Computer Organization and Software

COEN 311 (AL-X)

Experiment 3

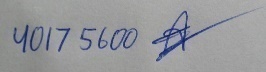
Andre Hei Wang Law

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.

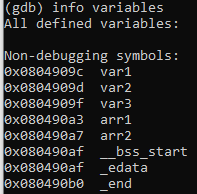
**1) Objectives**

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**

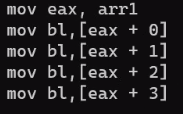
In terms of familiarity, students should of have already grasped all basic commands used throughout previous experiments (nano, nasm, ld, etc.). For such reason, this section will focus on new commands and notions in which will eventually also become part of students’ useful toolbox of commands.

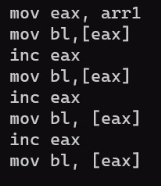
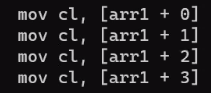
**Variables (“info variables”):** Used in gdb to display the actually variables present, whether global or static.



**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.

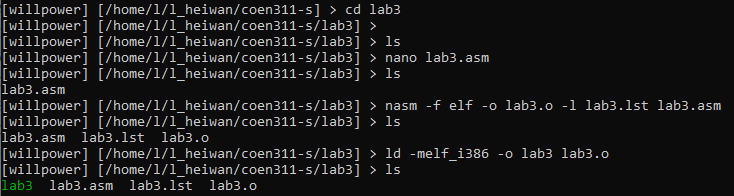


**Addressing Modes:** The four modes are Immediate, Register, Absolute and Register Indirect. These four modes dictate the method in which the data will be accessed. While each has their own use, many codes aren’t restricted to only one mode. For example, the following example shows the displacement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

