UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ciencias Químicas e Ingeniería

Práctica 1

Materia:
Organización y Arquitectura de Computadoras

Docente:
Garcia Rocha Jose Isabel

Elaborado por:
Liang Liu Kevin

Tijuana, B.C. a 11 de febrero de 2026

Objetivo
Conocer y dominar el uso de una máquina virtual con sistema operativo Linux analizando sus recursos de hardware y software, para conocer sus capacidades y limitaciones de forma organizada y responsable.

Desarrollo

.data
Es la sección donde se almacenan datos inicializados, variables que ya tienen un valor asignado al iniciar el programa.

Sirve para guardar variables globales, constantes, cadenas de texto y arreglos con valores iniciales. Estos datos quedan almacenados en el ejecutable.

Se declara la sección y luego las variables:

```
choice      DB  'y'
number      DW  12345
neg_number  DW  -12345
big_number  DQ  123456789
real_number1    DD  1.234
real_number2    DQ  123.456
```

Directive	Purpose	Storage Space
DB	Define Byte	allocates 1 byte
DW	Define Word	allocates 2 bytes
DD	Define Doubleword	allocates 4 bytes
DQ	Define Quadword	allocates 8 bytes
DT	Define Ten Bytes	allocates 10 bytes

.bss
Es la sección donde se reservan variables sin inicializar. El programa solo reserva espacio en memoria, pero no guarda valores en el ejecutable.

Sirve para variables que cambiarán durante la ejecución, no necesitan valor inicial, sólo requieren espacio reservado.

Se declara la sección y se indica el tamaño a reservar:

```
section .bss

    num resb 3      ; Reserves three bytes of memory for 'num'
```

Directive	Purpose
RESB	Reserve a Byte
RESW	Reserve a Word
RESD	Reserve a Doubleword
RESQ	Reserve a Quadword
REST	Reserve a Ten Bytes

.text
Se utiliza para mantener el código real. Esta sección debe comenzar con la declaración global _start, que le indica al kernel dónde comienza la ejecución del programa.

Sirve para almacenar instrucciones y funciones. El procesador ejecuta código desde esta sección.

La sintaxis para declarar la sección de texto es

```
section.text
   global _start
_start:
```

Directiva global
La directiva global permite exportar símbolos (como funciones o

variables) para que puedan ser utilizados en otros módulos del programa. Esto es esencial para la programación modular, donde se dividen distintas partes del código en archivos separados.
Se utiliza en conjunto con la directiva *extern* en otros archivos para declarar la función o variable.

Si tienes un archivo de ensamblador que define una función y deseas utilizarla en otro archivo, haces lo siguiente:

```
global mi_funcion  ; Hacer visible la función externamente

section .text
mi_funcion:
    mov eax, 1      ; Cargar valor en EAX
    ret             ; Retornar
```

```
extern mi_funcion  ; Declarar la función definida en otro archivo

section .text
global _start

_start:
    call mi_funcion ; Llamada a la función externa
    mov eax, 60     ; syscall exit
    xor edi, edi
    syscall
```

Para ensamblar y enlazar estos archivos, puedes usar los siguientes comandos en NASM:

```
nasm -f elf64 funciones.asm -o funciones.o
nasm -f elf64 main.asm -o main.o
ld main.o funciones.o -o programa
./programa
```

Comandos más populares en Linux
- ls
Muestra el contenido de un directorio.

- cd

Permite moverse entre carpetas.
cd carpeta
cd ..
cd /
cd ~

- pwd
Muestra la ruta del directorio actual.

- mkdir
Crea una carpeta.
mkdir proyecto

- rm
Borra archivos o directorios.
rm archivo.txt
rm -r carpeta

- cp
Copia archivos o carpetas.
cp archivo carpeta/archivo
cp -r carpeta destino

- mv
Mueve o cambia el nombre de archivos.
mv archivo carpeta/archivo
mv viejo.txt nuevo.txt

- touch
Crea un archivo vacío.
touch archivo.txt

- cat
Muestra el contenido de un archivo.
cat archivo.txt

- nano

Permiten editar archivos desde la terminal. CtrlX + Y + Enter para guardar.
nano archivo.txt

- clear
Limpia la terminal.

- man
Muestra ayuda de un comando.
man ls

- nasm y ld
Permiten compilar programas en ensamblador.
nasm -f elf32 programa.asm
ld -m elf_i386 programa.o -o programa
./programa

Extensiones de archivos relacionadas con ASM
.asm
Archivo fuente que contiene código en lenguaje ensamblador.

