

Computer Organization and Software

COEN 311 (AL-X)

Experiment 1

Andre Hei Wang Law

4017 5600

Mustafa Daraghmeh

Performed on May 27, 2021

Due on June 3, 2021

“I certify that this submission is my original work and meets the Faculty’s Expectations of Originality”, Tuesday, June 1, 2021.

4017 5600 *FA*

## **1) Objectives**

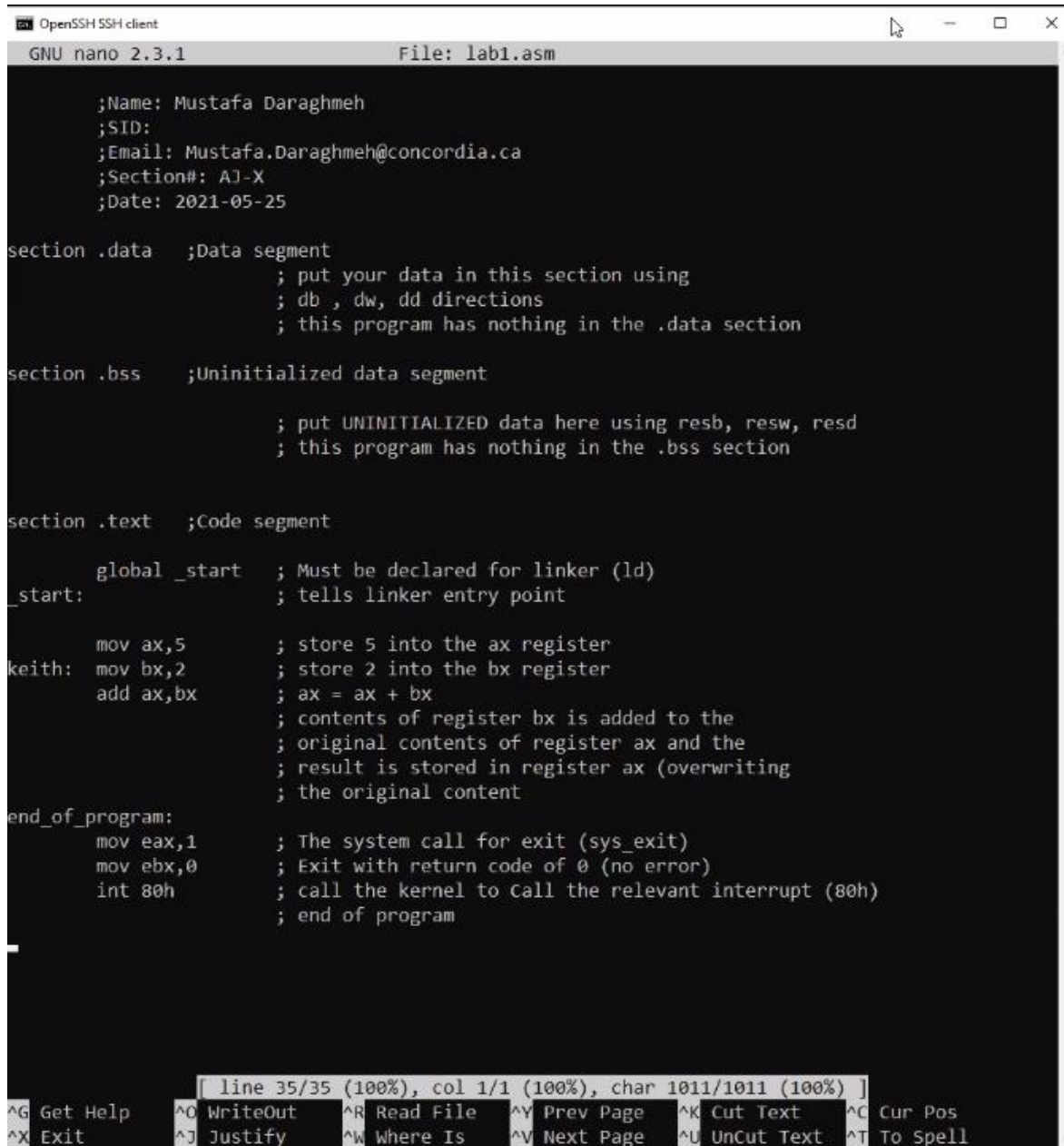
For the first evaluated experiment of the course COEN 311, students will continue to explore basic Linux commands and text editing with nano which were both topics introduced in lab 0. Students are also expected to familiarise themselves with connecting to the ENCS Linux server in order to assemble, load and execute program from Intel x86 assembly language source. Finally, students will be tasked to perform *gdb* command line debugger as a mean to comprehend each step of the written program.

