Computer Organization and Software

COEN 311 (AL-X)

Experiment 3

Andre Hei Wang Law

4017 5600

Mustafa Daraghmeh

Performed on June 10, 2021

Due on June 17, 2021

“I certify that this submission is my original work and meets the Faculty’s Expectations of Originality”, Thursday, June 17, 2021.

40175600

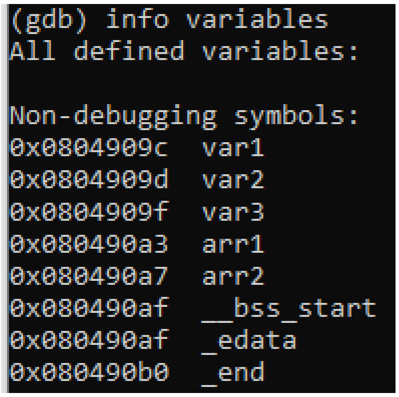
1) Objectives

For the third experiment of the course COEN 311, students will acquire strong understanding on addressing modes as well as basic arithmetic operations in order to write an assembly language program of a two-dimensional array of integers stored in memory. Furthermore, students will be expected to gain an appreciation of high-level programming language compiler for their complex tasks performed behind the scenes.

2) Theory

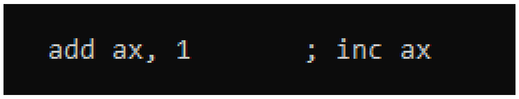
In terms of familiarity, students should of have already grasped all basic commands used throughout previous experiments (nano, nasm, ld, etc.). For such reason, this section will focus on new commands and notions in which will eventually also become part of students’ useful toolbox of commands.

Variables (“info variables”): Used in gdb to display the actually variables present, whether global or static.



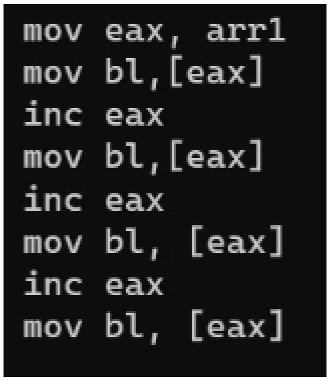
```
(gdb) info variables
All defined variables:

Non-debugging symbols:
0x0804909c  var1
0x0804909d  var2
0x0804909f  var3
0x080490a3  arr1
0x080490a7  arr2
0x080490af  __bss_start
0x080490af  _edata
0x080490b0  _end
```

Increment ("add/inc"): Add 1 to either a register or a memory location. While "add" uses 4 bytes of memory, "inc" only requires 2 bytes.

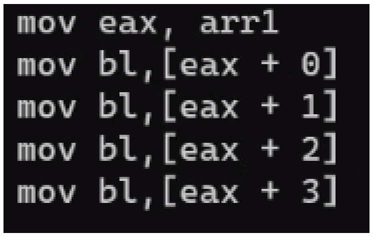


```
add ax, 1       ; inc ax
```

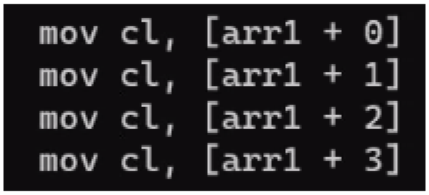
Addressing Modes: The four modes are Immediate, Register, Absolute and Register Indirect. These four modes dictate the method in which the data will be accessed. While each has their own use, many codes aren't restricted to only one mode. For example, the following example shows the displacement ("mov") segment of a code which all ultimately behaves the same.



```
mov eax, arr1
mov bl,[eax]
inc eax
mov bl,[eax]
inc eax
mov bl, [eax]
inc eax
mov bl, [eax]
```



