LENGUAJE DE BAJO NIVEL
OTOÑO 2025
C12025535
NCR 21298

14 DE SEPTIEMBRE DE 2025

Proyecto Unidad 1: Simulador de CPU

ALAN JESÚS TRUJILLO CAMARILLO 202462139
EDUARDO LÓPEZ ATANASIO 202431623
JIMENA TORRES PÉREZ 202461861
ALEXANDER LÓPEZ HERNÁNDEZ 202431690
ANGEL YAEL TELLEZ ORTIZ 202461096

INDICE

I. Introducción

Este documento presenta el diseño, implementación y validación de un **simulador educativo de CPU** capaz de ejecutar programas escritos en un **lenguaje ensamblador simplificado**, soportando **seis modos de direccionamiento** clásicos en arquitecturas de propósito general. El objetivo principal es **visualizar el ciclo de instrucción** (fetch-decode-execute) y **comprender cómo los modos de direccionamiento afectan la forma en que se accede a los operandos**, sin necesidad de hardware físico.

II. Marco Teórico

1.1 Unidad Central de Procesamiento (CPU)

La **Unidad Central de Procesamiento (CPU)** es el núcleo encargado de **ejecutar instrucciones y coordinar** las operaciones de **hardware y software** en un sistema de cómputo. Su función principal es **interpretar los programas y realizar cálculos lógicos y aritméticos**, así como **controlar el flujo de datos** entre los distintos componentes del sistema.

Componentes principales:

- **Unidad de Control (UC)**: se encarga de **leer las instrucciones de memoria, decodificarlas y coordinar** el resto de las unidades. Supervisa el ciclo de instrucción (fetch-decode-execute).
- **Unidad Aritmético-Lógica (ALU)**: ejecuta **operaciones matemáticas** (suma, resta, multiplicación, división) y **operaciones lógicas** (AND, OR, NOT, XOR, comparaciones).
- **Registros**: pequeñas **memorias internas** de gran velocidad que **almacenan datos temporales, direcciones de memoria o resultados intermedios**.

El ciclo de instrucción consta de:

1. **Fetch**: la CPU obtiene de la memoria la instrucción que se va a ejecutar.

2. **Decode**: la instrucción es interpretada para identificar la operación y los operandos.
3. **Execute**: la operación se lleva a cabo en la ALU o en otro subsistema, modificando registros o memoria.

Este ciclo ocurre de manera repetitiva y extremadamente rápida, llegando a ejecutarse millones o incluso miles de millones de instrucciones por segundo en procesadores modernos.

1.2 Memoria

La memoria constituye el espacio donde se **almacenan datos e instrucciones** que la CPU utiliza en su operación. En la arquitectura de **Von Neumann**, la memoria se comparte entre instrucciones y datos, lo que simplifica el diseño, pero también limita la velocidad de procesamiento debido a que CPU y memoria comparten el mismo bus (bottleneck de Von Neumann).

Tipos principales de memoria:

- **Memoria RAM (Random Access Memory)**: almacena temporalmente datos e instrucciones durante la ejecución de programas. Es volátil, lo que significa que pierde su contenido al apagar el sistema.
- **Memoria caché**: memoria ultrarrápida integrada en la CPU que almacena datos de uso frecuente para reducir la latencia en el acceso.
- **Memoria secundaria(HDDS/SSD)**: dispositivos como discos duros o SSD, utilizados para almacenamiento permanente, pero con mayor tiempo de acceso en comparación con la RAM.

En un simulador educativo de CPU, la memoria se modela como un arreglo de posiciones direccionables, donde cada celda puede contener una instrucción o un dato.

1.3 Registros

Los registros son elementos esenciales en la CPU, diseñados para el almacenamiento temporal y ultrarrápido de información. A diferencia de la memoria principal, los registros permiten acceder y modificar datos en un solo ciclo de reloj, lo que los convierte en el recurso más veloz para la ejecución de instrucciones.

1.3.1 Tipos de registros

- **Registros de propósito general (GPR)**: empleados en operaciones aritméticas, lógicas y de almacenamiento temporal de datos.
- **Registros de pila (SP – Stack Pointer)**: apuntan a la parte superior de la pila, utilizada para almacenar direcciones de retorno en subrutinas y datos temporales.
- **Contador de programa (PC – Program Counter)**: contiene la dirección de la siguiente instrucción a ejecutar, controlando el flujo secuencial del programa.
- **Registros de segmento**: permiten acceder a distintas áreas de la memoria, especialmente en arquitecturas como x86.
- **Registros de bandera (FLAGS)**: almacenan información del estado de la CPU y de las operaciones (por ejemplo, si un resultado fue cero, si ocurrió un desbordamiento o si hay acarreo).

1.3.2 Registros mínimos en el simulador

Para fines didácticos, se utilizan registros mínimos que simplifican el diseño del simulador:

- **ACC (Acumulador)**: almacena resultados de operaciones aritméticas o lógicas.
- **PC (Program Counter)**: lleva el control de la ejecución de instrucciones.
- **R1 y R2**: registros de propósito general que sirven para almacenar datos intermedios y operandos.

1.4 Conjunto de instrucciones básicas

Un conjunto de instrucciones define las operaciones que la CPU puede ejecutar. Cada instrucción se representa mediante un mnemónico, una abreviación de la acción a realizar, acompañado de uno o más operandos.

Entre las instrucciones fundamentales implementadas en el simulador se encuentran:

- **ADD**: suma de dos operandos.
- **SUB**: resta de dos operandos.
- **JMP**: salto incondicional a una dirección de memoria.
- **JZ**: salto condicional si la bandera de cero está activa.
- **LOAD**: carga un dato desde memoria a un registro.
- **STORE**: almacena un dato desde un registro en memoria.
- **OUTPUT**: envía datos a un dispositivo de salida o consola.
- **HALT**: detiene la ejecución del programa.

Estas instrucciones permiten la construcción de programas que combinan cálculos, almacenamiento y control de flujo.

