

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

section .text

_start:
    mov     eax,4      ;call the syscall for write
    mov     ebx,1      ;write to stdout
    mov     ecx, message ;pass the string to be written to
    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
```

```
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 message 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:

[illegible]

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

CODE:

```

                global      _start

                section     .text

_start:

    push        message    ;push the string pointer to the stack
    call        strlen     ;call strlen function
    add         esp, 4      ;move past the 'message' on the stack

    mov         edx, eax    ;grab the length of the string from eax
    mov         eax, 4      ;call the syscall for write
    mov         ebx, 1      ;write to stdout
    mov         ecx, message ;pass the string to be written to stdout
    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

strlen:

    push        ebp        ;push base pointer to stack
    mov         ebp, esp   ;move esp to ebp
    sub         esp, 4      ;move down to next address in stack
    mov         ecx, [ebp + 8] ;put message from stack to ecx register
    mov         eax, 0      ;set a counter to 0 in eax

```

```

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 message 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:

[illegible]