



Universidad Autónoma de Baja California

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA



Organización y Arquitectura de las Computadoras.

## Práctica No. 2: Introducción a ensamblador.

Alumno:

**Joshua Osorio O. – 1293271**

Docente:

**José Isabel García R.**

13/02/2026

## Tabla de contenido

Objetivo .....	2
Descripción de la practica.....	2
Lista de materiales .....	2
Explicación paso a paso .....	6
Conclusiones .....	15
Dificultades .....	16

## Objetivo

Distinguir las características de la organización y arquitectura del microprocesador de una computadora de propósito general, analizando sus recursos de hardware y software, para conocer capacidades y limitaciones de forma organizada y responsable.

## Descripción de la practica

### Lista de materiales

- Computadora
- Conexión a internet
- Git
- Vscod

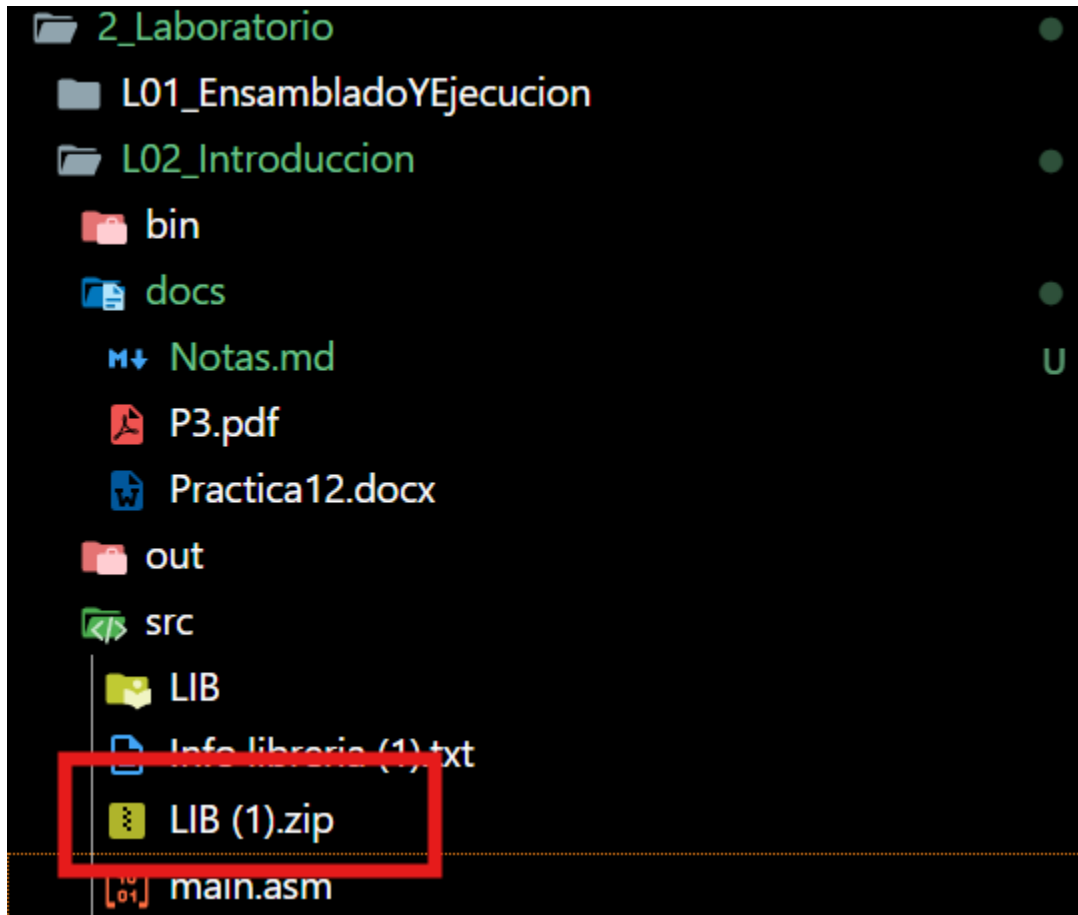
Se cargó en GitHub el código de la práctica anterior y se verificó su correcto funcionamiento.