.s
Archivo ensamblador utilizado comúnmente en sistemas Unix/Linux.

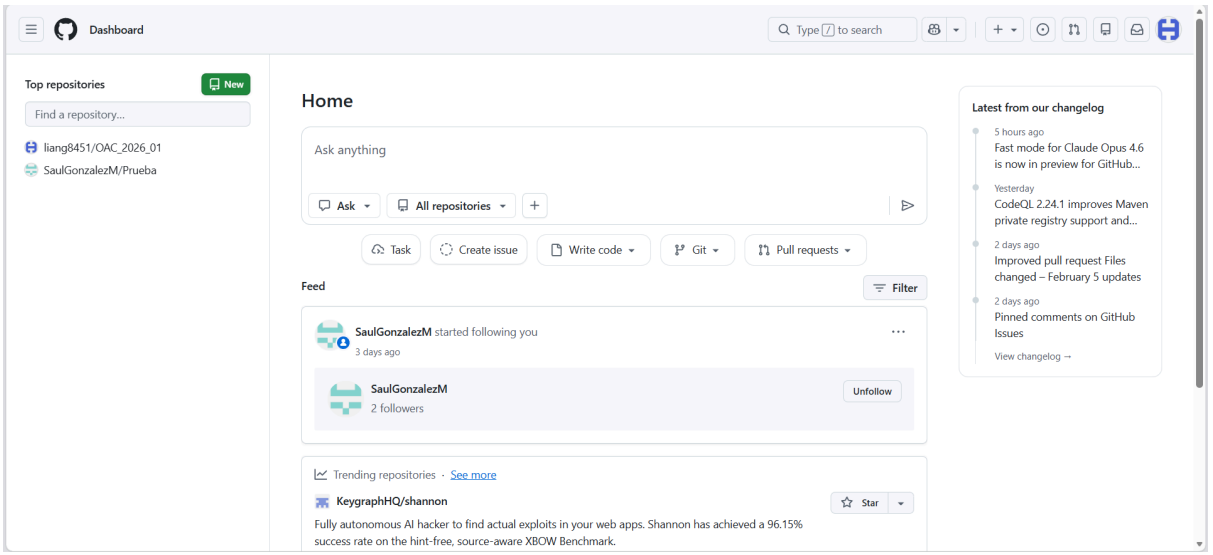
.S
Archivo ensamblador que pasa primero por el preprocesador de C.

.o
Archivo objeto generado tras ensamblar el código.
Ejecutable sin extensión
Archivo final ejecutable en Linux tras enlazar el programa.

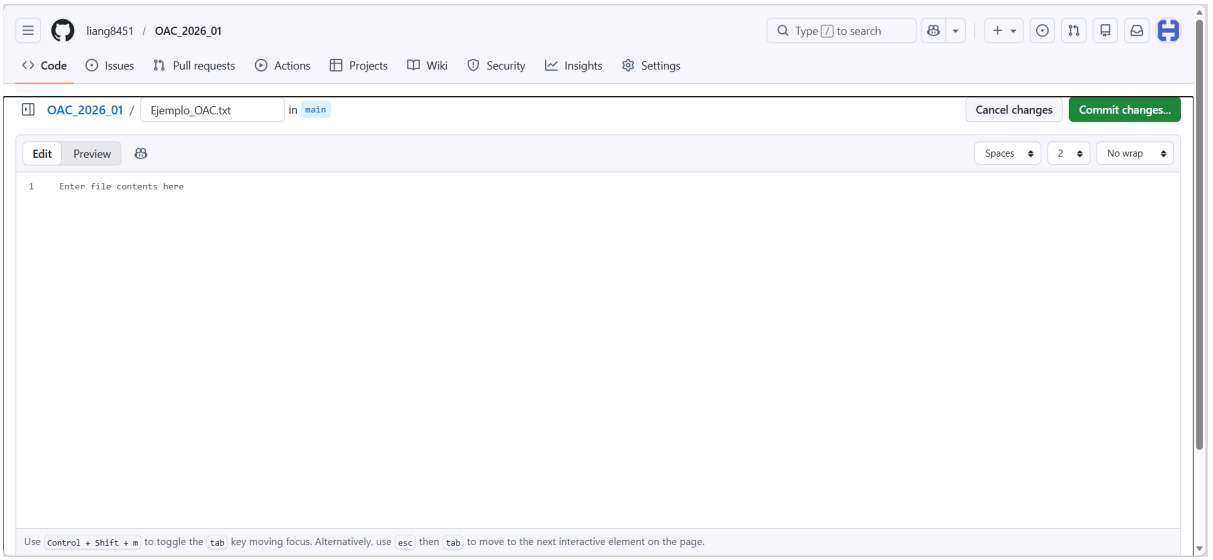
Flujo típico de compilación ASM
programa.asm → (nasm) → programa.o → (ld) → programa ejecutable → ejecución