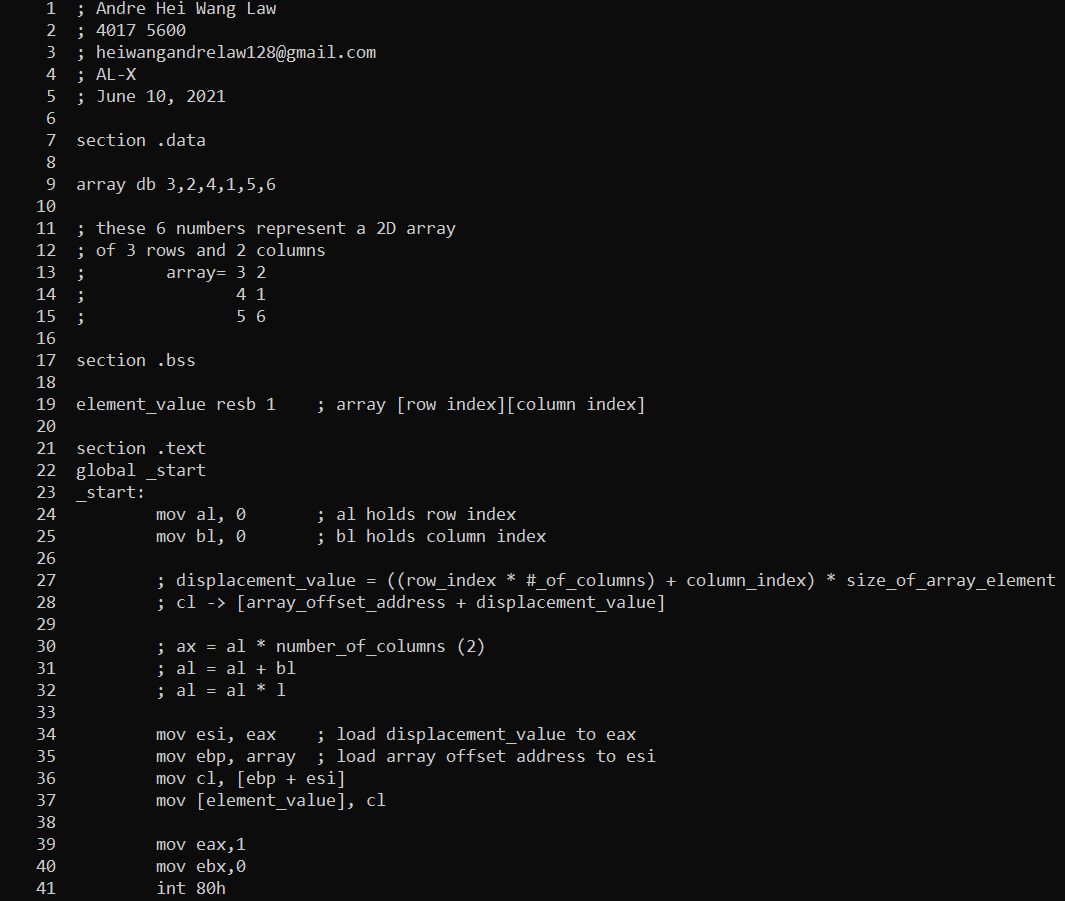


**3) Procedure (+Screenshots)**

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:

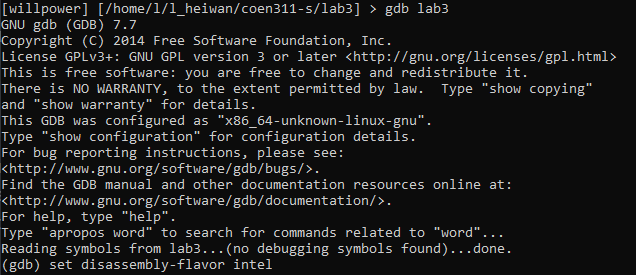


Through the guided solutions provided by the teacher assistant as well as the lab manual, the created source code file “lab3.asm” is as such:

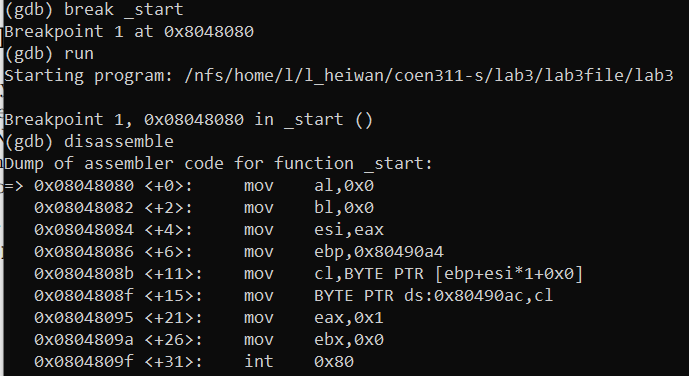


In addition to basic Linux commands, students are also expected to understand concepts of labels and single stepping using gdb debugger in order to run and visualise the backend process of their program such as the following:

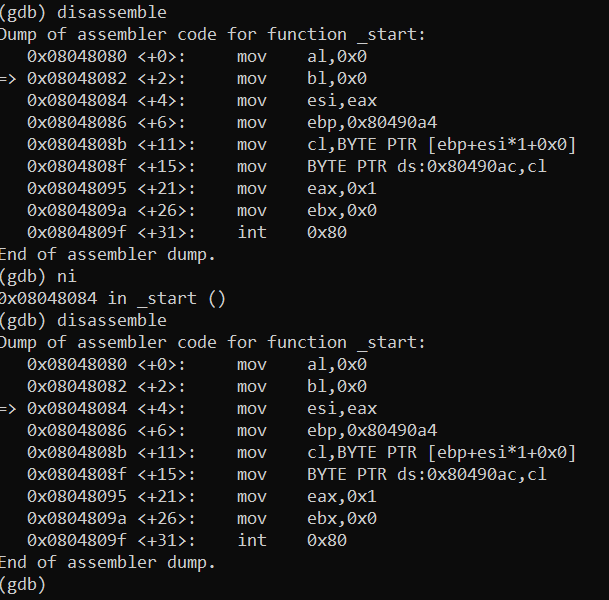
(running the executable program through gdb and setting the style syntax to Intel x86)



(going to the labelled breakpoint “\_start”, running the program and displaying the details)



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:



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 \* l

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 \* l

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