**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

Laboratory work 4:

Familiarize with the basics of Assembly Language

Elaborated:

st. gr. FAF-212 Cara Alexandr

Verified:

asist. Univ. Vladislav Voitcovschi

Chişinău - 2023

## <https://github.com/CaraAlexandr/AC-Labs.git> Install nasm Using apt-get

Update apt database with apt-get using the following command.

sudo apt-get update

After updating apt database, We can install nasm using apt-get by running the following command:

sudo apt-get -y install nasm

## 

**1. Hello world**

This code is a simple NASM assembly language program that writes "Hello, World" to the console using only system calls. It contains two sections: .text and .data. The .text section contains the actual code, while the .data section contains the message to be outputted.

The global \_start line declares the program entry point, and the section .text directive defines the start of the code section. The first instruction, mov rax, 1, loads the system call number for write into the rax register. The next three instructions, mov rdi, 1, mov rsi, message, and mov rdx, 13, set the arguments for the write system call. These arguments specify that the output should be directed to standard output (rdi = 1), that the address of the message should be used as the buffer (rsi = message), and that the length of the message is 13 bytes (rdx = 13). The syscall instruction then invokes the operating system to perform the write.

The next two instructions, mov rax, 60 and xor rdi, rdi, set up the exit system call with an exit code of 0. The syscall instruction is then used to exit the program.

Finally, the .data section contains the message to be outputted, "Hello, World", followed by a newline character (10 in ASCII).

To assemble and run the program, the following commands should be executed in a terminal:  
**nasm -felf64 hello.asm && ld hello.o && ./a.out**

a.out

**2.Echo Input**

; ----------------------------------------------------------------------------------------; To assemble and run:;; nasm -f elf64 -o echo\_input.o echo\_input.asm&&ld -o echo\_input echo\_input.o&&./echo\_input; ----------------------------------------------------------------------------------------global \_startsection .data; File descriptors for stdin and stdout*STDIN* equ 0*STDOUT* equ 1; System call numbers for sys\_read and sys\_write*SYS\_READ* equ 0*SYS\_WRITE* equ 1; System call number for sys\_exit*SYS\_EXIT* equ 60; Buffer size for reading input*BUFFER\_SIZE* equ 256section .bss; Buffer for storing inputbuffer: resb *BUFFER\_SIZE*section .text\_start:; Read input from the usermov rax, *SYS\_READ* ; sys\_readmov rdi, *STDIN* ; file descriptor for stdinmov rsi, buffer ; address of the buffermov rdx, *BUFFER\_SIZE* ; buffer sizesyscall; Store the number of bytes read in rcxmov rcx, rax; Write the input to the screenmov rax, *SYS\_WRITE* ; sys\_writemov rdi, *STDOUT* ; file descriptor for stdoutmov rsi, buffer ; address of the buffermov rdx, rcx ; number of bytes to writesyscall; Exit the programmov rax, *SYS\_EXIT* ; sys\_exitmov rdi, 0 ; exit status 0 (success)syscall

This code is a NASM assembly language program that reads input from the user and echoes it back to the console using system calls. It consists of three sections: .data, .bss, and .text.

The .data section contains constants used in the program, including file descriptors and system call numbers. The .bss section reserves space in memory for the input buffer, which is used to store the input read from the user. The .text section contains the actual program code.

The global \_start line declares the program entry point, and the \_start label marks the beginning of the program.

The first system call in the program is sys\_read, which reads input from the user. The mov instructions before the syscall instruction set the arguments for the sys\_read system call. rax is loaded with the system call number, rdi is set to the file descriptor for standard input (STDIN), rsi is set to the address of the buffer, and rdx is set to the buffer size. The syscall instruction then invokes the operating system to perform the read.

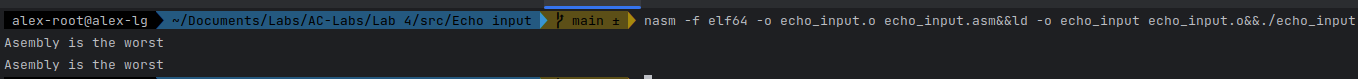
The number of bytes read is stored in the rcx register. The program then uses another mov instruction to set up the sys\_write system call, which writes the input to the console. The arguments for this system call are similar to those for sys\_read, with rax set to the system call number, rdi set to the file descriptor for standard output (STDOUT), rsi set to the address of the buffer, and rdx set to the number of bytes to write (rcx).

Finally, the program uses the sys\_exit system call to exit. The rax register is set to the system call number for sys\_exit, and rdi is set to the exit status (0 for success).

To assemble and run the program, the following commands should be executed in a terminal:

nasm -f elf64 -o echo\_input.o echo\_input.asm && ld -o echo\_input echo\_input.o && ./echo\_input

This will assemble the program, link it into an executable file, and then run the resulting executable. The program will read input from the user and echo it back to the console.