## **2) Theory (+Screenshots)**

In order to successfully perform experiment 1 of the course COEN 311, students will need to have a basic understanding on Linux as well as Intel x86 assembly language. As such, it is important to learn several useful commands as detailed in the lab manual which is explained and summarised in the following manner:

### **Using a Linux Command Line Text Editor in the ENCS Server:**

This section of the lab overlaps with lab 0 as connecting to the ENCS server and using the nano editor was already explained previously. However, to summarise, students can get remote access to the ENCS server by prompting “ssh -Y login.concordia.ca -l user\_name” in any command prompt and then inputting their password. After some file manipulation with “mkdir” (create a new directory) and “cd” (go to a directory) commands, it is then possible to create a “.asm” (source code) with the “nano” command in which the main code will be housed in. The final result yields the following:



```
OpenSSH SSH client
GNU nano 2.3.1 File: lab1.asm

;Name: Mustafa Daraghmeh
;SID:
;Email: Mustafa.Daraghmeh@concordia.ca
;Section#: AJ-X
;Date: 2021-05-25

section .data ;Data segment
    ; put your data in this section using
    ; db , dw, dd directions
    ; this program has nothing in the .data section

section .bss ;Uninitialized data segment
    ; put UNINITIALIZED data here using resb, resw, resd
    ; this program has nothing in the .bss section

section .text ;Code segment

    global _start ; Must be declared for linker (ld)
_start:
    ; tells linker entry point

    mov ax,5 ; store 5 into the ax register
keith: mov bx,2 ; store 2 into the bx register
    add ax,bx ; ax = ax + bx
    ; contents of register bx is added to the
    ; original contents of register ax and the
    ; result is stored in register ax (overwriting
    ; the original content

end_of_program:
    mov eax,1 ; The system call for exit (sys_exit)
    mov ebx,0 ; Exit with return code of 0 (no error)
    int 80h ; call the kernel to Call the relevant interrupt (80h)
    ; end of program

[ line 35/35 (100%), col 1/1 (100%), char 1011/1011 (100%) ]
^G Get Help ^O WriteOut ^R Read File ^V Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^N Next Page ^U UnCut Text ^T To Spell
```

## Nasm Assembler:

Before anything else, it is important to check if the assembler is on the system by prompting “which nasm”. This command will display the path location of the nasm program if it is already installed on your machine. Afterwards, students can assemble their written “.asm” code by inputting “nasm -f elf file\_name.asm -l file\_name.lis”. Unlike previous commands, this one requires dissecting piece by piece in order to fully understand what exactly this command does.

- “nasm” = Instruct a command from the nasm program
- “-f elf” = Specify the type of output file format as a Linux executable and linkable format
- “file\_name.asm” = The written source code .asm file
- “-l” = Create a listing file with a specified name
- “file\_name.lis” = The name of the specific .lis file

The assemble phase is deemed successful when no error message is prompted and writing the command “ls” (list all files and folders in current directory) displays three files (.asm, .lis and .o) such as shown in the following:

```
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > ls -l l*
-rw-rw---- 1 l_heiwan l_heiwan 1013 May 27 18:38 lab1.asm
-rw-rw---- 1 l_heiwan l_heiwan 2373 May 27 18:39 lab1.lis
-rw-rw---- 1 l_heiwan l_heiwan 608 May 27 18:39 lab1.o
```

The \* specifies that the list (“ls”) should only display files that start with “l” in this case. Except for “ls”, there is also “file” command that displays the specific type/format of file of the directory. Thus, prompting “file l\*” displays all format of files that starts with the letter “l”.

```
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > file l*
lab1.asm: ASCII text
lab1.lis: ASCII text
lab1.o:   ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
```

## Executable Program:

The “ld” command is a linkage binder that combines files into one output file which can be executed. In this case, the prompt will be as such, “ld -melf\_i386 -o output\_file input\_file.o”.

- “melf\_i386” = Emulator of choice for this specific environment
- “-o output\_file” = Name of the executable output file (“lab1” in this case)
- “input\_file.o” = Assembled object, input file used to create the executable program

```
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > ld lab1.o -m elf_i386 -o lab1
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > ls -l l*
-rwxrwx--- 1 l_heiwan l_heiwan 552 May 27 18:57 lab1
-rw-rw---- 1 l_heiwan l_heiwan 1013 May 27 18:38 lab1.asm
-rw-rw---- 1 l_heiwan l_heiwan 2373 May 27 18:39 lab1.lis
-rw-rw---- 1 l_heiwan l_heiwan 608 May 27 18:39 lab1.o
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > file l*
lab1:      ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
lab1.asm:  ASCII text
lab1.lis:  ASCII text
lab1.o:    ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
```

