Introducción

Un conjunto de instrucciones es una serie ordenada de comandos que una computadora o programa debe ejecutar para realizar una tarea específica. Cada instrucción indica una acción concreta como sumar dos números, mostrar un mensaje o almacenar un dato y se ejecuta de manera secuencial o condicional según la lógica del programa. En pocas palabras es el “paso a paso” que sigue el computador para resolver un problema o realizar una operación, escrito en un lenguaje que la máquina puede entender

Para el emulador elegí OneCompiler que es un emulador y entorno de desarrollo en línea que permite escribir, ejecutar y probar código directamente desde el navegador, sin necesidad de instalar software adicional y me ayudará a:

- Probar programas rápidamente en Assembly.
- Aprender a programar de forma práctica, visualizando resultados inmediatos.
- Compartir código fácilmente, ya que genera enlaces públicos para los proyectos.
- Simular la ejecución de un conjunto de instrucciones, observando cómo se comporta paso a paso.

Defina dos números en memoria.

OneCompiler

HelloWorld.asm 443gbk8ut

```
1  section .data
2      num1 db 1        ; Primer número (byte con valor 1)
3      num2 db 11       ; Segundo número (byte con valor 11)
4      result db 0      ; guardar resultado
5
6  section .text
7      global _start
8
9  _start:
10
11     mov eax, 1       ; Código de salida (sys_exit)
12     mov ebx, 0       ; Salida sin error
13     int 80h          ; Llamada al sistema para salir
14
```

Realice una suma y almacene el resultado en otra dirección de memoria.

HelloWorld.asm — Laboratorio 1 Lenguajes de Programacion

```
1  section .data
2      num1 db 1        ; Primer número (byte con valor 1)
3      num2 db 11       ; Segundo número (byte con valor 11)
4      result db 0      ; guardar resultado
5
6  section .text
7      global _start
8
9  _start:
10     ; Cargar los valores de num1 y num2
11     mov al, [num1]       ; Cargar el valor de num1 en AL (parte baja de EAX)
12     add al, [num2]       ; Sumar el valor de num2 al contenido de AL
13
14     ; result = al
15     mov [result], al     ; Almacenar el resultado en memoria
16
17     ; Terminar el programa
18     mov eax, 1       ; Código de salida (sys_exit)
19     mov ebx, 0       ; Salida sin error
20     int 80h          ; Llamada al sistema para salir
```

Línea	Acción	Explicación
num1 db 1	Define número	Guarda 1 en memoria
num2 db 11	Define número	Guarda 11 en memoria
result db 0	Espacio vacío	Donde guardaremos el resultado
mov al, [num1]	Carga valor	CPU toma el 1 de num1
add al, [num2]	Suma	Suma 11 al valor que tenía (1 + 11)
mov [result], al	Guarda resultado	Pone 12 en result
mov eax,1 / int 80h	Termina programa	Finaliza correctamente

Ejecute un bucle simple que reste 1 a un contador hasta llegar a 0.

```
section .data
    num db 5              ; contador inicial = 5
    comparition db 0      ; valor de comparación = 0

section .text
    global _start

_start:
    mov al, [num]         ; cargar el valor del contador (5)

loop_start:
    dec al                ; restar 1 al contador osea al=-1
    cmp al, [comparition] ; comparar con 0 al==0
    jne loop_start        ; si no es 0, repetir el bucle

    ; cuando el contador llega a 0, termina
    mov eax, 1            ; syscall: exit
    mov ebx, 0
    int 80h
```

Instrucción	Acción	AL
mov al, [num]	Carga el valor 5 de memoria en AL	5
dec al	Resta 1 al valor de AL	4
cmp al, [comparition]	Compara AL (4) con 0	4
dec al	Resta 1	3
cmp al, [comparition]	Compara AL (3) con 0	3
dec al	Resta 1	2
cmp al, [comparition]	Compara AL (2) con 0	2
dec al	Resta 1	1
cmp al, [comparition]	Compara AL (1) con 0	1

dec al	Resta 1	0
cmp al, [comparition]	Compara AL (0) con 0	0
mov eax, 1	sys_exit	—
int 80h	Termina la ejecución	—

