CS 218 – Assignment #1

Purpose: Become familiar with the tool chain → the assembler, linker, and debugger. Also become familiar with the operating system and a text editor (of your choice).
Points: 15

Assignment:
Learn to assemble, link, and utilize the debugger with a provided program.

- Create a working directory where the working files will be placed (**Home Folder** → right click and select **Create Folder**). *Note*, you may opt to choose a cloud storage option, which will mean using a different directory.

- Download the assignment #1 assembly language program from the class web site into the working directory.

- Edit the provided file to include your name, assignment number, and section number. This information should be included on all assignments. *Note*, if you wish to use *emacs*, you will need to install it first (from the Ubuntu Software Center).

- Start the terminal (**Dashboard→ Terminal**) and inside the terminal, navigate to the directory when the asst01 file was placed. The **`ls`** (list files) and **`cd <dirname>`** (change directory) commands will be useful. Refer to the *UNIX Terminal Command Line Summary Sheet* (on the class web page) for information on additional terminal commands.

- Assemble the program using the provided makefile. Use the following command:

  ```
  make ast01
  ```

  The make file will assemble and then link the program. Note, the file must be named **`ast01.asm`** which is expected by the make utility.

- Execute the program in the debugger. Use the following debugger command

  ```
  ddd ast01
  ```

  You will probably want to display the line numbers (Source → Display Line Numbers)

- Start the DDD debugger
  - Execute the program in the debugger.
    - Set a break point at the end of the program.
    - Page down the line number of first instruction after the label last (~ line 188), right click, and select set breakpoint option. You will see a "Stop" sign (on the right) when the breakpoint is set. Alternately, you can type "break last" in the bottom window at the (gdb) prompt.

  - Run the program.
    - Click on the **Run** option of the pop-up DDD menu. Alternately, you can type "run" in the bottom window at the (gdb) prompt.
    - Note, you must set a breakpoint or the program will run and terminate (so you will not be able to check the results).

- Display the variables:
  - Become familiar with how to set breakpoints (bp xx), the run command (run or r), continue (cont or c) command, and examine memory commands (x/<n><f><u> &varName).
  - Refer to the debugger information (last page) or the class text (Chapter 6) for additional information.

- Create a Debugger output file
  - Download the assignment #1 debugger input file.
  - In the debugger, at the (gdb) prompt, read the commands (from the file) via:

```
source <file_name>
```

  - Where <fileName> is the name of the assignment #1 debugger input file previously downloaded (***a1in.txt*** by default). If the default file name is used, the command would be:

```
source a1in.txt
```

  - *Note*, the debugger may prompt for Restart or Exit. If everything worked, you may choose Exit to terminate the debugger session.
  - The **`a1in.txt`** debugger input file creates and output file named **`a1out.txt`** where the results are placed.

- Refer to the text, *x86-64 Assembly Language Programming with Ubuntu*, Chapter 6 for detailed information and complete examples for using the DDD debugger.

Submission:

- All source files must assemble and execute on Ubuntu with **`yasm`**.

- Submit source file
  - Submit a copy of the program source file via the on-line submission

- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time.

- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
;  Name: <your name>
;  NSHE ID: <your id>
;  Section: <section>
;  Assignment: <assignment number>
;  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 20%.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Program Header	20%	Must include header block in the required format (see above).
General Comments	20%	Must include an appropriate level of program documentation.
Program Functionality (and on-time)	60%	Program must meet the functional requirements as outlined in the assignment. Must be submitted on time for full score.

Debugger Commands:

Execute the program in the debugger (as describe previously). You should review the DDD/GDB debugger information handout to understand the debugger commands examine memory variables.

You should use the provided "**a1in.txt**" to display the variables with the debugger.

- Each byte, word, double-word sized, and quadword variable is displayed twice (once in decimal and again in hex).
- The floating point values are display twice (once as a real value and again in hex).
- The strings are displayed twice, once showing both the decimal and ASCII values and then just the hex values for the first six characters

A brief summary of the command to examine memory is as follows:

x/<n><f><u> &<variable>	Examine memory location <variable>		
	<n>	number of locations to display, 1 (number one) is the default.	
	<f>	format:	d – decimal
			x – hex
			u – unsigned
			c – character
			s – string
			f – floating point
	<u>	unit size:	b – byte (8-bits)
			h – halfword (16-bits)
			w – word (32-bits)
			g – giant (64-bits)

For example, to display the 8-bit variable **num1**, the 16-bit variable **num3**, the 32-bit variable **num5**, and the 64-bit variable **num7**, the commands would be as follows:

```
x/db &num1
x/dh &num3
x/dw &num5
x/dg &num7
```

For future assignments you will need to select the correct command to display the data based on the defined size and any guidance from the assignment.

More detailed information can be found in Chapter 6 of the class text.