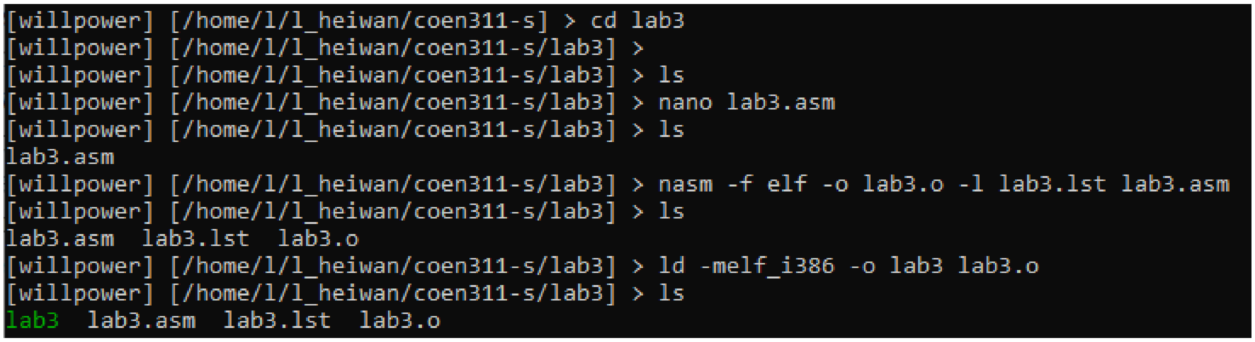
```
mov eax, arr1
mov bl,[eax + 0]
mov bl,[eax + 1]
mov bl,[eax + 2]
mov bl,[eax + 3]
```



```
mov cl, [arr1 + 0]
mov cl, [arr1 + 1]
mov cl, [arr1 + 2]
mov cl, [arr1 + 3]
```

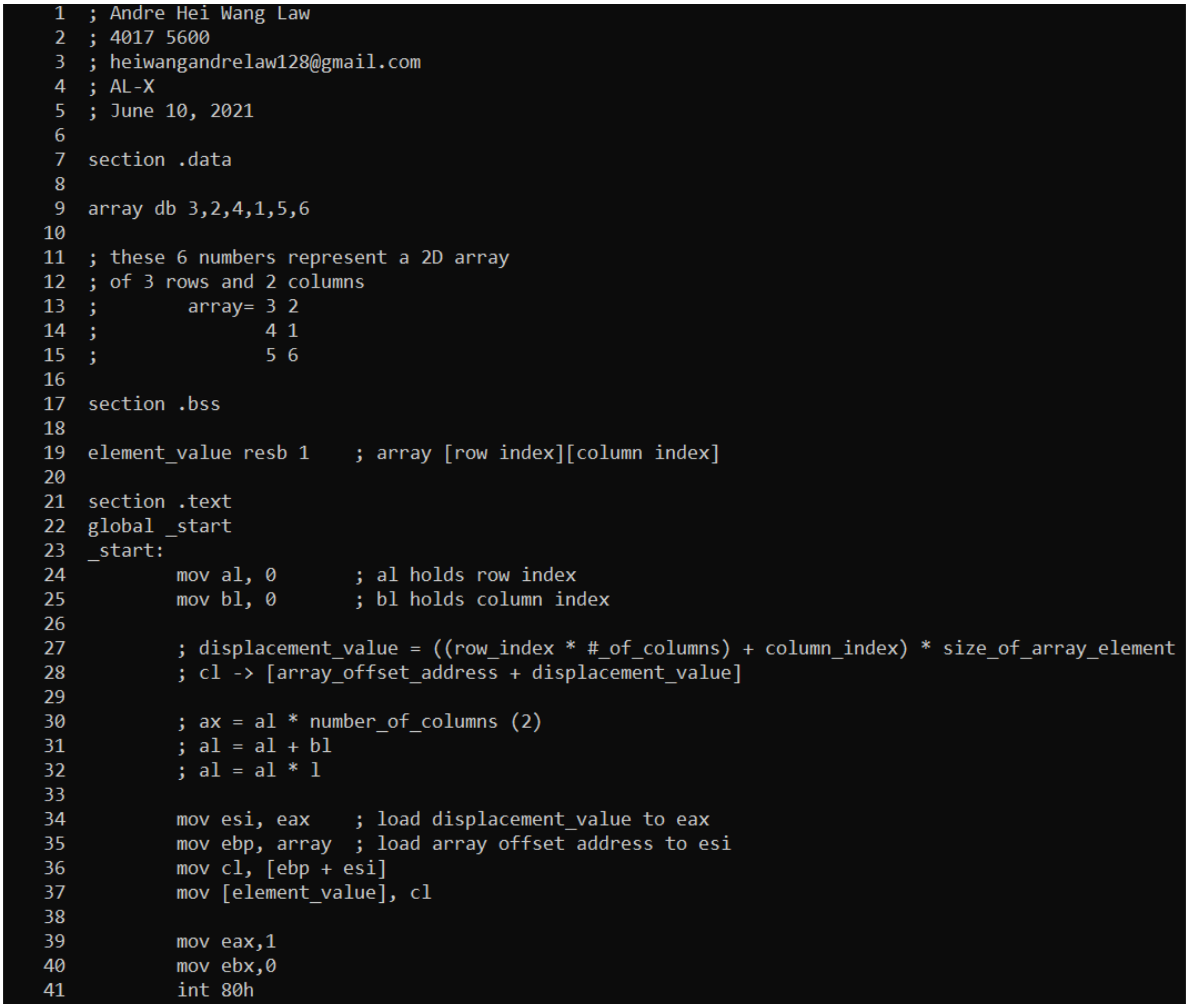
3) Procedure (+Screenshots)