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.

```
1 %include "./pc_io.inc"
2
3 section .data ;Datos inicializados
4     msg1: db "Ingresa tu nombre",10,0
5     msg2: db "Hola ",0
6     msgj: db 'Ingresa un digito (0-9)',0x0A
7     len: equ $-msgj
8
9 section .bss ;Datos no inicializados
10    num1 resb 1
11    nombre resb 256
12
13 section .text
14    global _start
15
16    _start:
17        mov edx, msg1 ;Imprimir mensaje 1
18        call puts
19
```

```
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
$ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a
$ ./bin/main
Ingresa tu nombre
Joshua ^?
Hola Joshua
$
```

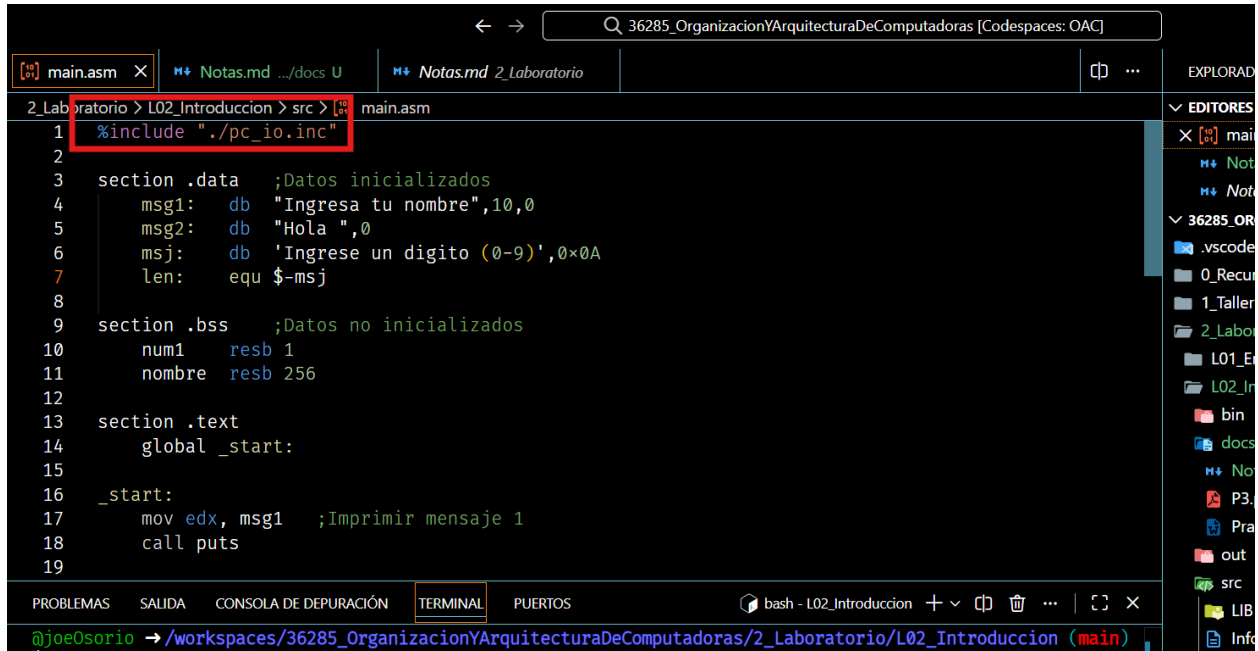
Se descargó el archivo lib.zip desde Moodle, se descomprimió y se colocaron los dos archivos contenidos en la carpeta de trabajo de GitHub.



En la parte superior del archivo nombre\_Practica.asm se agregó la instrucción:

```
%include "./pc_io.inc"
```

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.



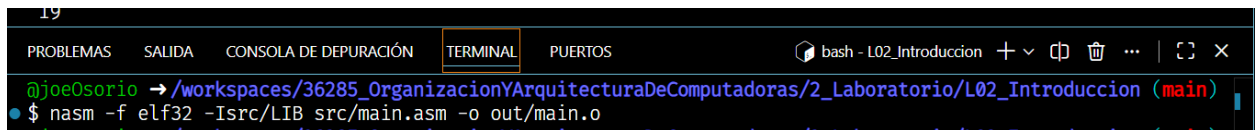
```
1 %include "./pc_io.inc"
2
3 section .data ;Datos inicializados
4     msg1: db "Ingresa tu nombre",10,0
5     msg2: db "Hola ",0
6     msj: db "Ingresa un digito (0-9)",0x0A
7     len: equ $-msj
8
9 section .bss ;Datos no inicializados
10     num1 resb 1
11     nombre resb 256
12
13 section .text
14     global _start
15
16 _start:
17     mov edx, msg1 ;Imprimir mensaje 1
18     call puts
19
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** PUERTOS

@joeOsoerio → /workspaces/36285\_OrganizacionYArquitecturaDeComputadoras/2\_Laboratorio/L02\_Introduccion (main)

El código se ensambló con el comando:

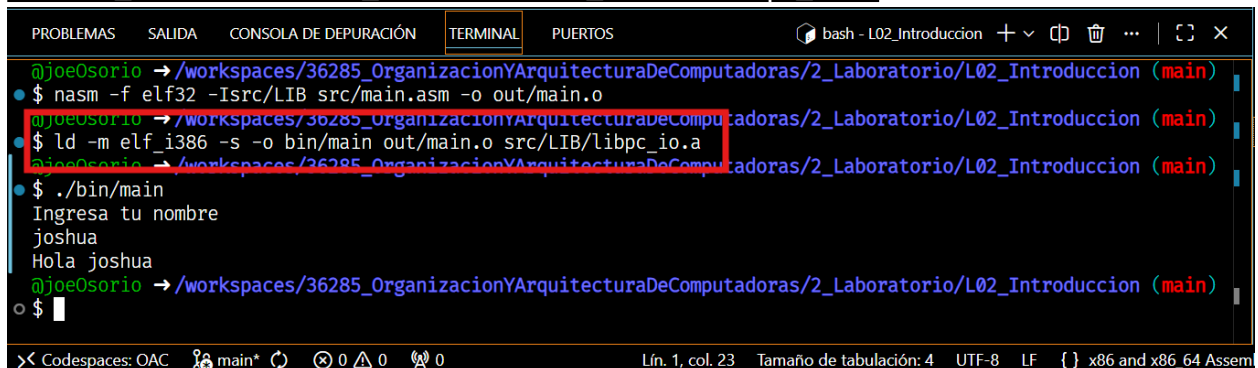
```
nasm -f elf nombre_Practica.asm
```



