Computer Organization and Software

COEN 311 (AL-X)

Experiment 2

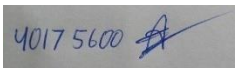
Andre Hei Wang Law

4017 5600

Mustafa Daraghmeh

Performed on June 3, 2021

Due on June 10, 2021

"I certify that this submission is my original work and meets the Faculty's Expectations of Originality", Thursday, June 10, 2021. 

1) Objectives

For the second experiment of the course COEN 311, students will continue to practice and gain experience on Linux utilities, nasm and gdb debugger. In addition, students will explore different addressing modes that Intel x86 assembly has to offer. As such, these goals will be met by assembling a program with nasm, creating an executable with ld and finally single-stepping through the program with the gdb command while also verifying the contents of the registers of different memory addresses throughout each instruction.

2) Theory

Experiment 2 of the course COEN 311 introduces many new topics and commands for Intel x86 assembly, many in which will be crucial for a better understanding of this lab. As such, the following definitions will briefly discuss the key concepts and commands that will be present for this experiment.

Data Size: Allocated storage space (db = defined byte in 1 byte, dw = defined word in 2 bytes and dd = defined double word in 4 bytes).

Endian: Order of sequence of stored data. (Little-Endian = low but, then high bit; Big-Endian = high bit, then low bit).

Immediate Mode: A data stored within the instruction which has been fetched from main memory and stored inside the processor, thus allowing the execution of said instruction without needing the CPU to access main memory.

Label: Optional marker used to refer to a particular instruction.

Direct Addressing Mode: A fundamental addressing mode used when the data is stored in main memory location.

3) Exercises (+Screenshots)

Having learned to connect to the ENCS Linux server and to handle, create and edit .asm files with the nano command from previous experiments, the focus of this report will lean towards on the topics of registers and memory contents.

Step 1: Assemble and Generate

After having written the necessary .asm source code file from the lab manual experiment 2 section using the nano command, it is possible to generate the .o and the .lst files as such:

```
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > ls
lab2.asm
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > nasm -f elf -o lab2.o -l lab2.lst lab2.asm
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > ls
lab2.asm  lab2.lst  lab2.o
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] >
```

Step 2: Create an Executable

With the object file (lab2.o), it is then possible to create an executable program with the ld command such were done in the following:

```
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > ld -melf_i386 -o lab2 lab2.o
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > ls
lab2  lab2.asm  lab2.lst  lab2.o
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] >
```

Step 3: Single-Step with gdb

For this experiment, the gdb command is used to observe each step of the created executable program “lab2”. This debugger provides useful information of the background process during the execution of said program such as the memory content and register values of a particular section. However, before that, it is important to run the gdb in Intel x86 style syntax rather than the non-intel default which is accomplished by the “set disassembly-flavor intel” command. As such, the following screenshot shows the program running in this syntax.

```
[punctuality] [/home/l/l_heiwan/coen311-s/lab2] > gdb lab2
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 lab2...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb)
```

With everything prepared and set up properly, it is now time to learn about single-stepping through the program and analyzing the differences in register values and memory content in each segment of the program. This can be accomplished by starting at a breakpoint (“break _start”), running the program (“run”) and disassembling the program (“disassemble”).

```
(gdb) break _start
Breakpoint 1 at 0x8048080
(gdb) run
Starting program: /nfs/home/l/l_heiwan/coen311-s/lab2/lab2

Breakpoint 1, 0x08048080 in _start ()
(gdb) disassemble
Dump of assembler code for function _start:
=> 0x08048080 <+0>:     mov    ax,ds:0x804909c
End of assembler dump.
(gdb)
```

In the previous screenshot, the disassembled segment represents “mov ax, [mick]” of the original source code .asm file as this is the first instruction starting from the “_start” label. An interesting observation to be made here is that the offset values are given in memory address such that “mick” is represented by 0x804909c. Following through, the instruction to proceed into the next segment is the “ni” command which allows the student to single-step and disassemble the next segment as such:

```
(gdb) ni
0x08048086 in ron ()
(gdb) disassemble
Dump of assembler code for function ron:
=> 0x08048086 <+0>:     mov    bx,WORD PTR ds:0x804909e
   0x0804808d <+7>:     add    ax,bx
   0x08048090 <+10>:    mov    eax,0x1
   0x08048095 <+15>:    mov    ebx,0x0
   0x0804809a <+20>:    int    0x80
End of assembler dump.
(gdb)
```

Based on the two previous screenshots, notice that we know for certain that we have successfully continue into the next part of the program as the target moved from "_start" to "ron". This is further confirmed as the disassembled function is different from earlier. Another observation to be made is that the disassembled section contains more content than previously which makes sense as there were more instructions in the "ron" label than the "_start" label (referring to the created .asm file). However, not so different from previously, the address associated with "keith" can be found in a similar manner that was done with "mick". By observing the last screenshot, it can be concluded that keith is associated with the address 0x803909e. Having learned the manner in which "mick" and "keith" were stored in their respective addresses, it is also possible to fetch said stored values by examining the content of the memory such with the x command. An example of this command is "x/1xb &mick" where "x" is calling the command itself, "1" is the size, "x" is the format, "b" is the data size and "&mick" is the address.

