COMP1562 – Operating System

Laboratory 3

Processing Programs

Group ID: 21

Group Task: Task 2

Usman Basharat

Mohamed Aden

Yunus Hassan

Derek U. Mafohla

Table of Contents

Task 1.1

During this task, we had to follow each step to print out the code that was given to give us, “Hello World!” The first step we had to do is start up putty and login through the server stulinux.cms.gre.ac.uk. We could use the option of nano, pico or vim, so I decided to use nano for the rest of the tasks. Referring to Figure 1, this is where the code put in to execute by following the next that were steps given. Referring to Figure 2, this shows the results of when exercise one was executed. Referring to Figure 3 and 4, they both show the relevant screenshots for this exercise.

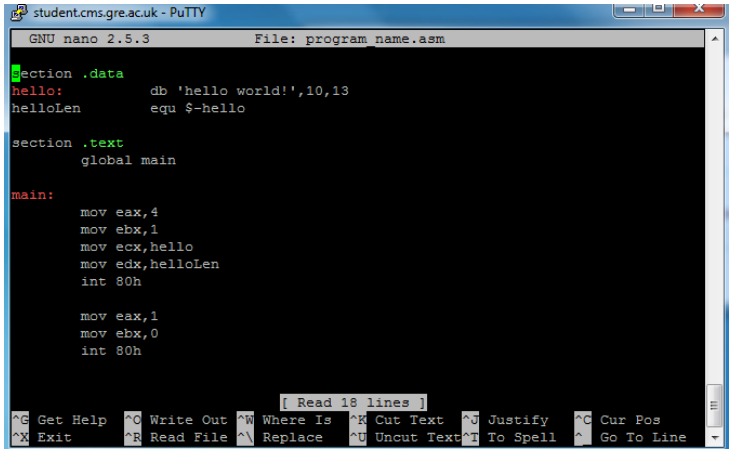

Figure 1 shows the place of the code where it was written

Figure 2 shows the code being execute and the result of it

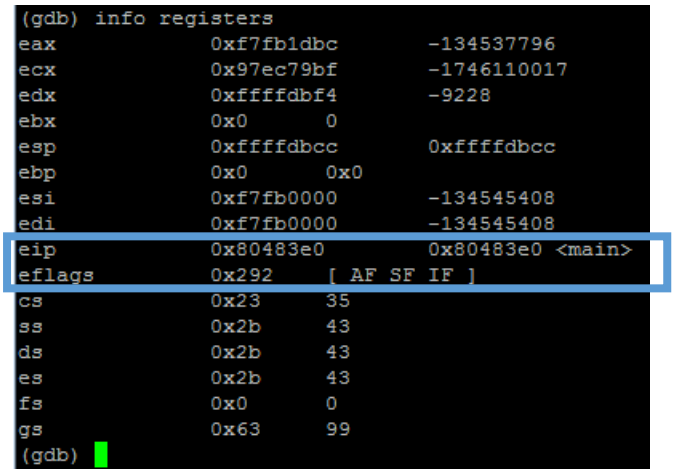

Figure 3 shows the EIP and EFLAGS registers

These two screenshots show two different outcomes of the code that is in. Figure 73 shows all the registers being displayed for it and what is stored in them for this code.

Figure 4, which is on the next page, shows the nexti instruction being implemented. As you can see in the code, eax register holds a value of four. After the nexti instruction is implemented, it shows the value is in place in hexadecimal.

```
(gdb) print/x $esp
$1 = 0xffffdbcc
(gdb)
```

```
(gdb) print/x $eax
$2 = 0xf7fb1dbc
(gdb)
```

```
(gdb) nexti
0x080483e5 in main ()
```

```
(gdb) print/x $esp
$3 = 0xffffdbcc
(gdb)
```

```
(gdb) print/x $eax
$4 = 0x4
(gdb)
```

Figure 4 shows the registers before and after the NEXTI instruction

Task 1.2

This exercise was the same as exercise one. However, the difference was that two numbers was being added together instead of being printed out hello world. Therefore, the same steps were followed and the screenshots below was the proof of the outcome of the following code that was given. Figure 6 shows us the code being executed.

```
section .data
NUMBER1:                dw 2
NUMBER2:                dw 4

section .text
        global main

main:
        mov ax,[NUMBER1]
        add ax,[NUMBER2]
        mov [NUMBER2],ax
        mov eax,1
        mov ebx,0
        int 80h


                        [ Read 14 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Figure 5 shows exercise two code being put in.