```
19
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
bash - L02_Introduccion + v [ ] [ ] ... | [ ] x
@joeOsoerio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
```

Se enlazó el objeto generado con el comando:

```
ld -m elf_i386 -s -o nombre_Practica nombre_Practica.o libpc_io.a
```



```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
bash - L02_Introduccion + v [ ] [ ] ... | [ ] x
@joeOsoerio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
@joeOsoerio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a
@joeOsoerio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ ./bin/main
Ingresa tu nombre
joshua
Hola joshua
@joeOsoerio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$
```

> Codespaces: OAC main\* 0 0 0 Lín. 1, col. 23 Tamaño de tabulación: 4 UTF-8 LF { } x86 and x86\_64 Assem

Finalmente, el programa se ejecutó con:

```
./nombre_Practica
```

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
• $ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
• $ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
• $ ./bin/main
  Ingresar tu nombre
  joshua
  Hola joshua
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
○ $ 

```

## Explicación paso a paso

En el código se deben agregar las siguientes partes:

1. En él .data agregar una constante de texto:

```

msj:    db  'Ingrese un dígito (0-9)',0x0A
len:    equ $-msj

```

Esto define una cadena de texto con salto de línea (0x0A) y una constante len que guarda la longitud del mensaje.

2. En él .bss agregar una variable:

```

num1    resb 1

```

Aquí se reserva un byte para almacenar el valor ingresado.

3. En él .text se debe colocar el siguiente código:

Código1:

```

section .text
    mov eax, 4
    mov ebx, 1
    mov ecx, msj
    mov edx, len
    int 80h
    global _start:

```

¿Qué piensa usted que hará este código?

Carga el registro eax con un 4 decimal.

Carga el registro ebx con 1 decimal.

Carga el registro ecx con la dirección de la variable msj

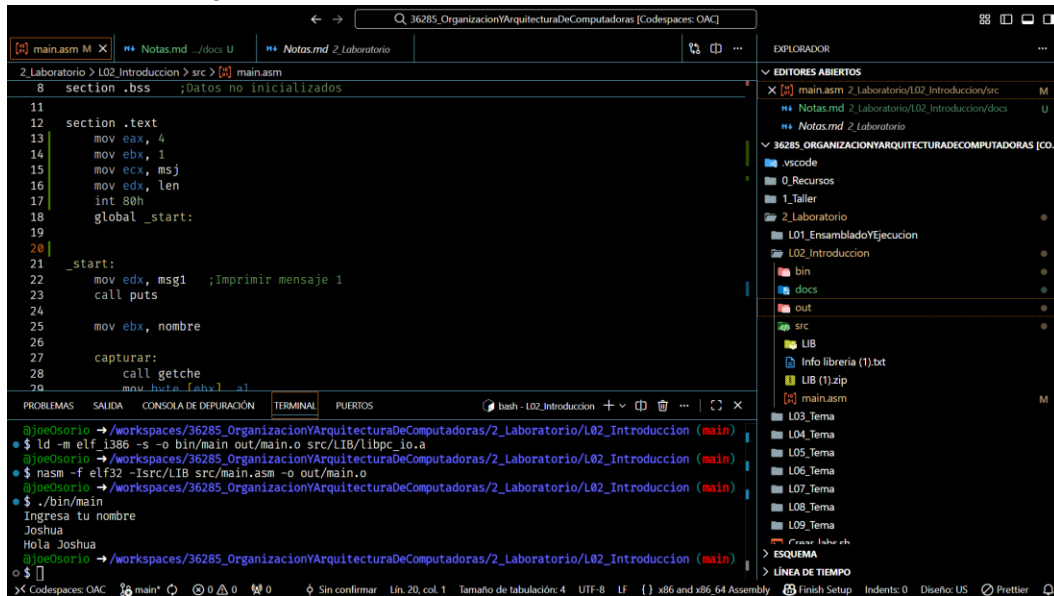
Carga el registro edx con la dirección de la variable len.

Finaliza programa.

¿Qué hizo realmente este código?

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.

Coloque el codigo despues del .text pero no cambio nada.



The screenshot shows the Visual Studio Code editor with the file `main.asm` open. The code defines a `.bss` section for uninitialized data and a `.text` section for instructions. The instructions load `eax` with 4, `ebx` with 1, `ecx` with `msj`, and `edx` with `len`, then execute `int 80h`. The `_start` function calls `puts` to print a message, then prompts the user for their name and prints it. The terminal shows the execution of `nasm` and `ld` to create the executable `main.o` and `main`, followed by running `./bin/main`, which outputs "Ingresa tu nombre" and "Hola Joshua".

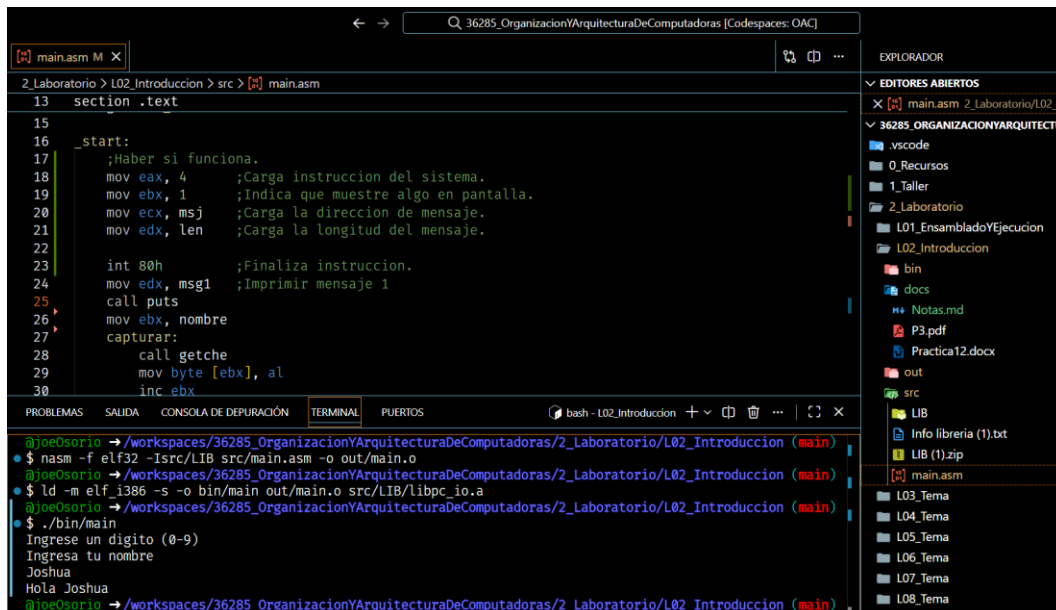
```
8 section .bss ;Datos no inicializados
11
12 section .text
13 mov eax, 4
14 mov ebx, 1
15 mov ecx, msj
16 mov edx, len
17 int 80h
18 global _start:
19
20
21 _start:
22 mov edx, msg1 ;Imprimir mensaje 1
23 call puts
24
25 mov ebx, nombre
26
27 capturar:
28 call getche
29 mov byte [ebx], al
30
```

```
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
$ ./bin/main
Ingresa tu nombre
Joshua
Hola Joshua
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$
```

Coloque el código después de `_start`: y aparece el mensaje.

Así que lo que realizo el Código fue:

```
mov eax, 4 ;Carga instrucción del sistema.
mov ebx, 1 ;Indica que muestre algo en pantalla.
mov ecx, msj ;Carga la dirección de mensaje.
mov edx, len ;Carga la longitud del mensaje.
int 80h ;Ejecuta La instrucción.
```



The screenshot shows the Visual Studio Code editor with the file `main.asm` open. The code defines a `.text` section for instructions. The instructions load `eax` with 4, `ebx` with 1, `ecx` with `msj`, and `edx` with `len`, then execute `int 80h`. The `_start` function calls `puts` to print a message, then prompts the user for their name and prints it. The terminal shows the execution of `nasm` and `ld` to create the executable `main.o` and `main`, followed by running `./bin/main`, which outputs "Ingresa un dígito (0-9)" and "Ingresa tu nombre" and "Hola Joshua".

```
13 section .text
15
16 _start:
17 ;Haber si funciona.
18 mov eax, 4 ;Carga instruccion del sistema.
19 mov ebx, 1 ;Indica que muestre algo en pantalla.
20 mov ecx, msj ;Carga la direccion de mensaje.
21 mov edx, len ;Carga la longitud del mensaje.
22
23 int 80h ;Finaliza instruccion.
24 mov edx, msg1 ;Imprimir mensaje 1
25 call puts
26 mov ebx, nombre
27 capturar:
28 call getche
29 mov byte [ebx], al
30 inc ebx
```

```
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
$ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a
$ ./bin/main
Ingresa un dígito (0-9)
Ingresa tu nombre
Joshua
Hola Joshua
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main)
$
```

Código2:

```

mov eax, 3
mov ebx, 0
mov ecx, num1
mov edx, 1
int 80h

```

¿Qué piensa usted que hará este código?

```

mov eax, 3      -> Carga una instrucción al sistema.
mov ebx, 0      -> Carga la instrucción de captura.
mov ecx, num1   -> Carga la dirección donde se almacenará.
mov edx, 1      -> Indica la longitud de la variable.
int 80h         -> Fin de la instrucción.

```

¿Qué hizo realmente este código?

```

mov eax, 3      ;carga instruccion al sistema.
mov ebx, 0      ;Instruccion captura.
mov ecx, num1   ;Direccion de variable.
mov edx, 1      ;Longitud de variable.
int 80h         ;Ejecuta instruccion.

```

The screenshot shows a Visual Studio Code editor with a file named `main.asm` open. The code in the editor is as follows:

```

15
16
17 start:
18 ;Codigo 1.
19 ; mov eax, 4 ;Carga instruccion del sistema.
20 ; mov ebx, 1 ;Indica que muestre algo en pantalla.
21 ; mov ecx, msg ;Carga la direccion de mensaje.
22 ; mov edx, len ;Carga la longitud del mensaje.
23 ; int 80h ;Finaliza instruccion.
24
25 ;codigo 2.
26 mov eax, 3 ;carga instruccion al sistema.
27 mov ebx, 0 ;Instruccion captura.
28 mov ecx, num1 ;Direccion de variable.
29 mov edx, 1 ;Longitud de variable.
30 int 80h ;Ejecuta instruccion.
31
32 ;Inicio de codigo.
33 mov edx, msg1 ;Imprimir mensaje 1
34 call puts

```

The terminal window shows the following commands and output:

```

@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ nasm -f elf32 -Isrc/L
IB src/main.asm -o out/main.o
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ld -m elf_i386 -s -o
bin/main out/main.o src/LIB/libc_io.a
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
1
Ingresa tu nombre
Hola
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $

```



The screenshot shows the Visual Studio Code interface. The main editor window displays the file `main.asm` with the following assembly code:

```

15
16 _start:
17     ;Codigo 1.
18     mov eax, 4      ;Carga instruccion del sistema.
19     mov ebx, 1      ;Indica que muestre algo en pantalla.
20     mov ecx, msg1   ;Carga la direccion de mensaje.
21     mov edx, len     ;Carga la longitud del mensaje.
22     int 80h         ;Finaliza instruccion.
23
24     ;codigo 2.
25     mov eax, 3      ;carga instruccion al sistema.
26     mov ebx, 0      ;Instruccion captura.
27     mov ecx, num1   ;Direccion de variable.
28     mov edx, 2      ;Longitud de variable.
29     int 80h         ;Ejecuta instruccion.
30
31     ;Inicio de codigo.
32     mov edx, msg1   ;Imprimir mensaje 1
33     call puts

```

The terminal window shows the execution of the program:

```

@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ nasm -f elf32 -Isrc/L
IB src/main.asm -o out/main.o
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ld -m elf_i386 -s -o
bin/main out/main.o src/LIB/libpc_io.a
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingresa un dígito (0-9)
1
Ingresa tu nombre
Joshua
Hola Joshua
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $

```

El compañero de la Peña me ayudó a resolver la incógnita de porque se comporta raro el código, me cuestiono.

Código 3:

```

mov eax, 4
mov ebx, 1
mov ecx, num1
mov edx, 1
int 80h

```

¿Qué piensa usted que hará este código?

```

mov eax, 4      ;Carga instruccion para el sistema
mov ebx, 1      ;Carga la instruccion de mostrar en pantalla.
mov ecx, num1   ;Direccion de variable a imprimir
mov edx, 1      ;Logitud de variable.
int 80h         ;Ejecutar instruccion

```

Mostrar en pantalla lo capturado en la variable num1.

¿Qué hizo realmente este código?

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.

```

17 section .text
18
19 _start:
20
21 ;Codigo 1.
22 mov eax, 4 ;Carga instruccion del sistema.
23 mov ebx, 1 ;Indica que muestre algo en pantalla.
24 mov ecx, msg ;Carga la direccion de mensaje.
25 mov edx, len ;Carga la longitud del mensaje.
26 int 80h ;Finaliza instruccion.
27
28 ;Codigo 2.
29 mov eax, 3 ;carga instruccion al sistema.
30 mov ebx, 0 ;Instruccion captura.
31 mov ecx, num1 ;Direccion de variable.
32 mov edx, 2 ;longitud de variable.
33 int 80h ;Ejecuta instruccion.
34
35 ;Codigo 3
36 mov eax, 4 ;Carga instruccion para el sistema
37 mov ebx, 1 ;Carga la instruccion de mostrar en pantalla.
38 mov ecx, num1 ;Direccion de variable a imprimir
39 mov edx, 1 ;logitud de variable.
40 int 80h ;Ejecutar instruccion
41
42 ; mov edx, msg
43 ; call puts
44
45 ;Inicio de codigo base.
46 mov edx, msg1 ;imprimir mensaje 1

```

PROBLEMAS SALIDA CONSOLA DE DEPURACION TERMINAL PUERTOS

```

@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)1
1Ingresa tu nombre
Jioshua
Hola Jioshua
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $

```

Mostro en pantalla lo capturado en la variable num1. Sin hacer enter y retorno.

```

@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libc_io.a
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)1
1Ingresa tu nombre
Jioshua
Hola Jioshua
@joeosorio → /workspaces/36285_OrganizacionYArquitecturaDeComputadoras/2_Laboratorio/L02_Introduccion (main) $

```

- Del paso anterior se debe explicar textualmente que se piensa que hace cada sección de código, con sus propias palabras.

Codigo 1: Muestra en pantalla el contenido de la variable msg y da un salto de línea.

Codigo 2: Captura y guardar valor en variable 1

Codigo 3: Muestra el contenido de la variable num1

- Ahora el código anterior lo vamos a modificar:

- El msg ahora debe quedar de la siguiente manera:

```
msg: db 'Ingresa un digito (0-9)',0x0
```

- El código del paso 3 se debe comentar y agregar el siguiente:

¿Qué piensa usted que hará este código?

```
mov edx, msj
call puts
```

Mueve la dirección al registro edx y luego la subrutina o funcione se encarga de mostrarlo en terminal.

¿Qué hizo realmente este código?

Mostro el mensaje en terminal, cargo la dirección de la variable msj en el registro edx y después con la función puts lo mostro en terminal.

¿Qué piensa usted que hará este código?

```
call getche
```

Capturar algún carácter.

¿Qué hizo realmente este código?

Esta mostrando el ultimo contenido de al  
En mi caso está guardando un 0.

The screenshot shows a Visual Studio Code editor with the following components:

- Editor:** Displays assembly code for `main.asm`. The code includes comments and instructions for moving memory addresses to registers and calling `puts` and `getche` functions.
- Terminal:** Shows the execution of the assembly code. It displays the command `nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o` and the command `ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc.io.a`. The output shows the program running and displaying "Hola" followed by a prompt "Ingrese un dígito (0-9)".
- Explorer:** Shows the file structure of the project, including `main.asm`, `Info librería (1).txt`, and `LIB (1).zip`.

¿Qué piensa usted que hará este código?

```
call putchar
```

¿Qué hizo realmente este código?

Muestra el contenido capturado por el teclado

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.

```

46 ; mov ecx, num1 ; direccion de variable a imprimir
47 ; mov edx, 1 ; longitud de variable.
48 ; int 80h ; Ejecutar instruccion
49
50 ; Parte 5.b practica
51 mov ecx, msg
52 call puts
53
54 call getche
55 call putchar
56
57 mov ebx, nombre
58 ;
59 capturar:
60 call getche
61 mov byte [ebx], al
62 inc ebx
63
64 cmp al, 10
65 jne capturar
66 mov byte [ebx], 0
67
68 mov edx, msg2
69 call puts
70
71 mov edx, nombre
72 call puts
73 call putchar
74
75 mov eax, 1 ; Carga la instruccion de salida de programa.
76 mov ebx, 0 ; Indica que termino correctamente, como un return 0 en c.
77 int 80h ; llamada a kernel con las anteriores mensajes.

```

```

@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
@joeOsorio → /workspaces/36285_OrganizacionYArqu
doras/2_Laboratorio/L02_Introduccion (main) $ nasm -f elf32 -isrc
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ld -m elf_i386 -s -
o bin/main out/main.o src/LIB/libpc_io.a
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)
11
Hola
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $

```

6. En la parte inferior del código, vamos a agregar la siguiente función:

```

salto:
    mov al, 13
    call putchar
    mov al, 10
    call putchar
    ret

```

7. Ahora el código se modifica con salto y debe quedar de la siguiente manera:

¿Qué piensa usted que hará todo este código?

```

    mov edx, msg
    call puts
    call salto
    call getch
    call salto
    call putchar
    call salto

```

- Se carga la dirección en el registro edx.
- Se muestra el mensaje con la rutina puts.
- Se genera un salto de línea.
- Se captura un carácter.
- Se genera un salto de línea.
- Se muestra lo capturado.
- Se genera un salto de línea.

¿Qué hizo realmente todo este código?

Realizo lo pensado y mencionado anteriormente.

The screenshot shows a code editor with assembly code on the left and a terminal window on the right. The assembly code includes instructions like `call putchar`, `mov edx, msg`, `call puts`, `call salto`, `call getch`, `call salto`, `call putchar`, `call salto`, `mov ebx, nombre`, `capturar:`, `call getche`, `mov byte [ebx], al`, `inc ebx`, `cmp al, 10`, `jne capturar`, `mov byte [ebx], 0`, `mov edx, msg2`, `call puts`, `mov edx, nombre`, `call puts`, and `call putchar`. The terminal window shows the output of the program, including "Hola 1" and "Hola 2".

## 8. Vamos y modificamos la función de salto:

```
salto:
    pushad
    mov al, 13
    call putchar
    mov al, 10
    call putchar
    popad
    ret
```

Agrega las banderas `pushad` y `popad` que indica donde inicia y donde termina la función.

¿Qué hizo realmente todo este código?

Agrego un salto después de mostrar el mensaje.

Captura un carácter.

Muestra el carácter.

Genera un salto y captura.

## 9. Ahora agregamos la funcionalidad de capturar una letra, por ejemplo la a y sumarle un 1 e imprimirla en terminal.

Agregamos el siguiente código después del último `call salto`

¿Qué piensa usted que hará todo este código?

```
mov ebx, num1
mov byte[ebx], al
add byte[ebx], 1
mov al, byte[ebx]
call putchar
```

Copia la dirección de `num1` a `ebx`.

## UABC\_FCQI\_ORGANIZACION\_Y\_ARQUITECTURA\_DE\_COMPUTADORAS.

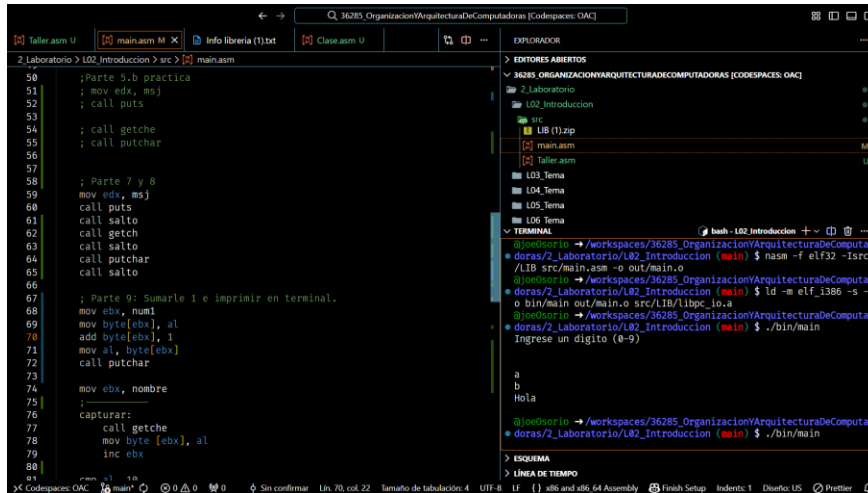
Copia el contenido de 'al' al contenido de ebx casteando un byte.

Suma 1 al contenido de ebx casteado a byte.

Muestra el contenido.

¿Qué hizo realmente todo este código?

Realizo lo mencionado anteriormente sin embargo cabe destacar que putchar muestra el contenido de 'al'



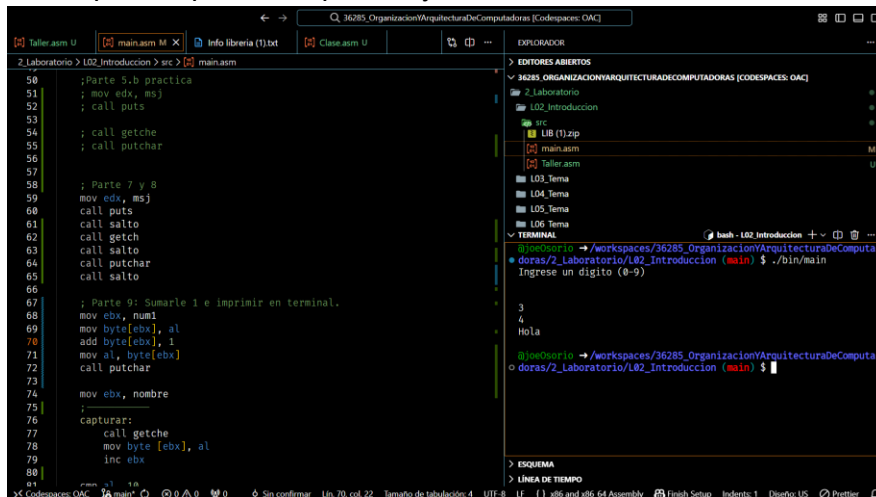
The screenshot shows a NASM assembler IDE with the following assembly code in the main window:

```
50 ;Parte 5.b practica
51 ; mov edx, msj
52 ; call puts
53
54 ; call getche
55 ; call putchar
56
57
58 ; Parte 7 y 8
59 mov edx, msj
60 call puts
61 call salto
62 call getch
63 call salto
64 call putchar
65 call salto
66
67 ; Parte 9: Sumarle 1 e imprimir en terminal.
68 mov ebx, num1
69 mov byte[ebx], al
70 add byte[ebx], 1
71 mov al, byte[ebx]
72 call putchar
73
74 mov ebx, nombre
75 ;
76 capturar:
77 call getche
78 mov byte[ebx], al
79 inc ebx
80
```