1.5 Modos de direccionamiento

Los modos de direccionamiento indican la forma en que se localizan los operandos de una instrucción. Su comprensión es fundamental para el diseño y simulación de una CPU, ya que definen la relación entre las instrucciones y los datos.

- **Inmediato**: el operando está incluido en la propia instrucción.
  Ejemplo: MOV AX, 5 (el valor 5 está dentro de la instrucción).
- **Directo**: la instrucción señala directamente la dirección de memoria donde se encuentra el operando.
  Ejemplo: MOV AX, [100] (se accede a la posición 100 de memoria).
- **Indirecto**: la dirección del operando se encuentra en un registro.
  Ejemplo: MOV AX, [BX] (BX contiene la dirección de memoria a acceder).

- **De registros**: el operando está dentro de un registro.
  Ejemplo: MOV AX, BX (el valor de BX se copia en AX).
- **Relativo**: la dirección se calcula sumando un desplazamiento al contador de programa (PC).
  Ejemplo: JMP PC+4 (salta cuatro instrucciones más adelante).
- **Indexado**: combina el valor de un registro índice con un desplazamiento, permitiendo acceso eficiente a arreglos o tablas.
  Ejemplo: MOV AX, [SI+10].

Estos modos ofrecen flexibilidad en la programación y son fundamentales para la optimización del acceso a datos en memoria y registros.

1.6 Relevancia en la simulación de CPU

El estudio de la CPU, memoria, registros, instrucciones y modos de direccionamiento en un simulador educativo permite:

- **Comprender** el funcionamiento interno de los procesadores sin necesidad de hardware real.
- **Visualizar** cómo el ciclo de instrucción controla el flujo del programa.
- **Analizar** cómo distintas formas de direccionamiento impactan en la eficiencia del acceso a datos.
- **Relacionar** la teoría de arquitecturas de computadores con la práctica en un entorno controlado.

De esta forma, el simulador se convierte en una herramienta didáctica que facilita la enseñanza y aprendizaje de los fundamentos de la arquitectura de computadoras y la programación en lenguaje ensamblador.

III. Diseño del simulador: (explicación del código.)

El simulador fue diseñado en Python bajo una arquitectura modular, utilizando programación orientada a objetos.

3.1 Estructura de la clase principal

La clase SimuladorEnsamblador concentra los elementos centrales de la CPU simulada:

- Registros implementados:
  - ACC (Acumulador).
  - PC (Contador de programa).
  - R1 y R2 (registros de propósito general).
- Memoria:
  - Representada como un arreglo de tamaño configurable (self.memoria), inicializado en ceros.
- Programa:
  - Almacenado como una lista de instrucciones (self.programa), cada una representada como lista [opcode, operando].
- Estado de ejecución:
  - Bandera self.ejecutando que controla la ejecución del ciclo.

3.2 Instrucciones implementadas

El simulador incluye un conjunto básico de instrucciones:

- CARGA x → carga el valor x en el acumulador.
- ALMACENA x → guarda el valor del acumulador en la dirección x de memoria.
- SUMA x → suma el valor x al acumulador.
- RESTA x → resta el valor x al acumulador.
- SALIDA x → muestra en consola el valor de salida y el estado del acumulador.
- SALTA x → salto incondicional a la instrucción ubicada en la posición x.
- SALTA_SI_CERO x → salto condicional a x si el acumulador es cero.
- DETENER → finaliza la ejecución del programa.

Estas instrucciones son equivalentes a las operaciones básicas de un procesador real (LOAD, STORE, ADD, SUB, JMP, JZ, OUTPUT, HALT).

3.3 Ciclo de ejecución

El método ejecutar() implementa el ciclo fetch–decode–execute:

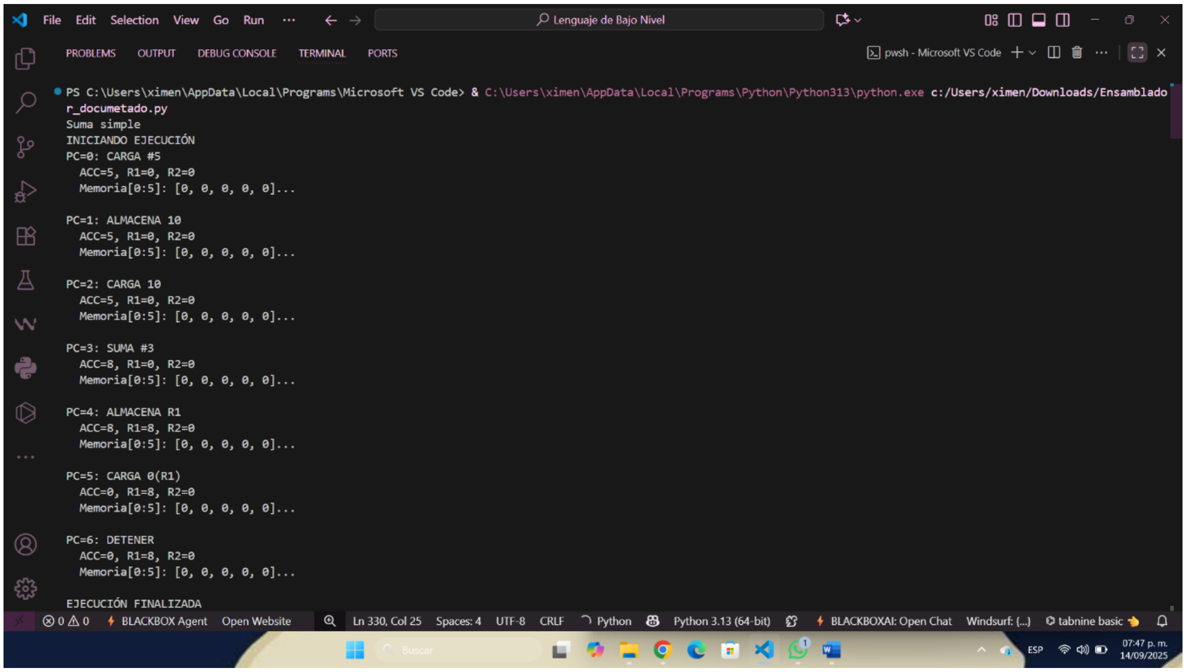
- Fetch: obtener la instrucción de la posición PC.
- Decode: separar opcode y operando.
- Execute: invocar el método correspondiente a la instrucción.