## Running the Executable Program:

As the nature of the written code never has any instructions to display any output on the command prompt, students shall use a debugger in order to verify their code. Generally speaking, a debugger is a computer program that runs the code under a controlled environment where every step is tracked in such that errors can be detected as it occurs. However, in this case, the usage of the gdb debugger is to simply be able to visualise each internal single-stepping as it executes. Thus, the following will occur with “gdb lab1” command:

```
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > gdb lab1
GNU gdb (GDB) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab1...(no debugging symbols found)...done.
```

Within the gdb debugger environment, there are many possible commands and instructions that allows the user truly inspect all internal steps of the written program. In such manner, those useful to experiment #1 shall be:

- “quit” = Exits the gdb environment and returns to the ENCS server

```
(gdb) q  
[orwell] [/home/l/l_heiwan/coen311-s/lab1] >
```

- “break” = Create a breakpoint in the program

- “run” = Runs the program

```
(gdb) r  
Starting program: /nfs/home/l/l_heiwan/coen311-s/lab1/lab1
```

- “disassemble” = Converts binary machine code into assembly language code

```
(gdb) disassemble  
Dump of assembler code for function _start:  
=> 0x08048060 <+0>:      mov     $0x5,%ax  
End of assembler dump.
```

- “set disassembly intel” = Use the Intel x86 style syntax for the ‘disassemble’ function

```
(gdb) disassemble  
Dump of assembler code for function _start:  
=> 0x08048060 <+0>:      mov     $0x5,%ax  
End of assembler dump.  
(gdb) set disassembly intel  
(gdb) disassemble  
Dump of assembler code for function _start:  
=> 0x08048060 <+0>:      mov     ax,0x5  
End of assembler dump.
```



- “ni” = Single step into the next part and execute the instruction

```
(gdb) ni
0x08048064 in keith ()
```

- “print” = Displays the content, use “/x \$register\_name” to specify the register and display in /x format (hexadecimal outputs)

```
(gdb) print /x $ax
$1 = 0x0
```

- “info r” = Display content of the CPU registers

```
(gdb) info r
eax                0x7          7
ecx                0x0          0
edx                0x0          0
ebx                0x2          2
esp                0xffffd2b0     0xffffd2b0
ebp                0x0          0x0
esi                0x0          0
edi                0x0          0
eip                0x804806b     0x804806b <end_of_program>
eflags             0x202        [ IF ]
cs                 0x23         35
ss                 0x2b         43
ds                 0x2b         43
es                 0x2b         43
fs                 0x0          0
gs                 0x0          0
```

### 3) Questions

*Explain the differences between an assembly language source code file, the listing file produced by nasm, the object file (.o) and the final executable program.*

**Assembly Language Source Code File (.asm):** `lab1.asm: ASCII text`

In general, codes in assembly language can be saved as a “.asm” file format which can be edited or converted into machine language. In our case, this file refers to the “lab1.asm” file where the written code lies. Then, as shown in the experiment, this assembly language source code file was edited by the nano editor and was later used into assembling the “lab1.lis” file.

**Listing File Produced by Nasm (.lis):** `lab1.lis: ASCII text`

Through the command “nasm -f elf lab1.asm -l lab1.lis”, the source code file in assembly language (lab1.asm) was assembled through the nasm assembler. As a result, a listing file “lab1.lis” was produced which is an ASCII text file containing the machine code on the left side while also keeping the original “lab1.asm” source code on the right. As such, the left side allows the user to see the path (data location) as well as the data value of said source code.

```
[orwell] [/home/l/1_heiwan/coen311-s/lab1] > cat lab1.lis
 1      ; Name: Andre Hei Wang Law
 2      ; STD:
 3      ; Email: heiwangandrelaw128@gmail.com
 4      ; Section: AL-X
 5      ; Date: 2021-05-27
 6
 7      section .data    ; Data segment
 8                      ; put your data in this section using
 9                      ; db, dw, dd directions
10                      ; this program has nothing in the .data section
11
12      section .bss     ; Uninitialized data segment
13
14                      ; put UNINITIALIZED data here using resb, resw, resd
15                      ; this program has nothing in the .bss section
16
17
18      section .text    ; Code segment
19
20      global _start    ; Must be declared for linker (ld)
21      _start:          ; tells linker entry point
22
23      00000000 66880500      mov ax,5      ; store 5 into the ax register
24      00000004 66880200      keith: mov bx,2      ; store 2 into the bx register
25      00000008 6601D8        add ax,bx      ; ax = ax + bx
26                                  ; contents of register bx is added to the
27                                  ; original contents of register ax and the
28                                  ; result is stored in register ax (overwriting
29                                  ; the original content)
30
31      end_of_program:
32      0000000B B801000000      mov eax,1      ; the system call for exit (sys_exit)
33      00000010 B800000000      mov ebx,0      ; exit with return code of 0 (no error)
34      00000015 CD80           int 80h        ; call the kernel to call the relevant interrupt (80h)
                                   ; end of program
[orwell] [/home/l/1_heiwan/coen311-s/lab1] >
```



**Object File (.o):** lab1.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

Again, through the same command “nasm -f elf lab1.asm -l lab1.lis”, an output file “lab1.o” was created. This “.o” file format means it is a Linux executable and linkable format and is later used in creating the executable program “lab1”.