  
**3.Arithmetic operations**

; ----------------------------------------------------------------------------------------  
; To assemble and run:  
;  
; nasm -f elf64 -o arithmetic\_operations.o arithmetic\_operations.asm&&ld -o arithmetic\_operations arithmetic\_operations.o&&./arithmetic\_operations  
; ----------------------------------------------------------------------------------------  
  
  
*SYS\_EXIT* equ 1  
*SYS\_READ* equ 3  
*SYS\_WRITE* equ 4  
*STDIN* equ 0  
*STDOUT* equ 1  
  
*segment* .data  
  
 msg db "Please enter a digit ", 0xA,0xD  
 *len* equ $- msg  
 newline db 0xA  
  
*segment* .bss  
  
 number1 resb 2  
 number2 resb 2  
 result resb 1  
 result2 resb 1  
  
*segment* .text  
  
 msg2 db "Please enter a second digit", 0xA,0xD  
 *len2* equ $- msg2  
  
 msg3 db "The sum is: "  
 *len3* equ $- msg3  
  
 msg4 db "The minus is: "  
 *len4* equ $- msg4  
  
global \_start  
  
\_start:  
  
 mov eax, *SYS\_WRITE* ; System write  
 mov ebx, *STDOUT* ; System output  
 mov ecx, msg ; What to write  
 mov edx, *len* ; Length to write  
 int 0x80 ; Interupt Kernel  
  
 mov eax, *SYS\_READ* ; System read  
 mov ebx, *STDIN* ;  
 mov ecx, number1  
 mov edx, 2  
 int 0x80  
  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, msg2  
 mov edx, *len2*  
int 0x80  
  
 mov eax, *SYS\_READ*  
mov ebx, *STDIN*  
mov ecx, number2  
 mov edx, 2  
 int 0x80  
  
 call add  
 call minus  
  
 ; Exit the program  
 mov eax, *SYS\_EXIT*  
xor ebx, ebx  
 int 0x80  
  
add:  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, msg3  
 mov edx, *len3*  
int 0x80  
  
 mov eax, [number1]  
 sub eax, '0'  
 mov ebx, [number2]  
 sub ebx, '0'  
  
 add eax, ebx  
  
 ; Check if the sum is greater than 9  
 cmp eax, 10  
 jl short single\_digit\_sum  
  
 ; If the sum is greater than 9, handle two-digit numbers  
 xor edx, edx  
 mov ebx, 10  
 div ebx  
  
 add eax, '0'  
 mov [result], al  
 add edx, '0'  
 mov [*result*+1], dl  
 inc edi  
  
 ; Adjust the length to write 2 characters  
 mov edx, 2  
 jmp short print\_sum\_result  
  
single\_digit\_sum:  
 add eax, '0'  
 mov [result], al  
  
 ; Adjust the length to write 1 character  
 mov edx, 1  
  
print\_sum\_result:  
  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, result  
 mov [result+edx], byte 0xA ; Add newline character after the sum result  
 add edx, 1 ; Increment length to include newline character  
 int 0x80  
  
  
minus:  
  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, msg4  
 mov edx, *len4*  
int 0x80  
  
 ; Load number1 into eax and subtract '0' to convert from ASCII to decimal  
 mov eax, [number1]  
 sub eax, '0'  
 ; Do the same for number2  
 mov ebx, [number2]  
 sub ebx, '0'  
  
 ; Subtract ebx from eax, storing the result in eax  
 sub eax, ebx  
 ; Add '0' to eax to convert the digit from decimal to ASCII  
 add eax, '0'  
  
 ; Store the result in result2  
 mov [result2], eax  
  
 ; Print the result digit  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, result2  
 mov edx, 1  
 int 0x80  
  
 ; Print the newline character  
 mov eax, *SYS\_WRITE*  
mov ebx, *STDOUT*  
mov ecx, newline  
 mov edx, 1  
 int 0x80  
  
 ret  
  
  
exit:  
 mov eax, *SYS\_EXIT*  
xor ebx, ebx  
 int 0x80

This is a NASM assembly language program that performs arithmetic operations on user input and displays the results. It contains three segments: .data, .bss, and .text.

The .data segment contains several messages to prompt the user for input and display the results of the arithmetic operations. The .bss segment reserves space in memory for the input and output variables. The .text segment contains the actual program code.

The program prompts the user to enter two digits, reads them in, and then performs two operations: addition and subtraction. The results of these operations are then displayed.

The mov instructions before the int 0x80 instructions set up the system calls for reading input from the user, writing output to the console, and exiting the program. The program uses int 0x80 to call the Linux kernel to perform these system calls.

The add subroutine performs addition on the two input digits. It first converts the ASCII characters representing the digits into decimal values using sub, then performs the addition using the add instruction. If the sum is greater than 9, the program handles two-digit numbers by using div to separate the tens and units places. The results are then converted back to ASCII using add and stored in the result variable. The results are then displayed using another system call to write to the console.

The minus subroutine performs subtraction on the two input digits. It first converts the ASCII characters representing the digits into decimal values using sub, then performs the subtraction using the sub instruction. The result is then converted back to ASCII using add and stored in the result2 variable. The result is then displayed using two system calls to write the result digit and a newline character to the console.

Finally, the program exits using the sys\_exit system call.

To assemble and run the program, the following commands should be executed in a terminal:

nasm -f elf64 -o arithmetic\_operations.o arithmetic\_operations.asm && ld -o arithmetic\_operations arithmetic\_operations.o && ./arithmetic\_operations

This will assemble the program, link it into an executable file, and then run the resulting executable. The program will prompt the user to enter two digits, perform addition and subtraction, and then display the results.