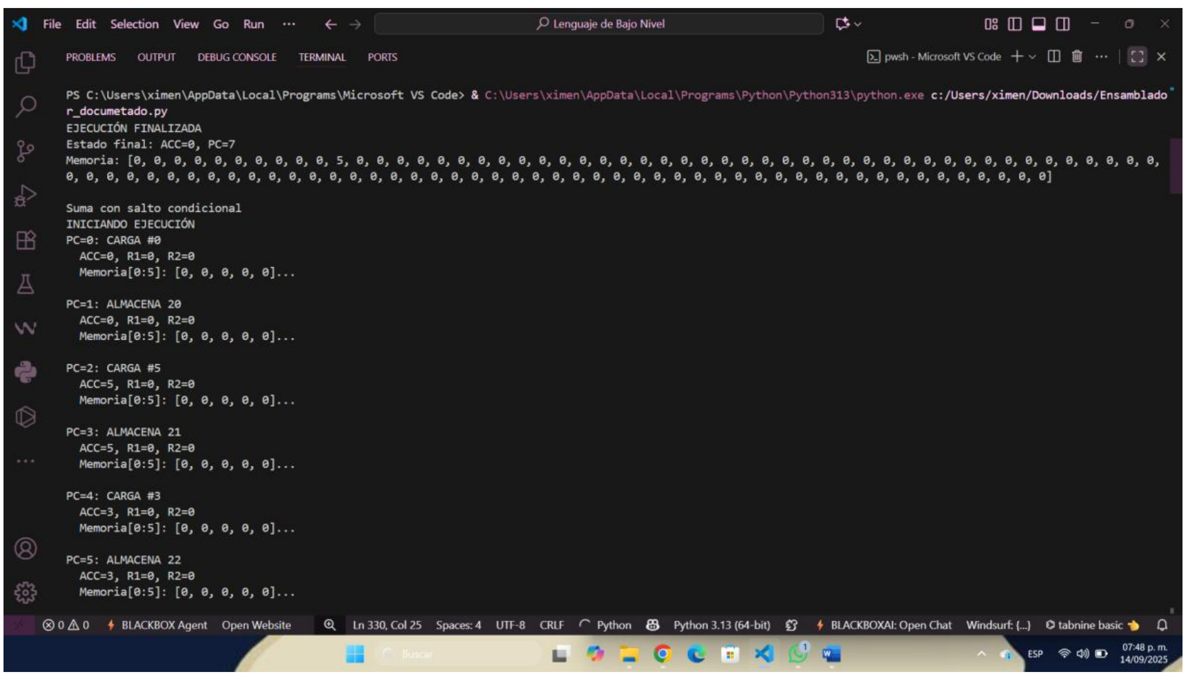
Actualizar estado: modificar registros y memoria según sea necesario.

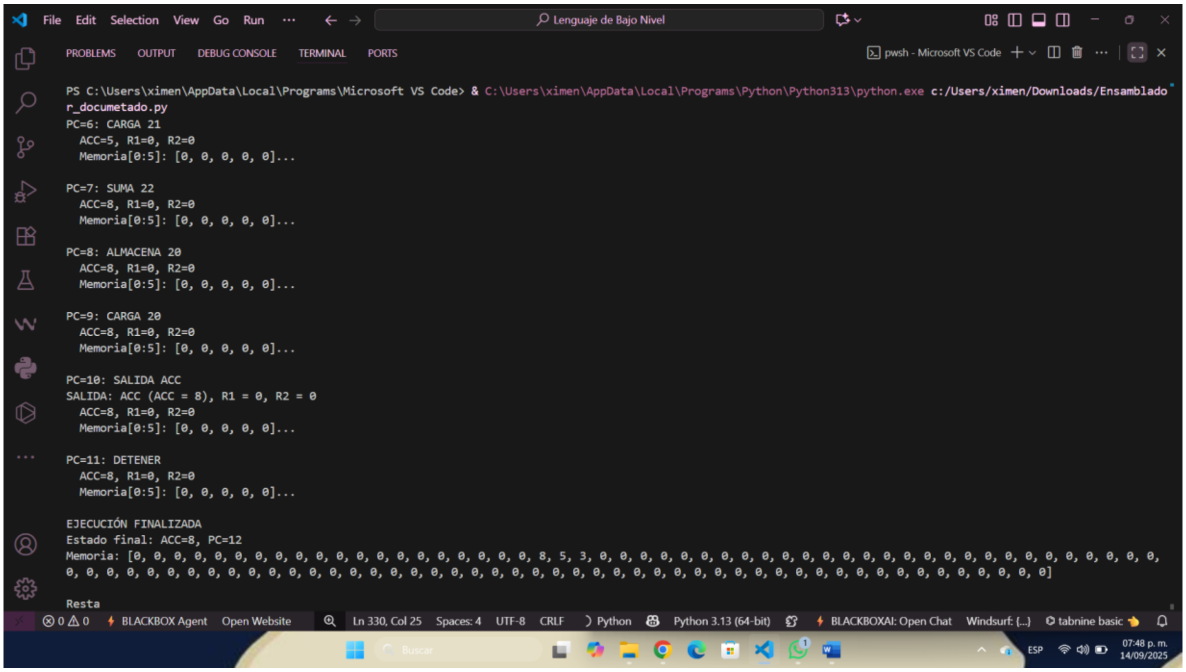
Se incluye trazado en consola para mostrar:

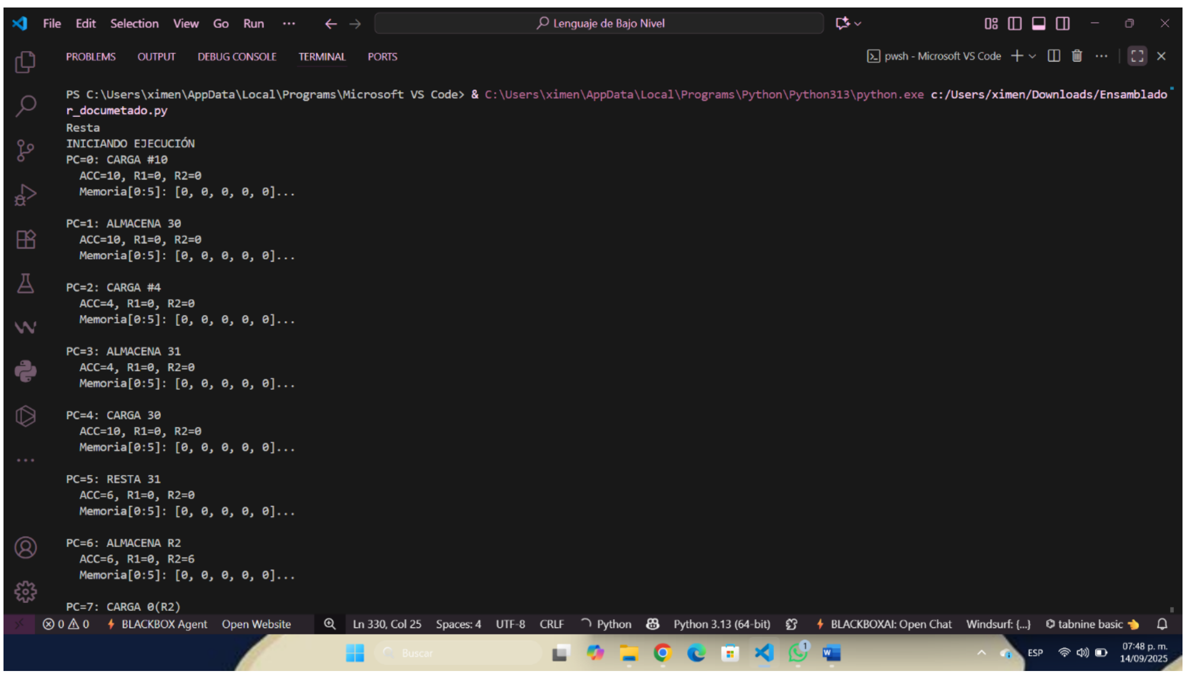
Valor de PC, instrucción actual, estado de registros y primeras posiciones de memoria.

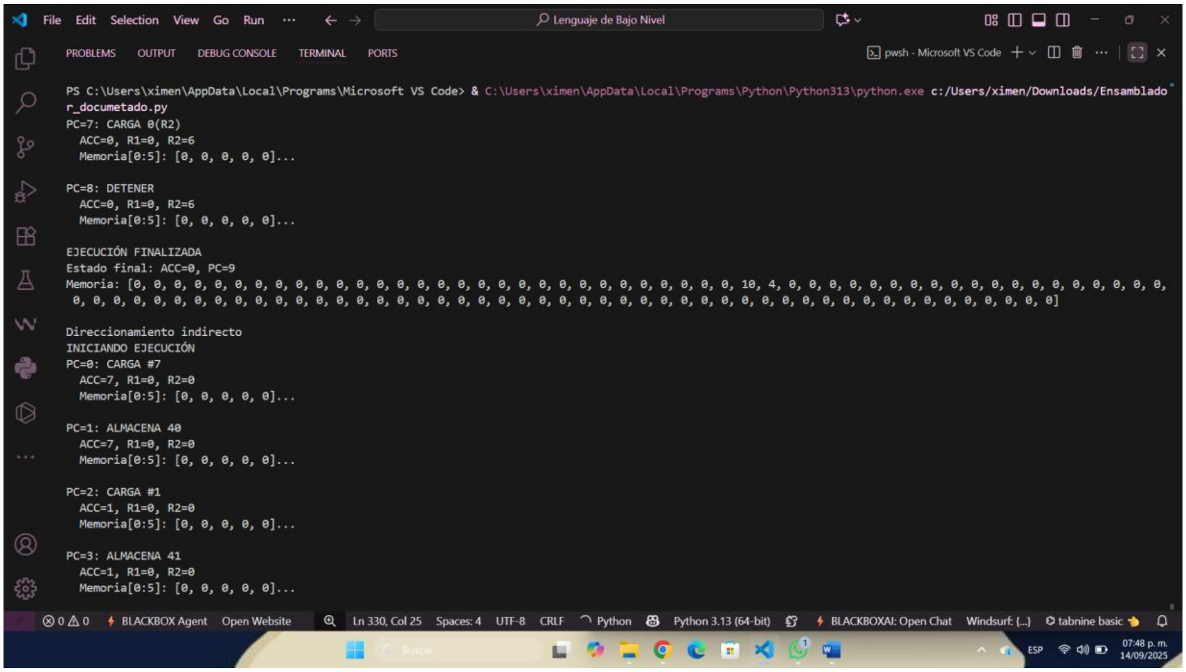
IV. Resultados de ejecución: (capturas de pantalla del simulador.)