**Final Executable Program:** lab1: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped

In general, the “ld” command creates an executable file that is the combination of numerous object and archive files which have their data relocated and tied up into symbol references. In our case, the “ld lab1.o -m elf\_i386 -o lab1” command uses the object file “lab1.o” and creates an executable program named “lab1”. In such manner, the final executable program “lab1” can be called and ran simply by typing the name of the file (“lab1”) in the command prompt. Knowing this code doesn’t display anything, this executable program can also be ran through a debugger.

```
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > ld lab1.o -m elf_i386 -o lab1
[orwell] [/home/l/l_heiwan/coen311-s/lab1] > ls -l l*
-rwxrwx--- 1 l_heiwan l_heiwan 552 May 27 18:57 lab1
-rw-rw---- 1 l_heiwan l_heiwan 1013 May 27 18:38 lab1.asm
-rw-rw---- 1 l_heiwan l_heiwan 2373 May 27 18:39 lab1.lis
-rw-rw---- 1 l_heiwan l_heiwan 608 May 27 18:39 lab1.o
```

#### **4) Conclusions**

In conclusion, experiment 1 of the course COEN 311 allowed the students to practice and gain experience manipulating and working with assembly language. More specifically, it allowed students to revise their knowledge on connecting to the ENCS Linux server as well as creating “.asm” files using the nano editor which were topics covered in the introductory lab 0. Then, students also worked on assembling, loading as well as executing their written source code while being conscious of file format (.asm, .lis and .o) and directory locations (mkdir, cd, ls \*, etc.). To wrap everything and ensure a concise understanding on the topic, they executed their code through the gdb debugger to inspect every aspect of their code in real time. In the end, lab 1 provided the students with a lot of commands to practice which will also be useful for future labs as these basic topics will be the foundation for future experiments.

## **5) Appendix:**

**->lab1.asm<-**

; Name: Andre Hei Wang Law

; STD:

; Email: heiwangandrelaw128@gmail.com

; Section: AL-X

; Date: 2021-05-27

section .data ; Data segment

; put your data in this section using

; db, dw, dd directions

; this program has nothing in the .data section

section .bss ; Uninitialized data segment

; put UNINITIALIZED data here using resb, resw, resd

; this program has nothing in the .bss section

section .text ; Code segment

global \_start ; Must be declared for linker (ld)

\_start: ; tells linker entry point

```

        mov ax,5      ; store 5 into the ax register

keith:  mov bx,2      ; store 2 into the bx register

        add ax,bx     ; ax = ax + bx

                        ; contents of register bx is added to the
                        ; original contents of register ax and the
                        ; result is stored in register ax (overwriting
                        ; the original content)

end_of_program:

        mov eax,1     ; the system call for exit (sys_exit)

        mov ebx,0     ; exit with return code of 0 (no error)

        int 80h       ; call the kernel to call the relevant interrupt (80h)

                        ; end of program

```

->lab1.lis<-

```
1          ; Name: Andre Hei Wang Law
2          ; STD:
3          ; Email: heiwangandrelaw128@gmail.com
4          ; Section: AL-X
5          ; Date: 2021-05-27
6
7          section .data      ; Data segment
8
9          ; put your data in this section using
10         ; db, dw, dd directions
11
12         ; this program has nothing in the .data section
13
14         section .bss       ; Uninitialized data segment
15
16         ; put UNINITIALIZED data here using resb, resw,
17         resd
18
19         ; this program has nothing in the .bss section
20
21         section .text      ; Code segment
22
23         global _start      ; Must be declared for linker (ld)
24
25         _start:            ; tells linker entry point
```

```

22
23 00000000 66B80500      mov ax,5      ; store 5 into the ax register
24 00000004 66BB0200      keith: mov bx,2      ; store 2 into the bx register
25 00000008 6601D8        add ax,bx      ; ax = ax + bx
26                          ; contents of register bx is added to the
27                          ; original contents of register ax and the
28                          ; result is stored in register ax (overwriting
29                          ; the original content)
30                          end_of_program:
31 0000000B B801000000      mov eax,1      ; the system call for exit (sys_exit)
32 00000010 BB00000000      mov ebx,0      ; exit with return code of 0 (no error)
33 00000015 CD80           int 80h          ; call the kernel to call the relevant
interrupt (80h)
34                          ; end of program

```