Hello world en ensamblador

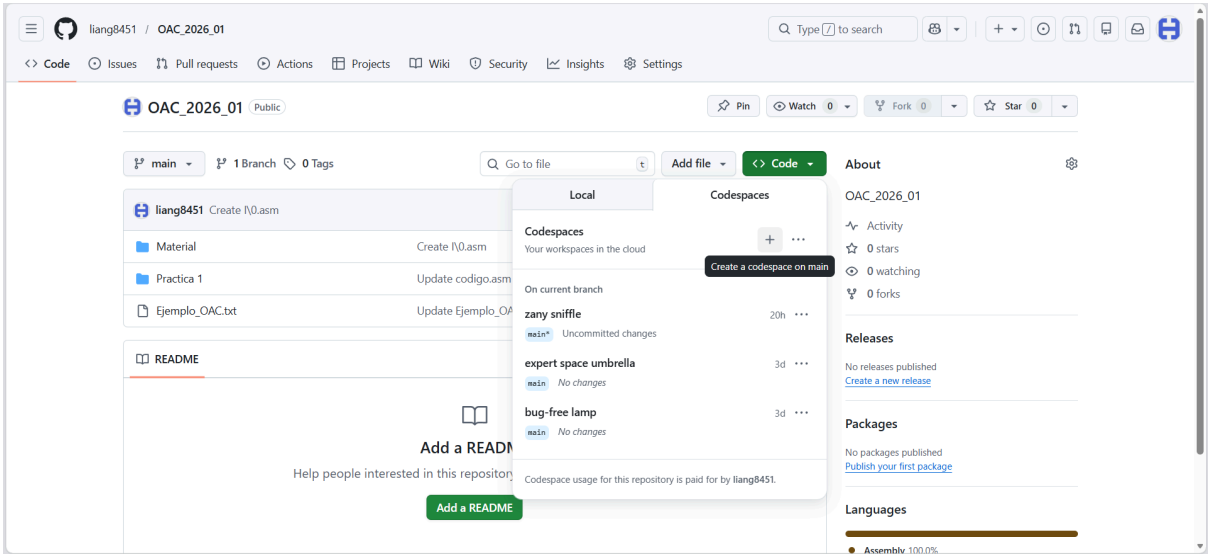
1. Entrar a GitHub



2. Crear un repositorio nuevo con un archivo con nombre “Ejemplo_OAC.txt”.



3. Para activar codespaces se requiere dar clic en <> Code luego en Codespaces



4. Instalar nasm, con la siguiente instrucción:

sudo apt update

sudo apt install nasm -y

```
@liang8451 ➜ /workspaces/OAC_2026_01 (main) $ sudo apt update
Get:1 https://dl.yarnpkg.com/debian stable InRelease
Get:2 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble InRelease [3600 B]
Get:3 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable InRelease [3961 B]
Err:1 https://dl.yarnpkg.com/debian stable InRelease
  The following signatures couldn't be verified because the public key is not available: NO_PUBKEY 62D54FD4003F6525
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble/main all Packages [650 B]
Get:6 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble/main amd64 Packages [89.3 kB]
Get:7 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable/main amd64 Packages [4557 B]
Get:8 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [34.8 kB]
Get:10 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [3061 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [1199 kB]
Get:12 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1802 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:16 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:17 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:18 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:19 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1978 kB]
Get:20 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [3280 kB]
Get:21 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [2198 kB]
Get:22 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [38.1 kB]
Get:23 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [34.6 kB]
Get:24 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Packages [49.5 kB]
Reading package lists... Done
W: GPG error: https://dl.yarnpkg.com/debian stable InRelease: The following signatures couldn't be verified because t
he public key is not available: NO_PUBKEY 62D54FD4003F6525
E: The repository 'https://dl.yarnpkg.com/debian stable InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
@liang8451 ➜ /workspaces/OAC_2026_01 (main) $
```

```
@liang8451 ➜/workspaces/OAC_2026_01 (main) $ sudo apt install nasm -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  nasm
0 upgraded, 1 newly installed, 0 to remove and 88 not upgraded.
Need to get 459 kB of archives.
After this operation, 3407 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble/universe amd64 nasm amd64 2.16.01-1build1 [459 kB]
Fetched 459 kB in 1s (425 kB/s)
Selecting previously unselected package nasm.
(Reading database ... 58629 files and directories currently installed.)
Preparing to unpack .../nasm_2.16.01-1build1_amd64.deb ...
Unpacking nasm (2.16.01-1build1) ...
Setting up nasm (2.16.01-1build1) ...
Processing triggers for man-db (2.12.0-4build2) ...
@liang8451 ➜/workspaces/OAC_2026_01 (main) $
```