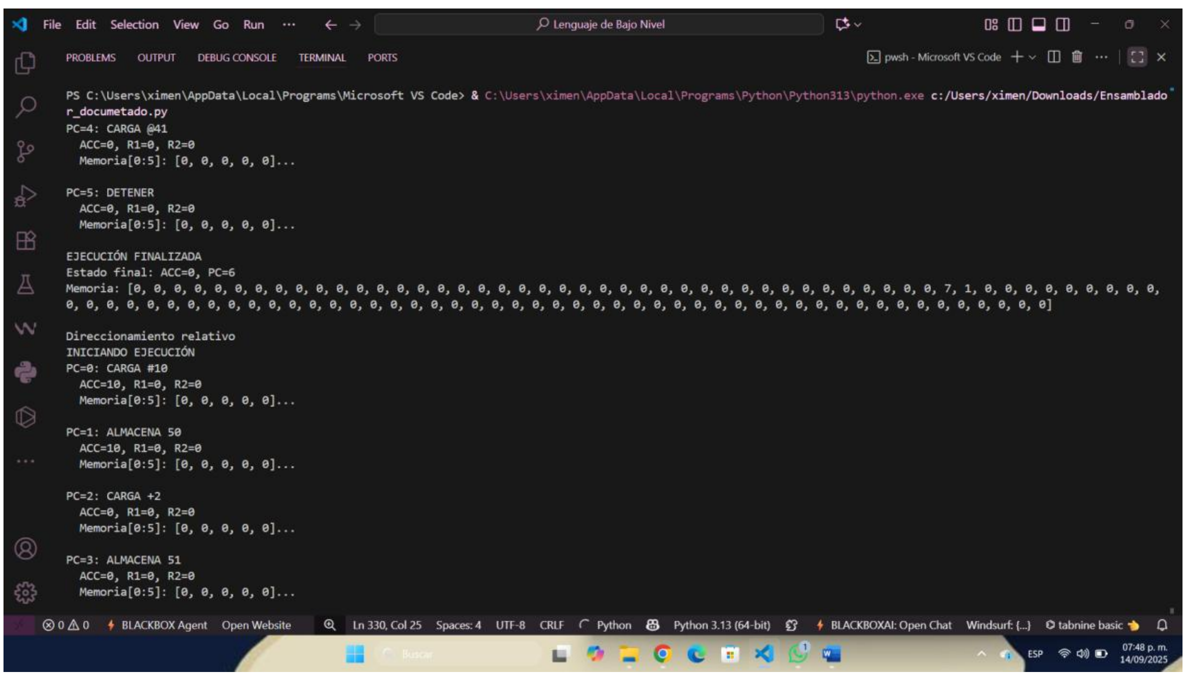


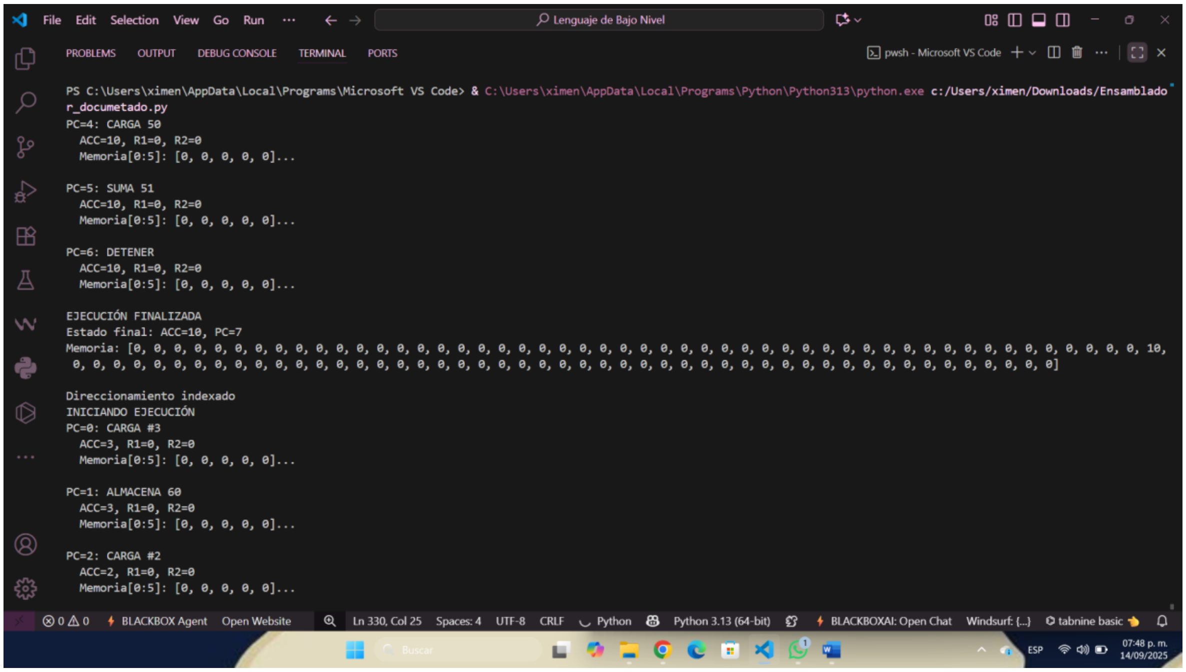


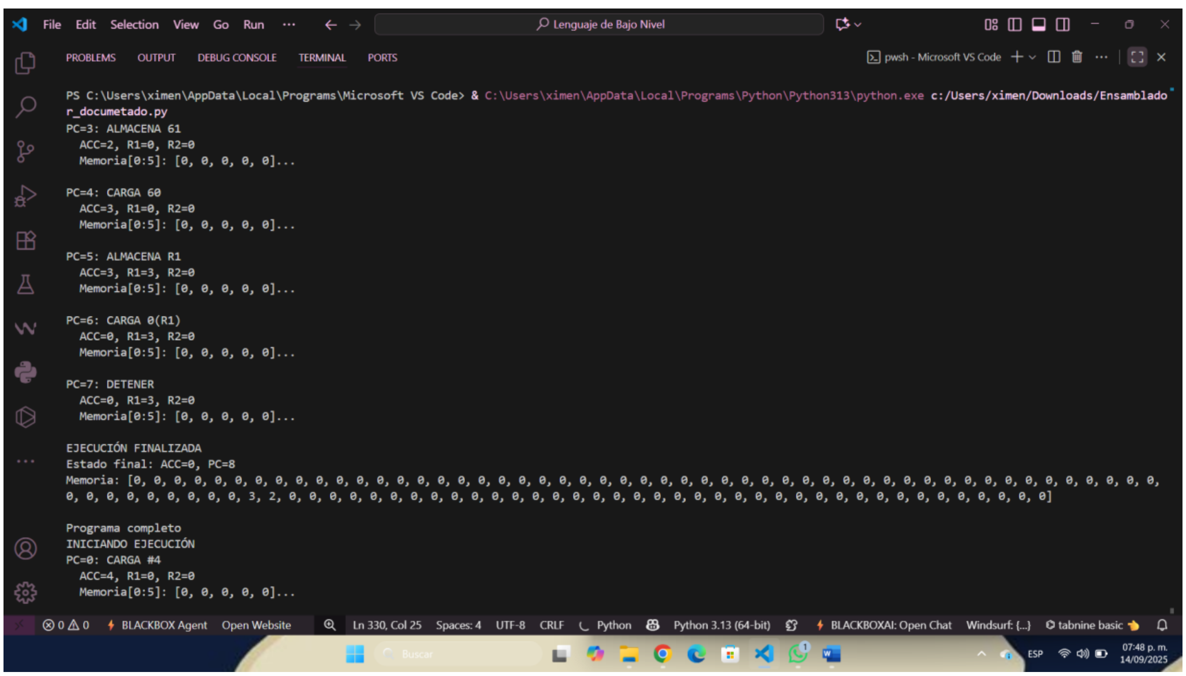

File Edit Selection View Go Run ···
Lenguaje de Bajo Nivel
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh - Microsoft VS Code
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
PC=6: CARGA 21
ACC=5, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=7: SUMA 22
ACC=8, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=8: ALMACENA 20
ACC=8, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=9: CARGA 20
ACC=8, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=10: SALIDA ACC
SALIDA: ACC (ACC = 8), R1 = 0, R2 = 0
ACC=8, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=11: DETENER
ACC=8, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
EJECUCIÓN FINALIZADA
Estado final: ACC=8, PC=12
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 5, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Resta
0 0 BLACKBOX Agent Open Website Ln 330, Col 25 Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit) BLACKBOXAI: Open Chat Windsurf: {...} tabnine basic
ESP 07:48 p. m. 14/09/2025

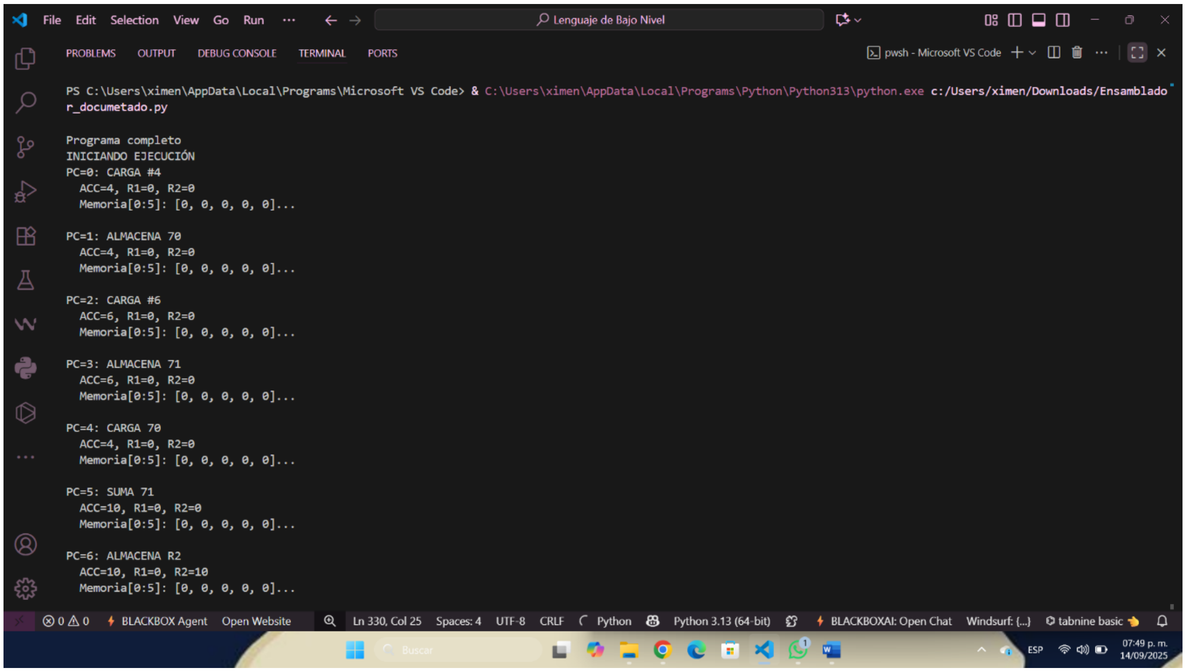

File Edit Selection View Go Run ···
Lenguaje de Bajo Nivel
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh - Microsoft VS Code
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
Resta
INICIANDO EJECUCIÓN
PC=0: CARGA #10
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=1: ALMACENA 30
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=2: CARGA #4
ACC=4, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=3: ALMACENA 31
ACC=4, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=4: CARGA 30
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=5: RESTA 31
ACC=6, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=6: ALMACENA R2
ACC=6, R1=0, R2=6
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=7: CARGA 0(R2)
0 0 BLACKBOX Agent Open Website Ln 330, Col 25 Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit) BLACKBOXAI: Open Chat Windsurf: {...} tabnine basic
ESP 07:48 p. m. 14/09/2025


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
PC=7: CARGA 0(R2)
ACC=0, R1=0, R2=6
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=8: DETENER
ACC=0, R1=0, R2=6
Memoria[0:5]: [0, 0, 0, 0, 0]...
EJECUCIÓN FINALIZADA
Estado final: ACC=0, PC=9
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Direccionamiento indirecto
INICIANDO EJECUCIÓN
PC=0: CARGA #7
ACC=7, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=1: ALMACENA 40
ACC=7, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=2: CARGA #1
ACC=1, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=3: ALMACENA 41
ACC=1, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...