Set of examples with "mick" and "keith" with different size (number of items shown):

```
(gdb) x/1xb &mick
0x804909c:      0x02
(gdb) x/2xb &mick
0x804909c:      0x02    0x00
(gdb) x/4xb &mick
0x804909c:      0x02    0x00    0x03    0x00
(gdb) x/4xb 0x804909c
0x804909c:      0x02    0x00    0x03    0x00
(gdb)
```

```
(gdb) x/1xb &keith
0x804909e:      0x03
(gdb) x/2xb &keith
0x804909e:      0x03    0x00
(gdb) x/4xb &keith
0x804909e:      0x03    0x00    0x00    0x00
(gdb) x/4xb 0x804909e
0x804909e:      0x03    0x00    0x00    0x00
(gdb)
```

Set of examples with “mick” and “keith” with different format (x for hexadecimal, d for decimal and t for binary):

```
(gdb) x/1xb &mick
0x804909c:      0x02
(gdb) x/1db &mick
0x804909c:      2
(gdb) x/1tb &mick
0x804909c:      00000010
(gdb)
```

```
(gdb) x/1xb &keith
0x804909e:      0x03
(gdb) x/1db &keith
0x804909e:      3
(gdb) x/1tb &keith
0x804909e:      00000011
(gdb)
```

Set of examples with “mick” and “keith” with different data size (b for 1 byte, h for 2 bytes, w for 4 bytes):

```
(gdb) x/1xb &mick
0x804909c:      0x02
(gdb) x/1xh &mick
0x804909c:      0x0002
(gdb) x/1xw &mick
0x804909c:      0x00030002
```

```
(gdb) x/1xb &keith
0x804909e:      0x03
(gdb) x/1xh &keith
0x804909e:      0x0003
(gdb) x/1xw &keith
0x804909e:      0x00000003
```

4) Conclusions

In conclusion, experiment 2 of the course COEN 311 allowed the students to work in assembly language with more depth and complexity, as well as allowing them to practice and familiarize themselves with topics already covered previously. Namely, students were able to review their knowledge on connecting to the ENCS Linux server, on creating .asm files with nano, on creating object files out of the .asm file, etc. In addition to these reviews, students were able to thoroughly practice with the gdb debugger by disassembling ("disassemble" command) the program and analyzing the register values and memory addresses for each single-step of the program. This made it clear as to where the data is store (address), what value is stored (register values) and in what manner it is stored (format). This in-depth dissection of the program in combination with the many different way to display the contents of memory using the x command helped the students meet the objective of experiment 2.

5) Appendix:

--lab2.asm--

```
; Andre Hei Wang Law

; June 3, 2021

; sample program to add two numbers which

; are stored somewhere in memory


section .data


; put your data in this section

; db, dw, dd directions
```

```
mick dw 2 ; define one word (2 bytes) of data
keith dw 3 ; define another word of data with value 3

section .bss

; put UNINITIALIZED data here using
; this program has nothing in the .bss section

section .text
        global _start

_start:
        mov ax,[mick] ; store contents of memory word at
                      ; location mick into the ax register
ron:    mov bx,[keith];store contents of memory word at
                      ; location keith 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 (overweriting
                      ; the original content)
```

```
    mov eax,1       ; the system call for exit (sys_exit)
    mov ebx,0       ; exit with return code of 0 (no error)
    int 80h
```

---lab2.lst---

```
 1                          ; Andre Hei Wang Law
 2                          ; June 3, 2021
 3                          ; sample program to add two numbers which
 4                          ; are stored somewhere in memory
 5
 6                          section .data
 7
 8                          ; put your data in this section
 9                          ; db, dw, dd directions
10
11 00000000 0200                   mick dw 2 ; define one word (2 bytes) of data
12 00000002 0300                   keith dw 3 ; define another word of data with value 3
13
14                          section .bss
15
16                          ; put UNINITIALIZED data here using
17                          ; this program has nothing in the .bss section
18
```

```
19                          section .text
20                              global _start
21
22                          _start:
23 00000000 66A1[00000000]              mov ax,[mick]; store contents of memory word at
24                                      ; location mick into the ax register
25 00000006 668B1D[02000000]    ron:    mov bx,[keith];store contents of memory word at
26                                      ; location keith into the bx register
27 0000000D 6601D8              add ax,bx       ; ax = ax + bx
28                                      ; contents of register bx is added to the
29                                      ; original contents of register ax and the
30                                      ; result is stored in register ax (overweriting
31                                      ; the original content)
32
33
34 00000010 B801000000                  mov eax,1       ; the system call for exit (sys_exit)
35 00000015 BB00000000                  mov ebx,0       ; exit with return code of 0 (no error)
36 0000001A CD80                int 80h
```