5. Revisar la versión de nasm instalado:
nasm -v

```
@liang8451 ➜/workspaces/OAC_2026_01 (main) $ nasm -v
NASM version 2.16.01
```

6. Crear una carpeta cuyo nombre sea “ejemplo”:
mkdir ejemplo

7. Cambiar del directorio /workspaces para que la carpeta nueva sea este, con:
cd /workspaces/”sustituir_con_nombre_repositorio”/ejemplo/

8. Comprobar que este cambio se haya realizado con:
pwd

```
@liang8451 ➜/workspaces/OAC_2026_01 (main) $ mkdir ejemplo
@liang8451 ➜/workspaces/OAC_2026_01 (main) $ cd /workspaces/OAC_2026_01/ejemplo/
@liang8451 ➜/workspaces/OAC_2026_01/ejemplo (main) $ pwd
/workspaces/OAC_2026_01/ejemplo
```

9. Crear un archivo con extensión .asm y con nombre “ejemplo”:
touch ejemplo.asm

```
@liang8451 ➜/workspaces/OAC_2026_01/ejemplo (main) $ touch ejemplo.asm
```

10. Abrir Notepad para trabajar en ese editor de texto sin ayudas.

11. Ingresar el siguiente código en el archivo:

```
global _start
section .text
_start:

    ; sys_write(stdout, message, length)
    mov eax, 4
    mov ebx, 1
    mov ecx, message
    mov edx, length
    int 80h

    ; sys_exit(return_code)
    mov eax, 1          ; sys_exit syscall
    mov ebx, 0          ; return 0 (todo correcto)
    int 80h

section .data

    message: db 'Hello, world!',0x0A    ; mensaje y nueva linea
    length: equ $-message               ; obtenemos la longitud de la cadena

```

12. Cargar el código escrito en Notepad sin errores en el archivo ejemplo.asm en GitHub.

13. Ensamblar el archivo creado anteriormente:
nasm -f elf ejemplo.asm

14. Enlazar el objeto creado:
ld -m elf_i386 -s -o ejemplo ejemplo.o

15. Ejecutar el programa obtenido:
./ejemplo

```
@liang8451 →/workspaces/OAC_2026_01/ejemplo (main) $ nasm -f elf ejemplo.asm
@liang8451 →/workspaces/OAC_2026_01/ejemplo (main) $ ld -m elf_i386 -s -o ejemplo ejemplo.o
@liang8451 →/workspaces/OAC_2026_01/ejemplo (main) $ ./ejemplo
Hello, world!
@liang8451 →/workspaces/OAC_2026_01/ejemplo (main) $
```

Conclusiones

Esta práctica ha permitido una comprensión sólida sobre el entorno de trabajo con máquinas virtuales y el sistema operativo Linux, así como una introducción práctica a la programación en lenguaje ensamblador

(NASM). Sirvió como un paso inicial para entender los cimientos del software, desde la interacción con el sistema operativo a través de la terminal hasta la programación directamente ligada a la arquitectura del procesador.

Dificultades

Se tuvo dificultad en seguir los pasos para la creación del programa ‘Hello world' por no estar familiarizado con el entorno y conceptos de GitHub. Y no había manera de saber si estaba siguiendo los pasos correctamente. Pero gracias a la guía del maestro se pudo concretar el programa.

Referencias

- Albornoz, D. (2026, 15 enero). *Los 40 comandos de Linux más utilizados en 2026 con ejemplos*. Hostinger Tutoriales. https://www.hostinger.com/es/tutoriales/linux-comandos

- *Assembly - Text section*. (2024, 30 agosto). Datacadamia - Data And Co. https://www.datacadamia.com/lang/assembly/code

- *Uninitialized data*. (s. f.). Registry. https://jotavare.github.io/docs/x86_assembly_nasm/data_and_memory/uninitialized_data.html

- *Assembly - Basic Syntax*. (s. f.). https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm

- García, J. (2026, 7 febrero). Cómo usar la directiva global en ensamblador. *Localhorse.net*. https://localhorse.net/article/como-usar-la-directiva-global-en-ensamblador