Since this is already the third experiment of the course COEN 311, students should fully comprehend the commands (nano, nasm, ld, etc.) and the steps required to create an executable program such is shown in the following:



```
[willpower] [/home/l/l_heiwan/coen311-s] > cd lab3
[willpower] [/home/l/l_heiwan/coen311-s/lab3] >
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > ls
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > nano lab3.asm
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > ls
lab3.asm
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > nasm -f elf -o lab3.o -l lab3.lst lab3.asm
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > ls
lab3.asm  lab3.lst  lab3.o
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > ld -melf_i386 -o lab3 lab3.o
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > ls
lab3  lab3.asm  lab3.lst  lab3.o
```

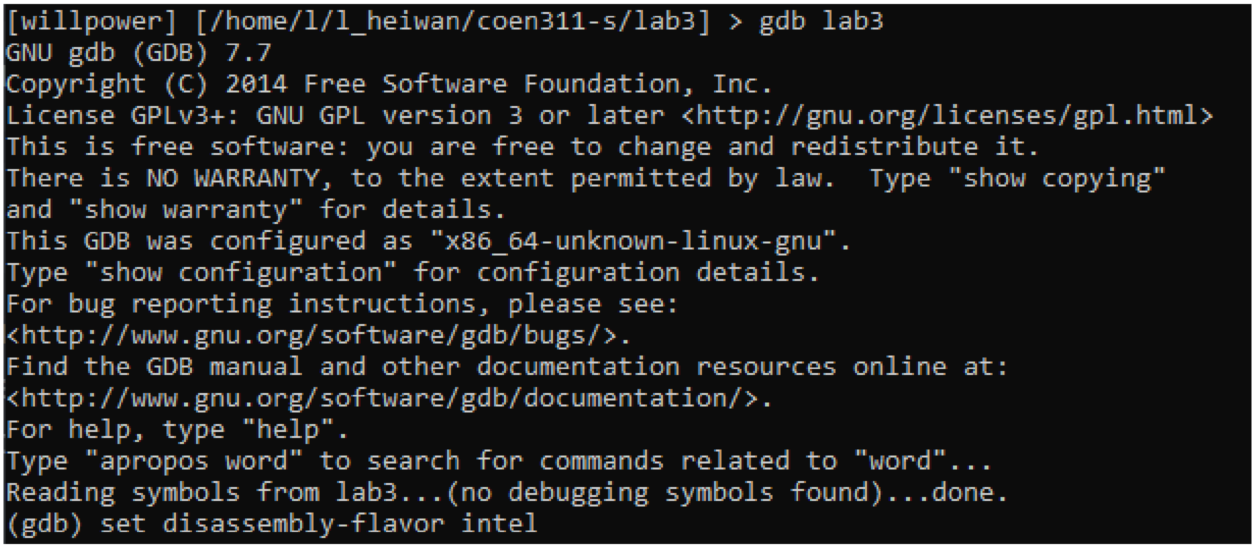
Through the guided solutions provided by the teacher assistant as well as the lab manual, the created source code file “lab3.asm” is as such:



```
; Andre Hei Wang Law
; 4017 5600
; heiwangandrelaw128@gmail.com
; AL-X
; June 10, 2021

section .data

array db 3,2,4,1,5,6

; these 6 numbers represent a 2D array
; of 3 rows and 2 columns
;        array= 3 2
;               4 1
;               5 6

section .bss

element_value resb 1    ; array [row index][column index]

section .text
global _start
_start:
        mov al, 0       ; al holds row index
        mov bl, 0       ; bl holds column index

        ; displacement_value = ((row_index * #_of_columns) + column_index) * size_of_array_element
        ; cl -> [array_offset_address + displacement_value]

        ; ax = al * number_of_columns (2)
        ; al = al + bl
        ; al = al * 1

        mov esi, eax    ; load displacement_value to eax
        mov ebp, array  ; load array offset address to esi
        mov cl, [ebp + esi]
        mov [element_value], cl

        mov eax,1
        mov ebx,0
        int 80h
```