```
student:~> chmod +x do_asm
student:~> ./do_asm program_two
student:~> ./program_two
student:~>
```

Figure 6 shows the result of exercise two

```
(gdb) info registers
eax            0xf7fb1dbc       -134537796
ecx            0x2a18fd48       706280776
edx            0xffffdbf4       -9228
ebx            0x0      0
esp            0xffffdbcc       0xffffdbcc
ebp            0x0      0x0
esi            0xf7fb0000       -134545408
edi            0xf7fb0000       -134545408
eip            0x80483e0        0x80483e0 <main>
eflags         0x292    [ AF SF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
(gdb)
```

Figure 7 shows the EIP and EFLAGS registers for exercise two

These two screenshots show two different outcomes of the code that is in. Figure 7 shows all the registers being displayed for it and what is stored in them.

Figure 8 shows what the specific register is stored before and after the NEXTI instruction is being executed. This means that after it being executed it should hold a different value. I assumed that $eax should of hold the value of 0x6 and I entered the same code displayed in the lab document. However, the outcome was completely different.

```
(gdb) print/x $esp
$1 = 0xffffdbcc
(gdb) print/x $eax
$2 = 0xf7fb1dbc
(gdb) nexti
0x080483e6 in main ()
(gdb) print/x $esp
$3 = 0xffffdbcc
(gdb) print/x $eax
$4 = 0xf7fb0002
(gdb) next
Single stepping until exit from function main,
which has no line number information.
[Inferior 1 (process 29267) exited normally]
(gdb) info stack
No stack.
```

Figure 8 shows the NEXTI registers before and after it is executed for exercise two.

Task 1.3

For this task, we were asked to assemble the code and debug it to show the correct answer. The code asked us to divide number 2 by number 1, which equals to 2. For this, to be able to be correct, the code being put in needs to be correct in order to do this. Therefore, Figure 9 shows that the code being used to divide the two numbers together. Figure 10 shows us that once we debugged the steps above, it showed us the correct result.

```
section .data
NUMBER1:                dw 2
NUMBER2:                dw 4
section .text
        global main
main:
        mov ax,[NUMBER1]
        mov bl,[NUMBER2]
        div bl
        mov [NUMBER2],ax

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

Figure 9 shows the assembled code used to divide the two numbers together

```
$3 = 0x2
(gdb)
```

Figure 10 shows us the result of it

Task 2.1

This one was similar to exercise three; however, it was more complex. This asked us to complete an equation of numbers being added in. We were asked to solve the equation below by representing the following numbers as a = 3, b = 7 and h = 4. The answer is 20, which was our aim.

$$\boldsymbol{result} = \frac{\boldsymbol{a} + \boldsymbol{b}}{\boldsymbol{2}} \times \boldsymbol{h}$$

```
section .data
a:              dw 3
b:              dw 7
h:              dw 4
result:         dw 1

section .text
        global main

main:
        mov ax,[a]
        add ax,[b]
        mov[b],ax

        mov ax,[b]
        mov bl,2
        div bl
        mov [result],ax

        mov ax,[h]
        mov bx,[result]
        mul bx
        mov [result],ax
        mov word[result],ax
```

Figure 11 shows us the assembled code being put in. I felt that the previous three exercises helped for this exercise being complete. The adding part above was done by using the exercise one.

The second part and the final part of the code helped by the example being given in the document. I implemented this in our example and I checked this code and it worked.

Figure 11 shows the assembled code of the equation above

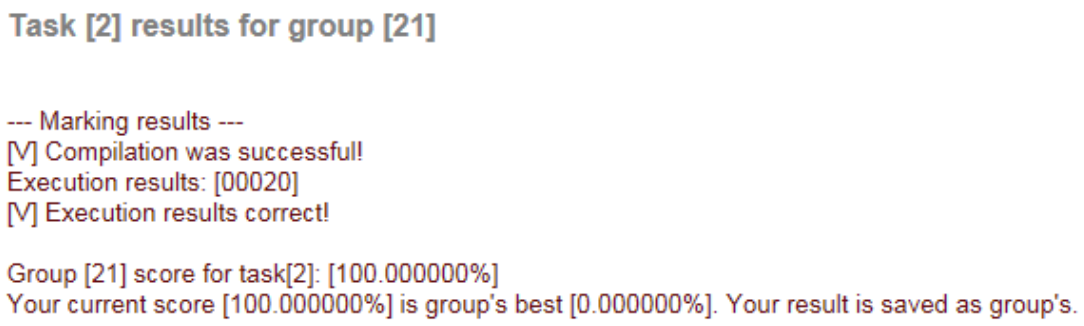

Figure 12 shows us that the score given for this task.

Reflection

During these tasks, I found that it was not as simple as it seemed for the above tasks. I found that each tasks were hugely important as the others. For example, when I did Task 2.1, I went back to previous exercises that I did before to help me add a+b. An example was given in the laboratory about multiplying 5^2; therefore, I used both of these examples to implement the last task. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to debug, run and produce code effectively.