple files in a project that refer to one another, such as objects, you usually compile them into their individual object files. However, they don't actually refer to anything other than symbols, until you use an overall compiler to join them into one main executable. Makefiles can compile and will also join the files together.

Whatever format you use to save the source for the Makefile, you'll need to compile that, using the command:

```
make -f MyMakefile
```

Similar to bash, signify different elements in a list using spaces

Declare:

- LIBS: folders
- INCLUDE_PATH:

Simply running the make command in the shell will run the first target in the file.

Makefiles have executable sections, similar to functions, known as **targets**. Each target is saved as an executable, so when you run a target, it'll only run it if you don't have a file that already has that name. When you execute a target, you run off a directory. To execute a target from the shell, run:

```
make target
```

However, if you want a function that does not save as executable, such as "all", at the very top above the variables, put

```
.PHONY all
```

If you have multiple executables you want to compile simultaneously, place a target called "all" at the top of your file that refers to each of your targets (one for each project)

Access Makefile variables with $(var_name)
Access bash variables with $$var_name

Compile using: gcc -c *.c –I../
This means: use the c-compiler from gcc on all c files (although you could just specify which file(s)) that are **I**ncluded in the given directory

The compiled files have the extension .o
You'll need to move all the object files to one central location before you make your executable from them.

You should also include a section to remove your previous files, called “clean”

.inc represents a library

driver.c calls assembly files

Processor Information

There are two main processors we’re going to look at:

- 8086
- 80386+

Registers

8086 16-bit registers:

- General Purpose: AX, BX, CX, DX
    - AX = [AH|AL] (higher and lower part)

80386+ 32-bit registers:

-

Assembly Files

Learning to program in assembly helps gain a deeper understanding of how computers work. It also helps better understand compilers and high level languages, such as C.

We’ll need to use a C driver to call assembly functions.