LAB 1: ASSEMBLY x86

All code was successfully compiled and ran on Ubuntu 20.04 using the following command: nasm -f elf FILE.asm && ld -m elf_i386 -s -o FILE FILE.o
./FILE

Task 1: Print name and ID

CODE:

| | global | _start | |
|---------|---------|---------|---|
| _start: | section | .text | |
| _0.0.1. | mov | eax,4 | ;call the syscall for write |
| | mov | ebx,1 | ;write to stdout |
| | mov | | ; ;pass the string to be written to |
| stdout | | | |
| | mov | edx, 49 | ;specify the length of the string |
| | int 80h | | ;request interrupt |
| | | | |
| exit: | | | |
| | mov | eax, 1 | ;call the syscall for exit |
| | mov | ebx, 0 | ;return 0 on exit if no errors occurred |
| | int 80h | | ;request interrupt |
| | | | |
| | | | |
| | section | .data | |

2

message: db "Kristina Ilyovska, ID: 23857351, Section ID: CC1",

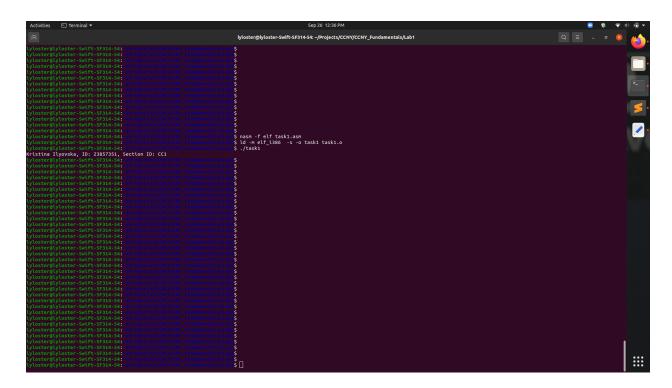
0Ah ;the string to be written to stdout

EXPLANATION:

The code in sequential order does the following:

- Puts the system call for Write in register EAX
- Puts the descriptor for writing to STDOUT in register EBX
- Puts the string pointer of massage in register ECX
- Puts the length of message in register EDX
- Requests an interrupt from the system
- On exit, assigns 1 to EAX if the program executed successfully and 0 to EBX if the problem did not execute successfully

OUTPUT:



Task 2: Implement strlen function to find the length of a string to be printed

CODE:

| | global | _start | |
|----------|----------------------------------|--|---|
| at anti- | section | .text | |
| _start: | push call add | message strlen esp, 4 | ;push the string pointer to the stack ;call strlen function ;move past the 'message' on the stack |
| | mov mov | edx, eax eax,4 ebx,1 | ;grab the length of the string from eax ;call the syscall for write ;write to stdout |
| | mov int 80h | • | ;pass the string to be written to stdout ;request interrupt |
| exit: | | | |
| | mov mov int 80h | eax, 1 ebx, 0 | ;call the syscall for exit ;return 0 on exit if no errors occurred ;request interrupt |
| strlen: | | | |
| | push mov sub mov mov | ebp, esp esp, 4 ecx, [ebp + 8] eax, 0 | ;push base pointer to stack ;move esp to ebp ;move down to next address in stack ;put message from stack to ecx register ;set a counter to 0 in eax |
| | | | |

4

beginning: ;while loop

cmp byte[ecx + eax], 0x00 ;check if current char is null

je end ;if yes, end the loop

inc eax ;if not, increment the value in eax

jmp beginning ;return to beginning

end:

mov esp, ebp ;move ebp to esp
pop ebp ;pop ebp from stack

ret

section .data

message: db "Kristina Ilyovska, ID: 23857351, Section ID: CC1",

0Ah ; the string to be written to stdout

EXPLANATION:

The code in sequential order does the following:

- _start
 - Push the string pointer to the stack
 - Call strlen function
 - Move the stack pointer past the string pointer on the stack, done instead of popping the 'message'
 - Save the length of the string, returned from the strlen function from EAX to EDX
 - Puts the system call for Write in register EAX
 - Puts the descriptor for writing to STDOUT in register EBX
 - Puts the string pointer of massage in register ECX
 - Puts the length of message in register EDX
 - Requests an interrupt from the system
 - On exit, assigns 1 to EAX if the program executed successfully and 0 to EBX if the problem did not execute successfully
- strlen:
- Push the EBP, base pointer to the stack

- Move the stack pointer ESP to EBP
- Move ESP down to the 'message' on the stack
- Save the pointer to the message from the stack, dereferencing the EBP + 8
 location in memory in ECX
- Initialize a counter to 0 in EAX, the current length of the 'message'
- Begin while loop
- Compare the current character to NULL
- If the current character is NULL, exit the while loop
- If the current character is not NULL, increase the value in EAX
- Go back to beginning of while loop, until the length of 'message' is found
- Move the EBP to ESP
- Pop EBP from stack
- Return to the location where strlen was called in _start, the length of the string is saved in EAX

OUTPUT:

