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

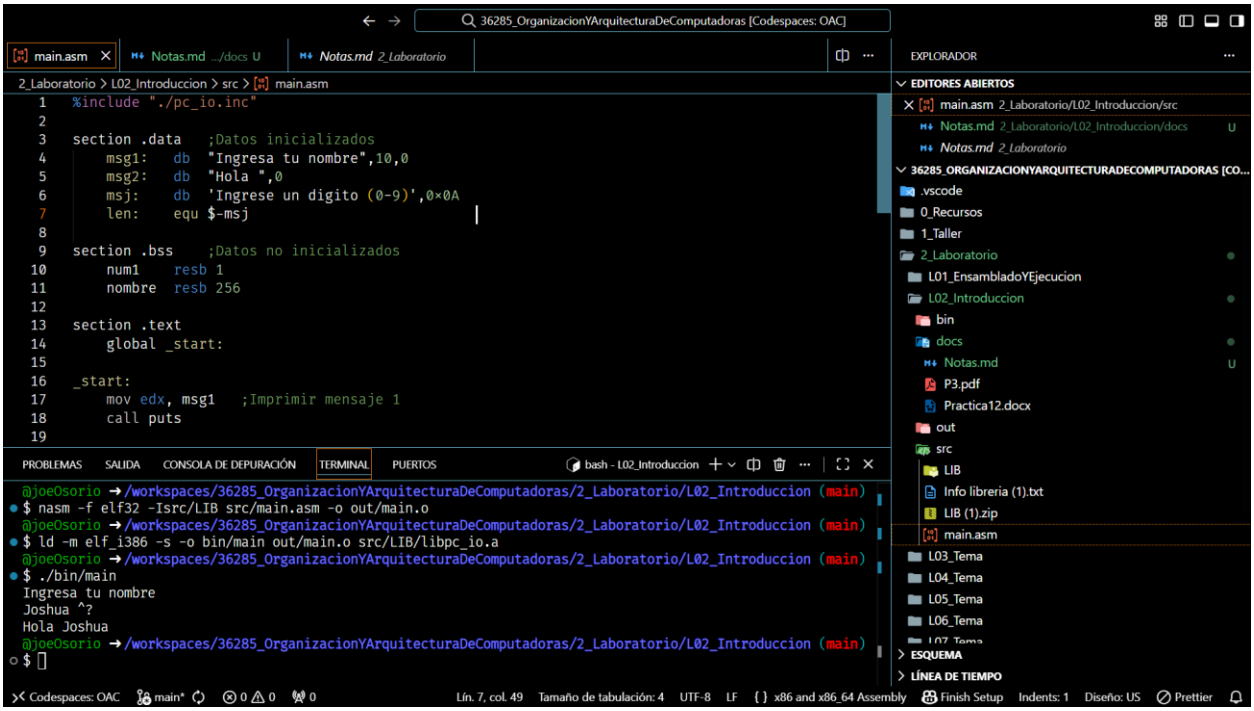
Distinguir las características de la organización y arquitectura del microprocesador de una computadora de propósito general, analizando sus recursos de hardware y software, para conocer capacidades y limitaciones de forma organizada y responsable.

Descripción de la practica

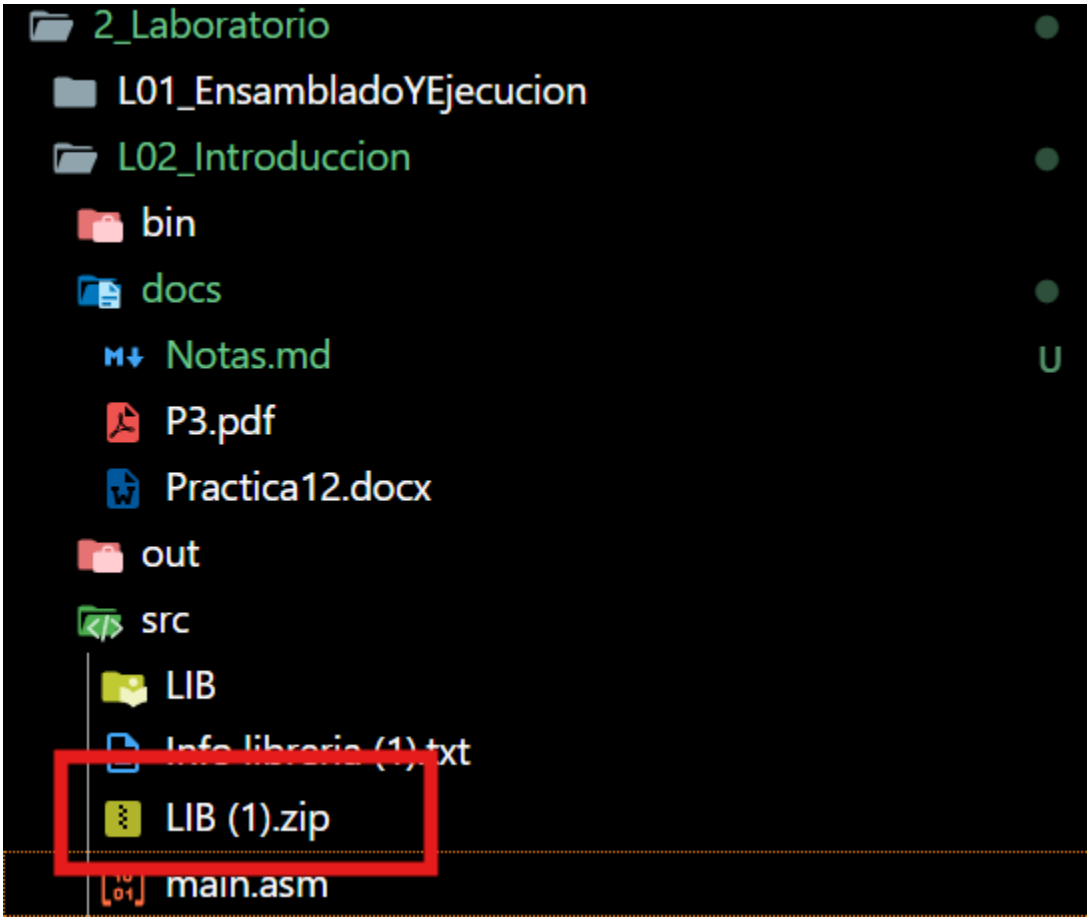
Lista de materiales

- Computadora
- Conexión a internet
- Giit
- Vscode

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

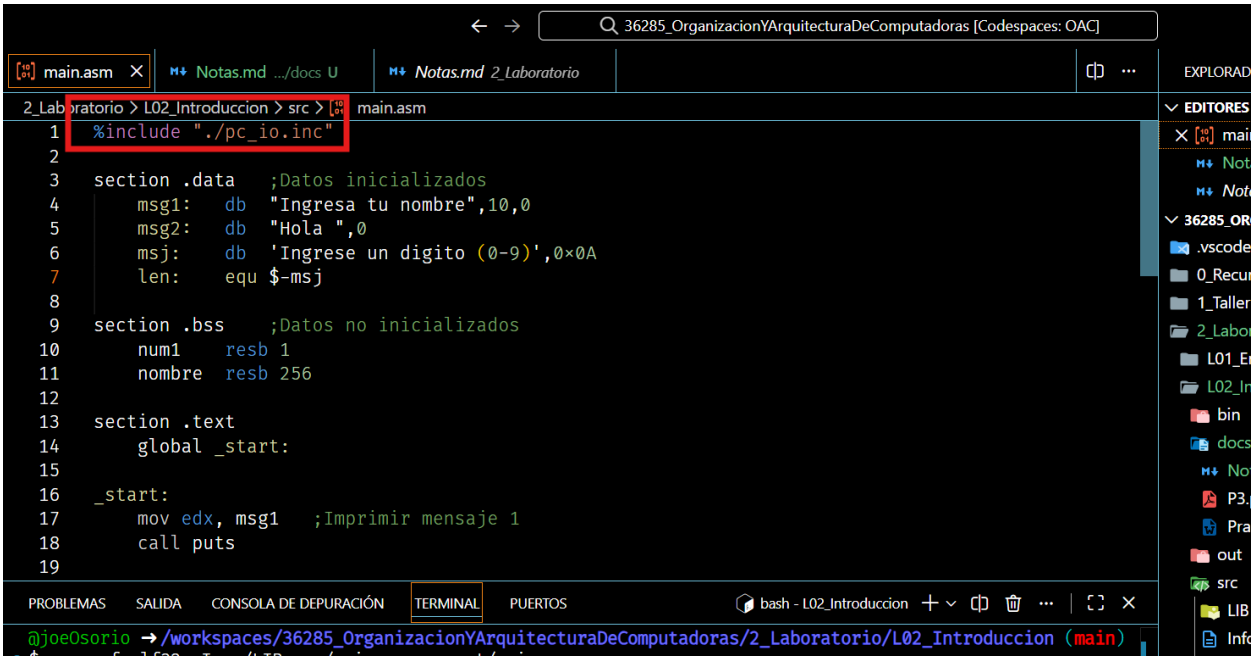


Se descargó el archivo lib.zip desde Moodle, se descomprimió y se colocaron los dos archivos contenidos en la carpeta de trabajo de GitHub.



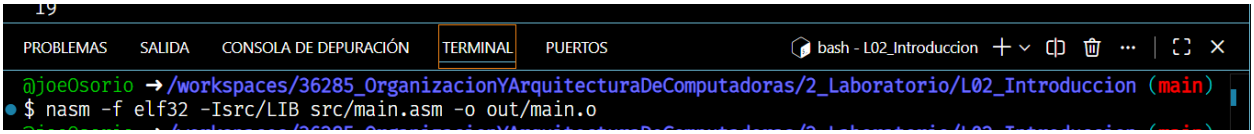
En la parte superior del archivo nombre_Practica.asm se agregó la instrucción:

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



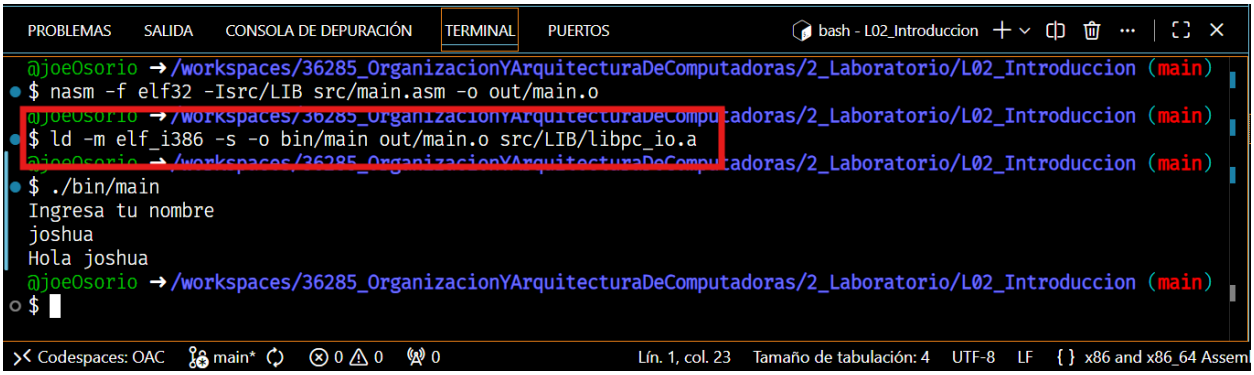
El código se ensambló con el comando:

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



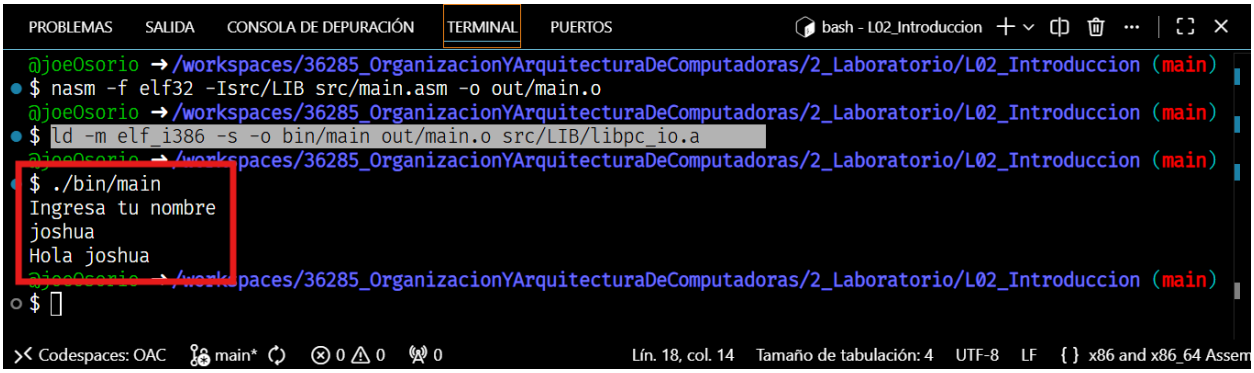
Se enlazó el objeto generado con el comando:

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



Finalmente, el programa se ejecutó con:

```
./nombre_Practica
```



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 digito (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:

Codigo1:

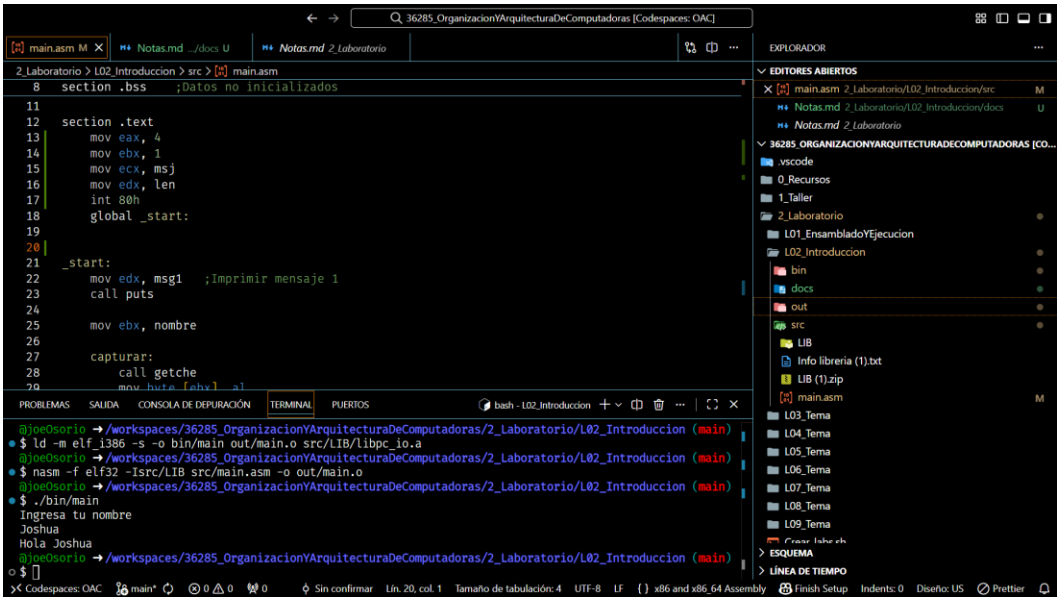
```
section .text
    mov eax, 4
    mov ebx, 1
    mov ecx, msj
    mov edx, len
    int 80h
    global _start:
```

¿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 direccion de la variable msj
Carga el registro edx con la direccion de la vriable len.
Finaliza programa.

¿Qué hizo realmente este código?

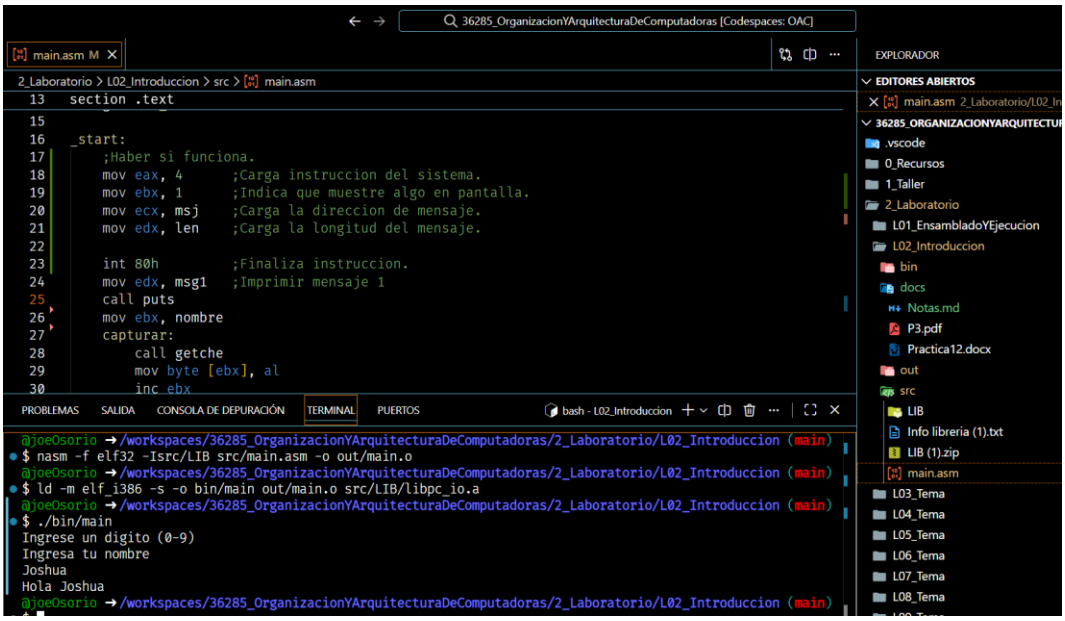
Coloque el codigo despues del .text pero no cambio nada.