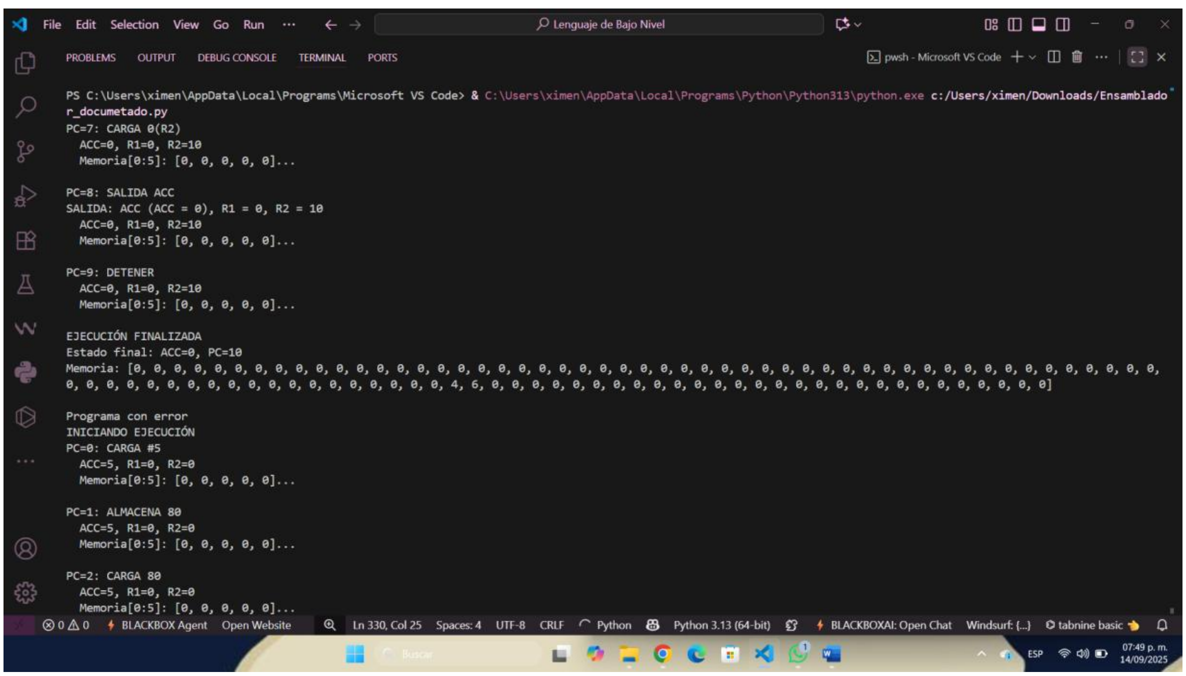

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
PC=4: CARGA @41
ACC=0, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=5: DETENER
ACC=0, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
EJECUCIÓN FINALIZADA
Estado final: ACC=0, PC=6
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Direccionamiento relativo
INICIANDO EJECUCIÓN
PC=0: CARGA #10
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=1: ALMACENA 50
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=2: CARGA +2
ACC=0, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=3: ALMACENA 51
ACC=0, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...


File Edit Selection View Go Run ···
Lenguaje de Bajo Nivel
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh - Microsoft VS Code
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
PC=4: CARGA 50
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=5: SUMA 51
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=6: DETENER
ACC=10, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
EJECUCIÓN FINALIZADA
Estado final: ACC=10, PC=7
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Direccionamiento indexado
INICIANDO EJECUCIÓN
PC=0: CARGA #3
ACC=3, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=1: ALMACENA 60
ACC=3, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=2: CARGA #2
ACC=2, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
BLACKBOX Agent Open Website Ln 330, Col 25 Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit) BLACKBOXAI: Open Chat Windsurf: {...} tabnine basic
ESP 07:48 p. m. 14/09/2025