The terminal window shows the following output:

```
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ nasm -f elf32 -isrc
/LIB src/main.asm -o out/main.o
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ld -m elf_i386 -s -i
o bin/main out/main.o src/LIB/libc.lib -o
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)
a
b
Hola
```

10. Ahora para la prueba capture 3 y súmele 1 como el caso anterior.



The screenshot shows the same NASM assembler IDE with the same assembly code as before. The terminal window shows the following output:

```
@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)
3
4
Hola
```

11. Ahora capture otro número, modifique lo necesario para ello, y pruebe con una suma de 3 + 4 = 7:

¿Qué piensa usted que hará todo este código?

```
mov ebx, num1
mov al, [ebx]
mov ebx, num2
add [ebx], al
mov al, byte[ebx]
call putchar
```

Copia la dirección de num1 al registro ebx.

Copia el valor de ebx al registro 'al'.

Copia la dirección de num2 a ebx.

Suma el contenido de 'al' al contenido de ebx.

Copia el contenido casteado a byte del registro ebx al registro 'al'.

Muestra el contenido en terminal del registro 'al'.

¿Qué hizo realmente todo este código?

```

67
68 ; Parte 9: Sumarle 1 e imprimir en terminal.
69 mov ebx, num1
70 mov byte[ebx], al
71 add byte[ebx], 1
72 mov al, byte[ebx]
73 call putchar
74
75 ; Parte 11:
76 mov ebx, num1
77 mov al, [ebx]
78 mov ebx, num2
79 add [ebx], al
80 mov al, byte[ebx]
81 call putchar
82
83 mov ebx, nombre
84 ;
85 capturar:
86 call getche
87 mov byte [ebx], al
88 inc ebx
89
90 cmp al, 10
91 jne capturar
92 mov byte [ebx], 0
93
94 mov edx, msg2
95 call puts
96
97 mov edx, nombre
98 call puts

```

```

@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $ ./bin/main
Ingrese un digito (0-9)

3
44
Hola

@joeOsorio → /workspaces/36285_OrganizacionYArquitecturaDeComputa
doras/2_Laboratorio/L02_Introduccion (main) $

```

Muestra en terminal el contenido de msg1.

Se realizan 2 saltos.

Se captura y muestra el carácter.

Muestra el contenido 'al' con el 1 sumado que da 4 en ASCII

Muestra en terminal el contenido de msg1 y muestra lo capturado en nombre.

Obligatoriamente de cada paso debe tener las capturas del código y de la terminal.

Las preguntas van en relación con terminal, solo eso.

## Conclusiones

Esta práctica de laboratorio permitió integrar los fundamentos del lenguaje ensamblador, destacando la manipulación directa de recursos de hardware como los registros y la memoria. A través de la declaración de variables en las secciones .data y .bss, y el uso de subrutinas (call puts, call getche, call putchar), se dominaron las operaciones básicas de entrada/salida. Un aprendizaje clave fue la relación entre la manipulación numérica de datos y su representación como caracteres mediante el código ASCII, como se evidenció al sumar 1 a un

valor capturado. En resumen, se logró el objetivo de distinguir las características de la organización y arquitectura del microprocesador, sentando bases sólidas para el desarrollo de programas eficientes a bajo nivel.

## Dificultades

Tuve dificultades para definir una organización adecuada de carpetas para el repositorio, así como para encapsular y estructurar de mejor manera las bibliotecas proporcionadas por el docente.

Otra dificultad consistió en adaptar el ensamblaje y el enlazado de los archivos .asm a la estructura de carpetas propuesta, lo que derivó finalmente en el uso de los siguientes comandos.

```
$ nasm -f elf32 -Isrc/LIB src/main.asm -o out/main.o  
$ ld -m elf_i386 -s -o bin/main out/main.o src/LIB/libpc_io.a  
$ ./bin/main
```