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.
```



Codigo2:

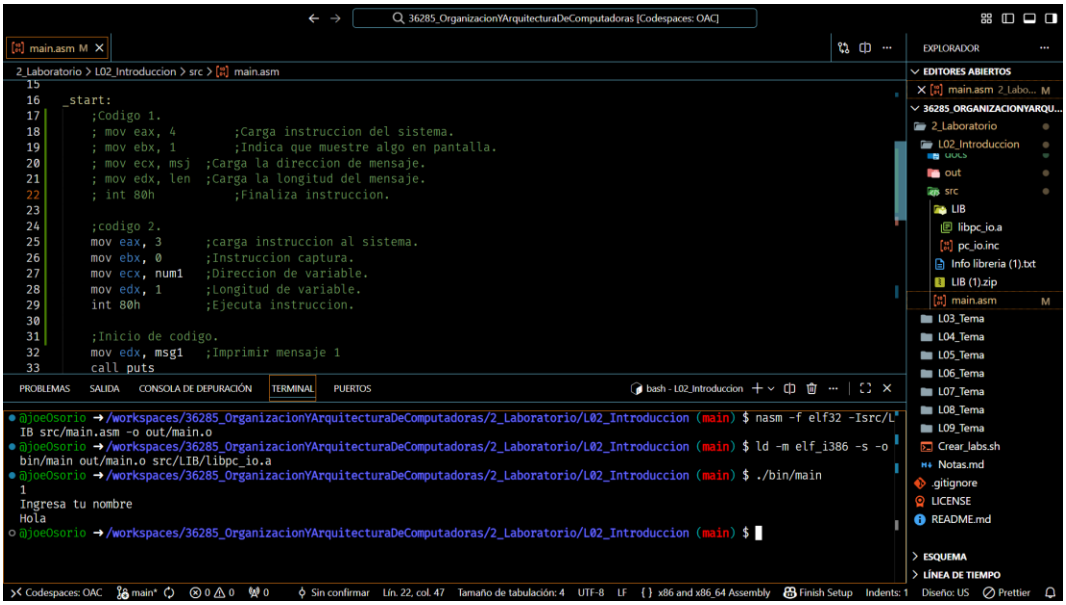
```
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 1
    int 80h
```

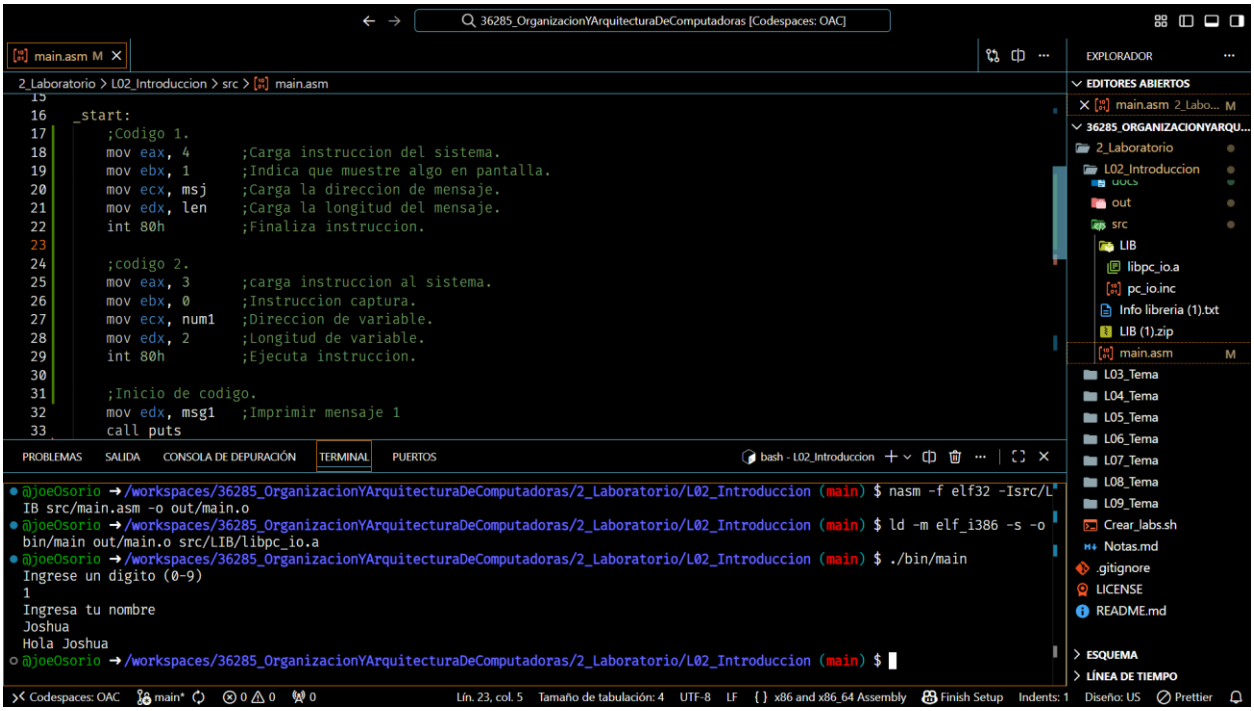
¿Qué piensa usted que hará este código?