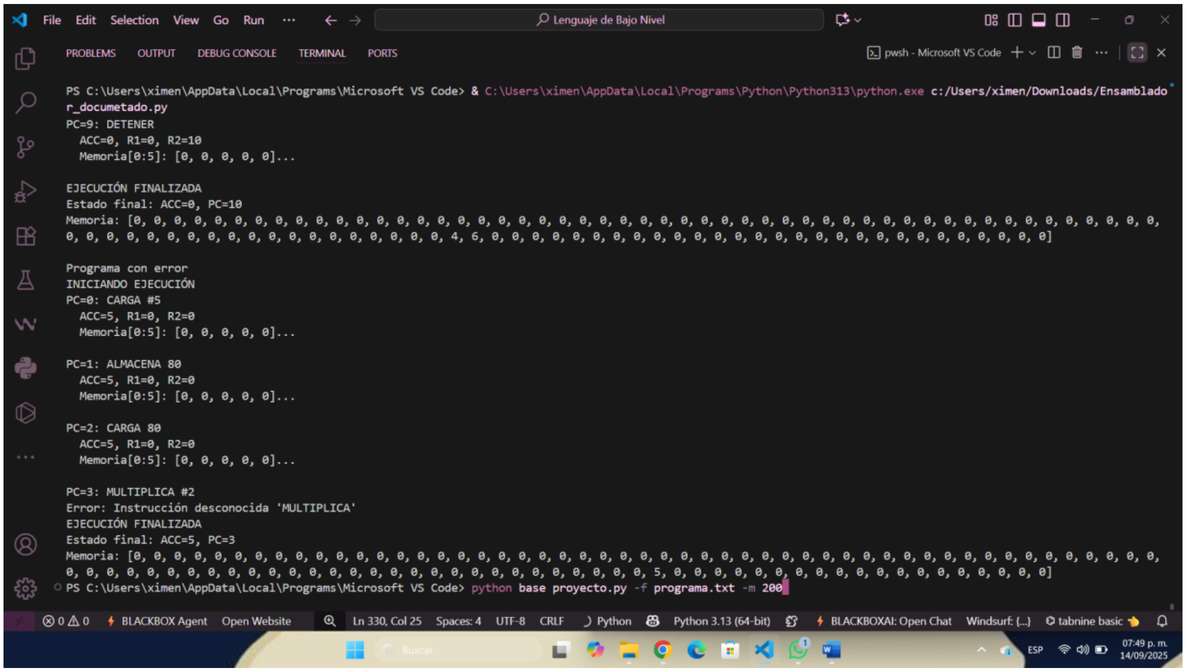

File Edit Selection View Go Run ···
Lenguaje de Bajo Nivel
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh - Microsoft VS Code
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblador_documetado.py
PC=3: ALMACENA 61
ACC=2, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=4: CARGA 60
ACC=3, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=5: ALMACENA R1
ACC=3, R1=3, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=6: CARGA 0(R1)
ACC=0, R1=3, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
PC=7: DETENER
ACC=0, R1=3, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
EJECUCIÓN FINALIZADA
Estado final: ACC=0, PC=8
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Programa completo
INICIANDO EJECUCIÓN
PC=0: CARGA #4
ACC=4, R1=0, R2=0
Memoria[0:5]: [0, 0, 0, 0, 0]...
BLACKBOX Agent Open Website Ln 330, Col 25 Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit) BLACKBOXAI: Open Chat Windsurf: {...} tabnine basic
ESP 07:48 p. m. 14/09/2025



```
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblado
r_documetado.py

Programa completo
INICIANDO EJECUCIÓN
PC=0: CARGA #4
  ACC=4, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=1: ALMACENA 70
  ACC=4, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=2: CARGA #6
  ACC=6, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=3: ALMACENA 71
  ACC=6, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=4: CARGA 70
  ACC=4, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=5: SUMA 71
  ACC=10, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=6: ALMACENA R2
  ACC=10, R1=0, R2=10
  Memoria[0:5]: [0, 0, 0, 0, 0]...
```



```
PS C:\Users\ximen\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ximen\AppData\Local\Programs\Python\Python313\python.exe c:/Users/ximen/Downloads/Ensamblado
r_documetado.py
PC=7: CARGA 0(R2)
  ACC=0, R1=0, R2=10
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=8: SALIDA ACC
SALIDA: ACC (ACC = 0), R1 = 0, R2 = 10
  ACC=0, R1=0, R2=10
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=9: DETENER
  ACC=0, R1=0, R2=10
  Memoria[0:5]: [0, 0, 0, 0, 0]...

EJECUCIÓN FINALIZADA
Estado final: ACC=0, PC=10
Memoria: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Programa con error
INICIANDO EJECUCIÓN
PC=0: CARGA #5
  ACC=5, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=1: ALMACENA 80
  ACC=5, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...

PC=2: CARGA 80
  ACC=5, R1=0, R2=0
  Memoria[0:5]: [0, 0, 0, 0, 0]...
```



Link del código en github:
https://github.com/Chmena72/Lenguaje-de-Bajo-Nivel/tree/5833aa98fa9464b7d389d56366db48b504377614/Proyecto%20Unidad%201

V. Conclusiones:

El desarrollo de este simulador de CPU en Python permitió comprender de manera práctica los principios fundamentales de la arquitectura de computadoras y el funcionamiento del ciclo de instrucción *fetch–decode–execute*. A través de la implementación de registros básicos, memoria y un conjunto reducido de instrucciones en lenguaje ensamblador simplificado, se logró visualizar cómo interactúan los componentes internos de un procesador.

Asimismo, el proyecto evidenció la importancia de los modos de direccionamiento y de la correcta gestión de registros y memoria para la ejecución de programas. Aunque se trata de un modelo simplificado, el simulador constituye una base sólida para futuros trabajos que incluyan más instrucciones, distintos modos de

direccionamiento, depuración paso a paso e interfaces gráficas que hagan más interactivo el aprendizaje.

En conclusión, este trabajo no solo cumplió con el objetivo de emular el funcionamiento básico de una CPU, sino que también fomentó un aprendizaje significativo al conectar la teoría con la práctica, fortaleciendo las habilidades de programación y el entendimiento de los conceptos esenciales de la arquitectura computacional.

VI. Bibliografía:

Cratecode. (2023, 5 mayo). Assembly Registers and Memory. *Cratecode*. https://cratecode.com/info/assembly-registers-and-memory

x86 Assembly/16, 32, and 64 Bits - Wikibooks, open books for an open world. (n.d.). https://en.wikibooks.org/wiki/X86_Assembly/16,_32,_and_64_Bits

http://itpn.mx/recursosisc/6semestre/lenguajesdeinterfaz/Unidad%20I#:~:text=Registro%20SI.,de%2032%20bits%2C%20el%20ESI

Tipu, A. D. (2024, February 7). A beginners guide to Assembly language using emu8086. *Amrito's Blog*. https://amritoo.hashnode.dev/a-beginners-guide-to-assembly-language-using-emu8086

https://www.cs.buap.mx/~mgonzalez/asm_mododir2.pdf

Gonzalez, L. (2018, July 31). Tema 33 – Programación en lenguaje ensamblador. - Oposinet. *Oposinet*. https://www.oposinet.com/temario-de-informatica/temario-3-informatica/tema-33-programacin-en-lenguaje-ensamblador-2/

https://academicos.azc.uam.mx/jestrada/archivos/lsdm/2.RegSeg.pdf#:~:text=Permite%20la%20colocaci%C3%B3n%20en%20memoria%20de%20una,de%20un%20programa%20en%20el%20registro%20SS

http://wiki.sc3.uis.edu.co/images/9/96/Ensambla20.pdf#:~:text=%E2%97%A6%20La%20mitad%20inferior%20tambien%20se%20puede,desplazamiento%20de%20la%20siguiente%20instrucci%C3%B3n%20a%20ejecutar

https://www.welivesecurity.com/la-es/2014/01/28/instrucciones-registros-operadores-x86/

Estructura de computadores. (n.d.). https://cv.uoc.edu/annotation/8255a8c320f60c2bfd6c9f2ce11b2e7f/619469/PID_00218272/PID_00218272.html

https://nlaredo.tecnm.mx/takeyas/Apuntes/Lenguaje_Ensamblador/Apuntes/LIBRO_ENSAMBLADOR_2004.pdf

Introducción a Ensamblador (ASM). (n.d.). https://code0x66.blogspot.com/2014/03/introduccion-ensamblador-asm.html