Análisis y reflexión

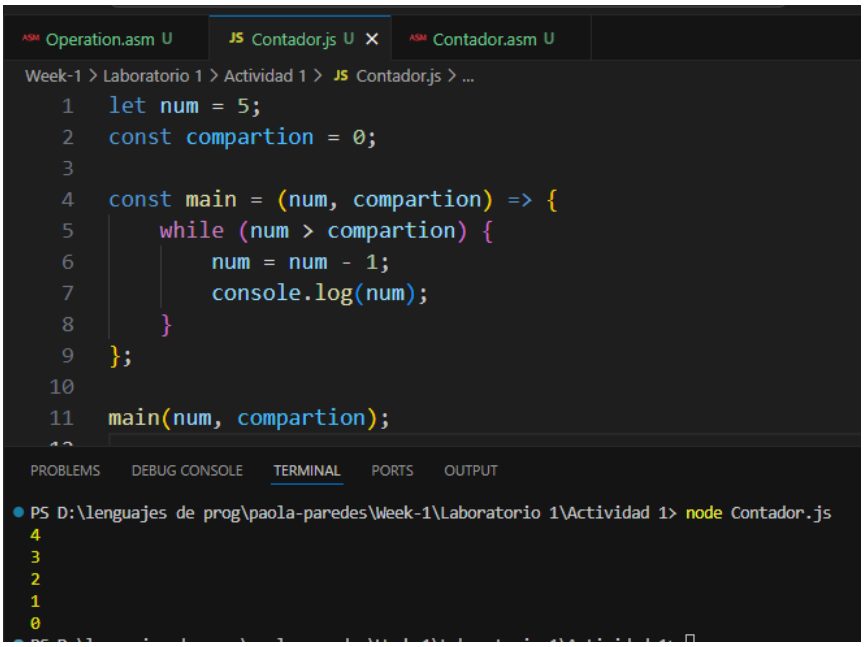
Aspecto	Lenguaje de Alto Nivel (JavaScript)	Lenguaje Ensamblador (Assembly)
Nivel de abstracción	Alto — se parece al lenguaje humano, fácil de leer.	Muy bajo — cercano al hardware, depende del procesador.
Complejidad	Una línea puede realizar operaciones complejas.	Cada paso debe escribirse explícitamente.
Portabilidad	Funciona en cualquier sistema con un intérprete o navegador.	Depende de la arquitectura (x86, ARM, etc.).
Tiempo de desarrollo	Rápido; código más corto y fácil de mantener.	Lento; requiere conocimientos técnicos de arquitectura.

JALA UNIVERSITY

LABORATORIO 1

PAOLA PAREDES

Fecha de entrega
2/11/25
Clase:
Lenguajes de programación



Taller de Ensamblador Básico

Operaciones Aritméticas

```
section .data
    num1 db 2       ; Primer número (byte con valor 2)
    num2 db 11      ; Segundo número (byte con valor 11)
    resultsum db 0     ; guardar resultado de Suma
    resultsus db 0     ; guardar resultado de Resta
    resultmult db 0     ; guardar resultado de Multiplicacion

section .text
    global _start

_start:
    ; Suma: num1 + num2
    mov al, [num1]          ; AL = num1 (2)
    add al, [num2]          ; AL = AL + num2 → 2 + 11 = 13
    mov [resultsum], al     ; Guardar resultado (13)
```


```
17      ; Resta: num2 - num1
18      mov al, [num2]          ; AL = num2 (11)
19      sub al, [num1]          ; AL = AL - num1 → 11 - 2 = 9
20      mov [resultsus], al     ; Guardar resultado (9)
21
22      ; Multiplicación: num1 * num2
23      mov al, [num1]          ; AL = num1 (2)
24      mov bl, [num2]          ; BL = num2 (11)
25      mul bl                  ; AX = AL * BL → 2 * 11 = 22
26      mov [resultmult], al    ; Guardar solo la parte baja (22)
27
28      ; Salida limpia del programa
29      mov eax, 1              ; sys_exit
30      mov ebx, 0
31      int 80h
```

.

Tomando en cuenta que los valores en el simulador de https://schweigi.github.io/assembler-simulator/ son hexa decimales se cumple lo esperado

Labels

Name	Address	Value
num1	02	02
num2	03	0B
resultmult	06	16
resultsum	04	0D
resultsus	05	09
start	07	02

Control de Flujo con Bucle

Para este archivo puedes regresar al punto [Bucle simple]