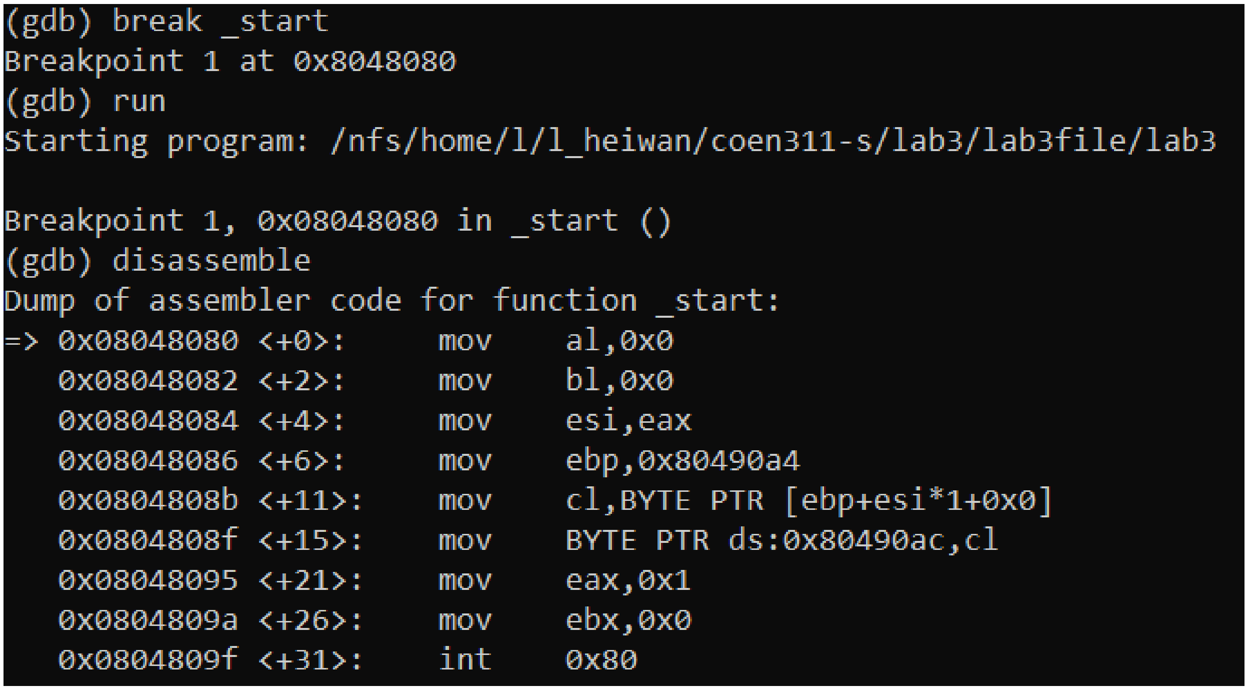
In addition to basic Linux commands, students are also expected to understand concepts of labels and single stepping using gdb debugger in order to run and visualise the backend process of their program such as the following:

(running the executable program through gdb and setting the style syntax to Intel x86)