```
    mov eax, 3          -> Carga una instrucción alsistema.
    mov ebx, 0          -> Carga la instrucción de captura.
    mov ecx, num1       -> Carga la dirección donde se almacenara.
    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.
```





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

Código 3:

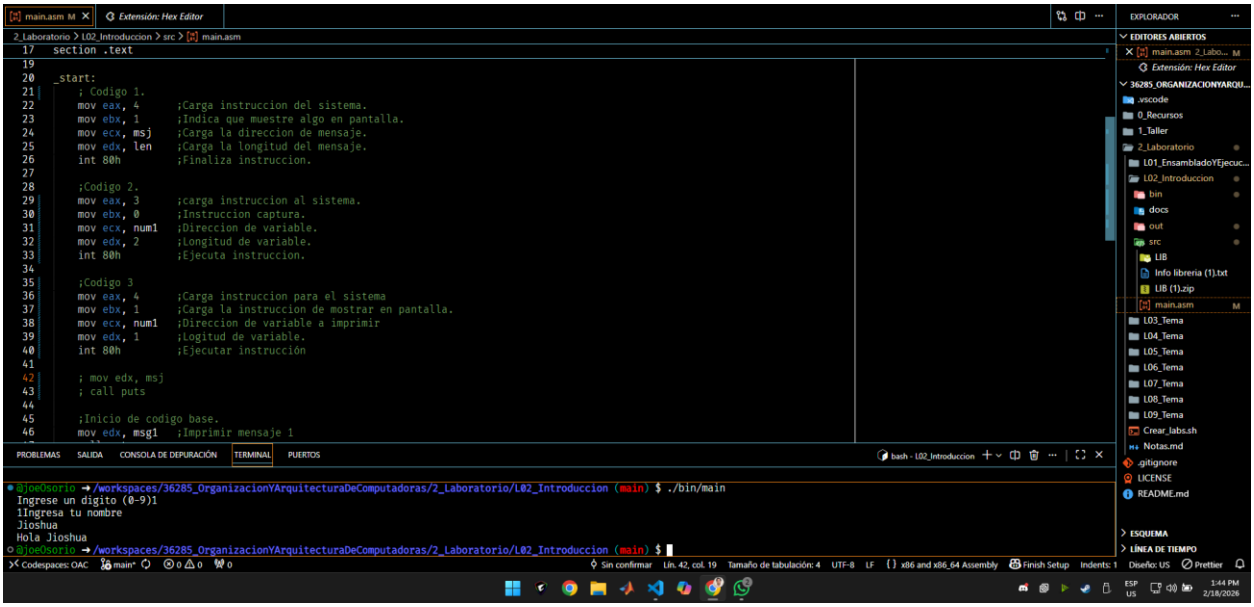
```
    mov eax, 4
    mov ebx, 1
    mov ecx, num1
    mov edx, 1
    int 80h
```

¿Qué piensa usted que hará este código?

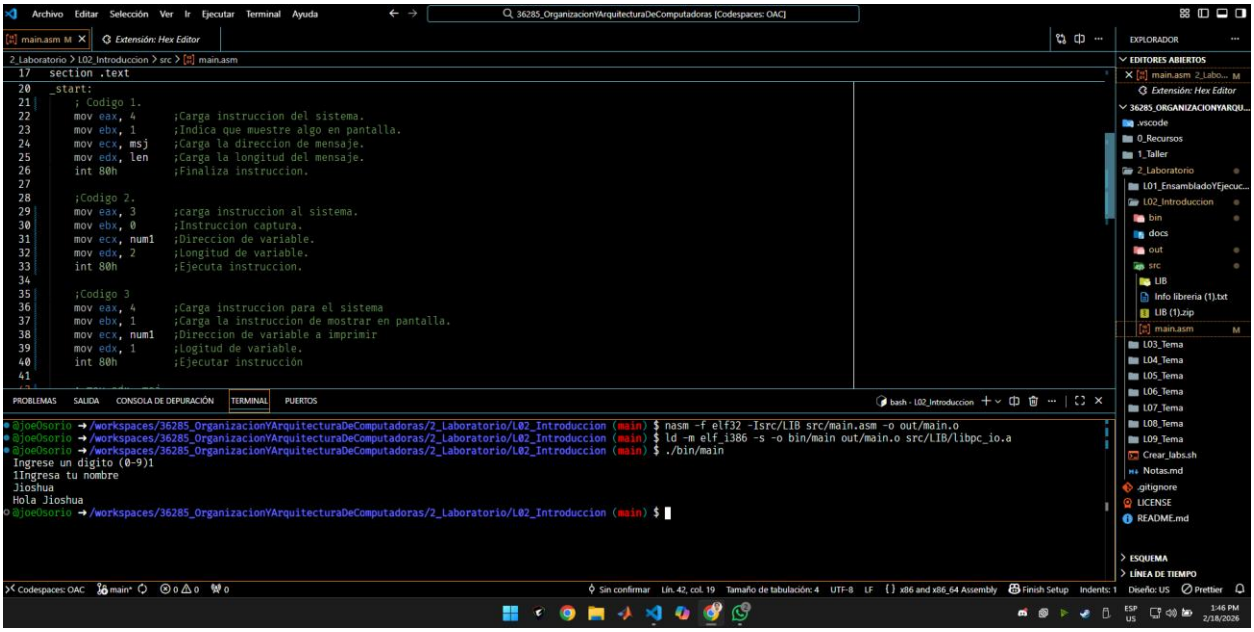
```
    mov eax, 4      ;Carga instruccion para el sistema
    mov ebx, 1      ;Carga la instruccion de mostrar en pantalla.
    mov ecx, num1   ;Direccion de variable a imprimir
    mov edx, 1      ;Logitud de variable.
    int 80h         ;Ejecutar instrucción
```

Mostrar en pantalla lo capturado en la variable num1.

¿Qué hizo realmente este código?



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



4. 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 msj y da un salto de línea.
   Codigo 2: Captura y guardar valor en variable 1
   Codigo 3: Muestra el contenido de la variable num1

5. Ahora el código anterior lo vamos a modificar:
   a. El msj ahora debe quedar de la siguiente manera:

```
msj:    db  'Ingrese un digito (0-9)',0x0
```

   b. 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

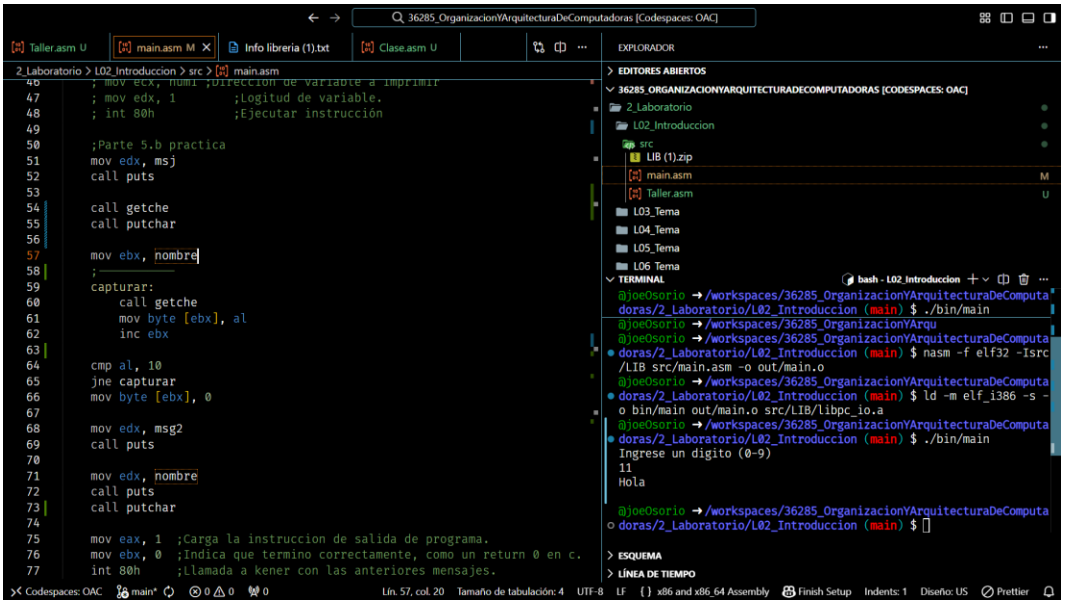
En mi caso está guardando un 0.

¿Qué piensa usted que hará este código?

```
        call putchar
```

¿Qué hizo realmente este código?

Muestra el contenido capturado por el teclado



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
```

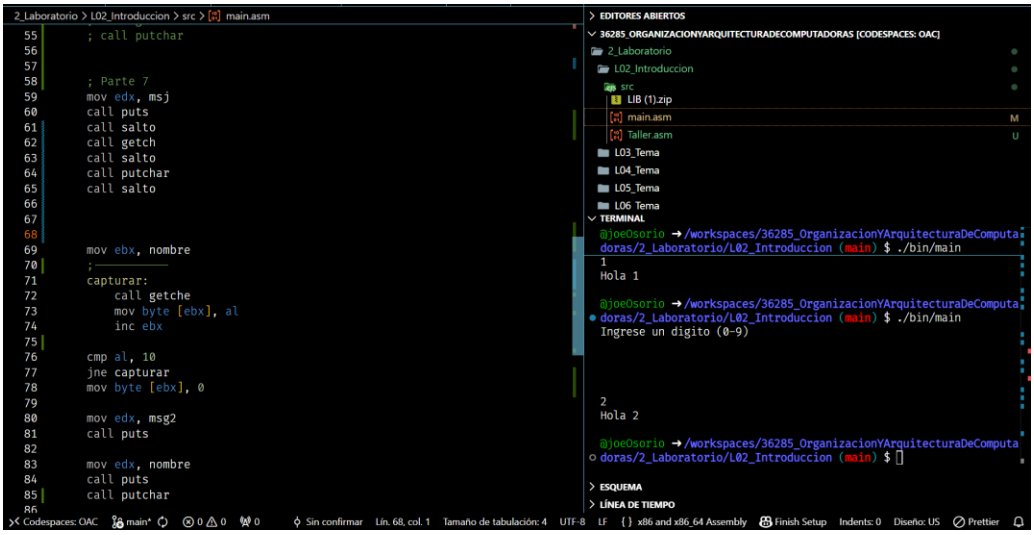
7. Ahora el código se modifica con salto y debe quedar de la siguiente manera:

¿Qué piensa usted que hará todo este código?

```
mov edx, msj
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.



8. Vamos y modificamos la función de salto:

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

Arega las banderas pusad 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

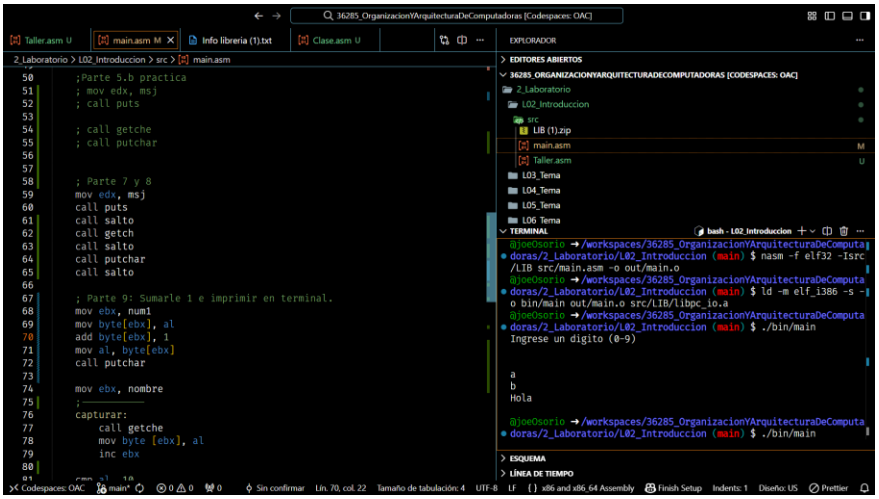
¿Qué piensa usted que hará todo este código?

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

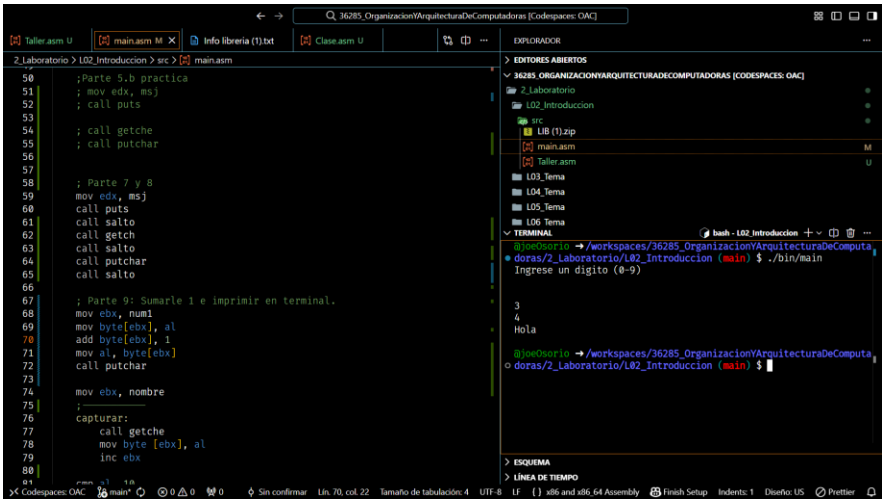
Copia la dirección de num1 a ebx.

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’



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



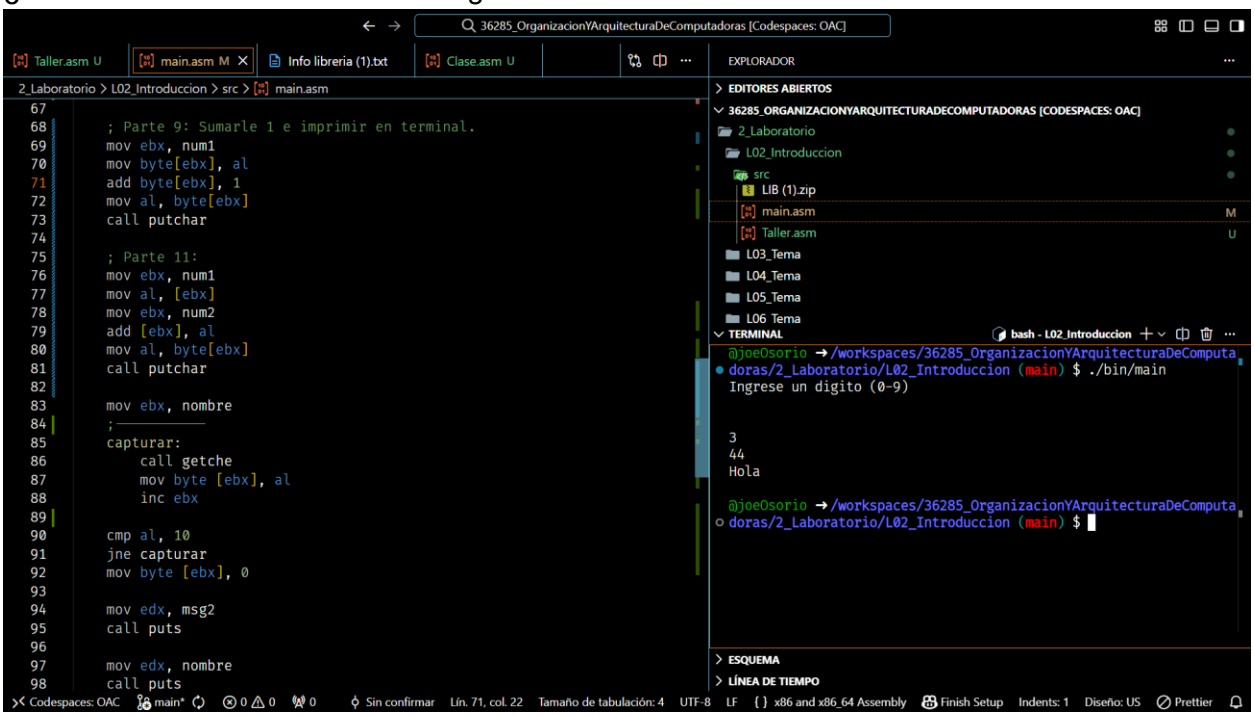
11. Ahora capture otro número, modifique lo necesario para ello, y pruebe con una suma de 3 + 4 = 7:

¿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?



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
```