

## Oblig2:

### Programmet:

```
1 section .data
2 inpm db 'Enter a number: '
3 inpm_len equ $-inpm
4 outm db 'The result is: '
5 outm_len equ $-outm
6
7 section .bss
8 res resb 2
9
10 section .text
11 global _start
12
13 _start:
14
15 mov eax, 4
16 mov ebx, 1
17 mov ecx, inpm
18 mov edx, inpm_len
19 int 80h
20
21 mov eax, 3
22 mov ebx, 2
23 mov ecx, res
24 mov edx, 20
25 int 80h
26
27 ;; Task 2
28
29 mov eax, 4
30 mov ebx, 1
31 mov ecx, outm
32 mov edx, outm_len
33 int 80h
34
35 mov eax, 4
36 mov ebx, 1
37 mov ecx, res
38 mov edx, 7
39 int 80h
40
41 exit:
42 mov eax, 1
43 mov ebx, 0
44 int 80h
```

Figure 1: printnum.asm

### I terminalvinduet:

#### Lage fil:

- touch printnum.asm

#### Redigere filen:

- vi printnum.asm

#### Gjøre filen om til en kompilerbar fil:

- nasm -f elf -F dwarf -g printnum.asm

#### Gjøre filen kjørbar:

- ld -m elf\_i386 -o printnum printnum.o

```
186480@dat103:~/assembly/oblig2$ ./printnum
```

```
Enter a number: 186480
```

```
The result is: 186480
```

```
186480@dat103:~/assembly/oblig2$ _
```

Task1:

**1. Explain what Lines 3 and 5 do:**

- Line 3: We need to find out how long the text line is if we want to do something with it. The code \$-inpm takes where the current address increment in RAM where .data-section is saved, and subtracts the starting position to the variable **inpm**. For example \*name\* is at increment 820, and after the variable inpm is saved, the \*name\* is at 836. 836-820 is 16, then we know from 820 and 16 increments up is the text variable we want to use.
- Line 5: is the same as Line 3, just with an other sections of text. If we want to print the saved text variable to file or terminal, we need to start from were the text is saved, and increment \*name\* 16 times to print all the letters we want.

**2. In Line 24, if we replace 20 with 3, can we still read a 6 digit number and print it?**

- It will not work, because System Call: 3 is **sys\_read** and **edx** register expects a “count”, if we put 3 bytes count in the register it will only count up 3 bytes from the starting address of buffer to store input(**ecx**). Meaning 3 -letter string, in stead of 20.

**3. In Line 38, if we replace 7 with 3, what will happen?**

- Line 38 “mov ecx, res -> Refers to the section of uninitialized data. If we are to type in our student number 6 digits, there is need to print 7 because it need 1 extra space for the ASCII code “\n” which is automatically added when you press enter to submit your text. If you change to 6, it will print out all the numbers but not go to a new line. If you change it to 3, it will only print out 3 numbers, and also not go to new line. “mov ecx, res” refers to the storage were input data is, and then “mov edx, 7” is how many increments(1 byte per increment) it going to print out.

**4. If the instruction “add ecx, 3 “is inserted between Lines 37 and 38, what will the program now write to STDOUT?**

- Line 37 – “mov ecx, res”, now ecx register contains the address to were res is stored. If we write the code “add ecx, 3” it will add number 3 to the address number in the ecx register. It now contain an address with 3 more increments of bytes than before. It will then skip the first 3 original bytes, and write only 3-6, and not 0-6.

**5. What is the maximum length of the input number the program can write to STDOUT?**

- At Line 8 we reserve 2 bytes to the **resb** uninitialized storage. That means we will safely store 2 bytes of input later. The rest will overwrite the next 5 data address increments if there are data stored there with 6 digit number (7 with “new line” code after enter is pressed).

At line 24 – “mov edx, 20” we read 20 bytes increment from the address starting at **res**. Even though we only have safely reserved 2 bytes for storage of input. It will print whatever data is saved from the **res** address and 20 increments up. Maximum length of input which is printed out is 20.

Task 2:

```
1  mov  eax, 0
2
3  checkstart:
4  movzx   edx, byte [ecx]
5  inc    ecx
6  cmp    edx, '0'
7  jb     checkfinish
8  cmp    edx, '9'
9  ja     checkfinish
10 sub   edx, '0'
11 add   eax, edx
12 jmp   checkstart
13
14 checkfinish:
15 add  eax, '0'
16 mov  [res], eax
```

Line1: **mov eax, 0**

- Initialize the accumulator to Zero.

Line2: **checkstart:**

- Marking the top of a loop

Line3: **movzx edx, byte [ecx]**

- Load the *byte* at memory address ECX into EDX, zero-extend (upper 24 bits of EDX = 0). Now EDX = zero-extended(fill rest of the byte with 0) value of the character at [ECX] register.

The code “movzx” prevents sign extension and ensure EDX contains 0-255 bytes?

Line4: **inc ECX**

- Increment the pointer ECX to the next byte for the next iteration.

Line5: **cmp edx, '0'**

- Compare the byte read with '0' which has value 48 in ASCII table.

Line6: **jb checkfinish:**

- Jb = jump if below (unsigned). If EDX < '0' -> jump to checkfinish.

Line7: **cmp edx, '9'**

- Compare the bytes read with '9' which as the value 57 in ASCII table.

Line8: **jp checkfinish:**

- Jb = jump if below. If EDX < '9' -> jump to checkfinish.

In short: We are checking if the byte at EDX is between 0-9, in ASCII 48-57. Which means that if the byte stored is not a number it end the loop and jump to checkfinish.

Line 9: **sub edx, '0'**

- Subtract ASCII code for '0' from EDX, which gives us a number between 0-9.

Line10: **add eax, edx**

- Add the digit we got form the subtraction to EAX

Line11: **jmp checkstart**

- Jumps back to the start of the loop to get the next number.

Line 14: **checkfinish:**

- label for were the loop jump to if it encounter a byte which is not a number.

Line15: **add eax, '0'**

- This will add ASCII-48 to the existing number at EAX, to convert it back to a ASCII number. In this case there is a problem if the sum of digits is greater than 9, it will produce a non-digit character.

Line 16: **mov [res], eax**

- Stores EAX into memory at lable res. res = variable name(can be whatever you want).

```
186480@dat103:~/assembly/oblig2$ ./task2
```

```
Enter a number: 123
```

```
Task 3:
```

```
The result is: 6186480@dat103:~/assembly/oblig2$ _
```

```
186480@dat103:~/assembly/oblig2$ ./task3
```

```
Enter a number: 555
```

```
The result is: 15
```