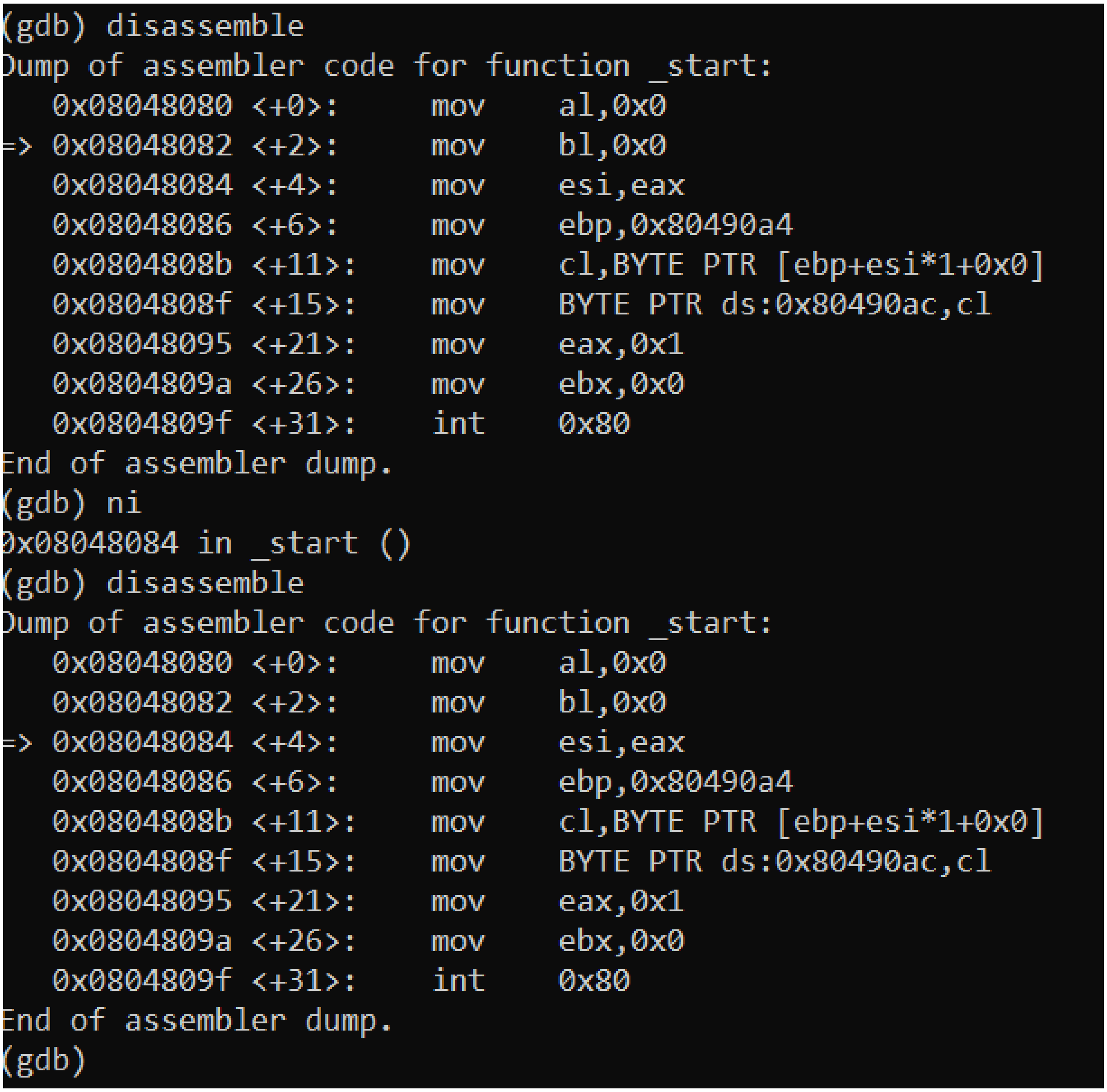
```
[willpower] [/home/l/l_heiwan/coen311-s/lab3] > gdb lab3
GNU gdb (GDB) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the 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 lab3...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
```

(going to the labelled breakpoint "_start", running the program and displaying the details)



```
(gdb) break _start
Breakpoint 1 at 0x8048080
(gdb) run
Starting program: /nfs/home/l/l_heiwan/coen311-s/lab3/lab3file/lab3

Breakpoint 1, 0x08048080 in _start ()
(gdb) disassemble
Dump of assembler code for function _start:
=> 0x08048080 <+0>:     mov    al,0x0
   0x08048082 <+2>:     mov    bl,0x0
   0x08048084 <+4>:     mov    esi,eax
   0x08048086 <+6>:     mov    ebp,0x80490a4
   0x0804808b <+11>:    mov    cl,BYTE PTR [ebp+esi*1+0x0]
   0x0804808f <+15>:    mov    BYTE PTR ds:0x80490ac,cl
   0x08048095 <+21>:    mov    eax,0x1
   0x0804809a <+26>:    mov    ebx,0x0
   0x0804809f <+31>:    int    0x80
```

The debugger gdb allows single stepping through the code such that each element can be inspected individually. The commands for such are a combination of "ni" and "disassemble" in which one goes over to the next line of code and one displays the content, respectively. The following screenshot is a sample of perfuming single stepping of the executable “lab3” program:



```
(gdb) disassemble
Dump of assembler code for function _start:
   0x08048080 <+0>:     mov    al,0x0
=> 0x08048082 <+2>:     mov    bl,0x0
   0x08048084 <+4>:     mov    esi,eax
   0x08048086 <+6>:     mov    ebp,0x80490a4
   0x0804808b <+11>:    mov    cl,BYTE PTR [ebp+esi*1+0x0]
   0x0804808f <+15>:    mov    BYTE PTR ds:0x80490ac,cl
   0x08048095 <+21>:    mov    eax,0x1
   0x0804809a <+26>:    mov    ebx,0x0
   0x0804809f <+31>:    int    0x80
End of assembler dump.
(gdb) ni
0x08048084 in _start ()
(gdb) disassemble
Dump of assembler code for function _start:
   0x08048080 <+0>:     mov    al,0x0
   0x08048082 <+2>:     mov    bl,0x0
=> 0x08048084 <+4>:     mov    esi,eax
   0x08048086 <+6>:     mov    ebp,0x80490a4
   0x0804808b <+11>:    mov    cl,BYTE PTR [ebp+esi*1+0x0]
   0x0804808f <+15>:    mov    BYTE PTR ds:0x80490ac,cl
   0x08048095 <+21>:    mov    eax,0x1
   0x0804809a <+26>:    mov    ebx,0x0
   0x0804809f <+31>:    int    0x80
End of assembler dump.
(gdb)
```

Having arrived to this point, students are then required to return to their source code file to edit, re-assemble and re-run the executable program through gdb with their new and edited values of row and column indices. This repetition will ensure that the program functions properly and the desired array elements in the register shows up accordingly.

4) Conclusions

In conclusion, experiment 3 of the course COEN 311 provided the students a place to put into practice their knowledge of addressing modes and arithmetic operations by writing an assembly language program of a two-dimensional array stored in memory. More specifically, students started with manipulating and accessing data through different addressing modes in order to grasp their functionalities and purpose. Afterwards, they experimented on arithmetic instructions “add” and “inc” for which their differences in memory size were to be noticed. In addition to these notions, students got to understand more in detail the backend process of many tasks throughout the experiment such as knowing when it was “nano”, “nasm” or “ld” who was manipulating the files in a specific manner. In the end, by practicing the entire process of writing, editing and running multiple programs of a two-dimensional array, students have strengthened their understanding on Intel x86 assembly language as well as gaining new knowledge such were the different addressing modes and the arithmetic operations.

5) Appendix:

-----lab3.asm---

```
; Andre Hei Wang Law
; 4017 5600
; heiwangandrelaw128@gmail.com
; AL-X
; June 10, 2021

section .data

array db 3,2,4,1,5,6
```

```
; these 6 numbers represent a 2D array
; of 3 rows and 2 columns
;       array= 3 2
;              4 1
;              5 6

section .bss

element_value resb 1  ; array [row index][column index]

section .text
global _start
_start:
        mov al, 0       ; al holds row index
        mov bl, 0       ; bl holds column index

        ; displacement_value = ((row_index * #_of_columns) + column_index) * size_of_array_element
        ; cl -> [array_offset_address + displacement_value]

        ; ax = al * number_of_columns (2)
        ; al = al + bl
        ; al = al * 1
```

```
    mov esi, eax    ; load displacement_value to eax

    mov ebp, array            ; load array offset address to esi

    mov cl, [ebp + esi]

    mov [element_value], cl


    mov eax,1

    mov ebx,0

    int 80h
```

---lab3.lst---

```
 1                              ; Andre Hei Wang Law

 2                              ; 4017 5600

 3                              ; heiwangandrelaw128@gmail.com

 4                              ; AL-X

 5                              ; June 10, 2021

 6

 7                              section .data

 8

 9 00000000 030204010506            array db 3,2,4,1,5,6

10

11                               ; these 6 numbers represent a 2D array

12                               ; of 3 rows and 2 columns
```

```
13                      ;     array= 3 2
14                      ;            4 1
15                      ;            5 6
16
17                      section .bss
18
19 00000000 ??                element_value resb 1      ; array [row index][column index]
20
21                      section .text
22                      global _start
23                      _start:
24 00000000 B000                    mov al, 0      ; al holds row index
25 00000002 B300                    mov bl, 0      ; bl holds column index
26
27                          ; displacement_value = ((row_index * #_of_columns) + column_index) * size_of_array_element
28                          ; cl -> [array_offset_address + displacement_value]
29
30                          ; ax = al * number_of_columns (2)
31                          ; al = al + bl
32                          ; al = al * 1
33
34 00000004 89C6                    mov esi, eax   ; load displacement_value to eax
```

```
35 00000006 BD[00000000]          mov ebp, array       ; load array offset address to esi
36 0000000B 8A4C3500              mov cl, [ebp + esi]
37 0000000F 880D[00000000]        mov [element_value], cl
38
39 00000015 B801000000            mov eax,1
40 0000001A BB00000000            mov ebx,0
41 0000001F CD80                  int 80h
```