Universidad Católica Boliviana

Proyecto-Simulador-de-Arquitectura-x86-UCB

***Materia**: Arquitectura de Computadoras*

***Carrera:** Ingeniería de Software*

Integrantes:

Andrés Mallea
Mijael Callejas
Israel Gutiérrez

Docente: Paulo Cesar Loayza Carrasco

06/10/2025

1. Introducción

El propósito del simulador es ilustrar el funcionamiento interno de los principales componentes de una computadora tipo x86, incluyendo la CPU, la Unidad Aritmético-Lógica (ALU), los registros, la memoria RAM y el pipeline.
El proyecto se implementó en Microsoft Excel utilizando Visual Basic for Applications (VBA), con el objetivo de crear una herramienta didáctica que permita a los estudiantes comprender de forma visual el ciclo de instrucción, las banderas (flags), el funcionamiento de la memoria caché y el flujo de ejecución de instrucciones.

Objetivos

- Simular el ciclo de instrucción de una arquitectura x86 de forma interactiva.
- Visualizar en tiempo real el cambio de valores en registros, memoria y flags.
- Mostrar el funcionamiento del pipeline a través de una animación en Excel.
- Proveer una base didáctica para el aprendizaje de los conceptos de arquitectura de computadoras.

2. Marco Teórico

El simulador se fundamenta en los principios de la arquitectura x86, que sigue el paradigma CISC (Complex Instruction Set Computer), caracterizado por un conjunto extenso de instrucciones y operaciones complejas a nivel de hardware.

2.1. Ciclo de Instrucción

El ciclo de instrucción describe las etapas que sigue la CPU para ejecutar una instrucción: Fetch, Decode, Execute, Memory y Writeback. En el simulador, este proceso se realiza paso a paso mediante la macro principal “Simular_ALU_Paso()”.

2.2. Componentes del Sistema

- CPU: Incluye la Unidad de Control, la ALU y los registros.
- Memoria: Representada por celdas de Excel que simulan RAM, Caché y Memoria Virtual.
- Pipeline: Simula el flujo de instrucciones en el rango U:Y (filas 9–28).
- Flags: ZF, CF, NF y SWAP, que indican los estados de la ALU.

3. Diseño del Simulador

El simulador fue desarrollado sobre Microsoft Excel, empleando macros en VBA para controlar la lógica de ejecución. Los principales componentes se encuentran mapeados a celdas específicas:

• PC (Program Counter): Celda C30.
• Acumulador (AC): Celda M9.
• Entradas A/B: C5 y C6.
• Registros R1–R4: M25–M28.
• Flags: P14–P17.
• RAM: C33:E54.
• Caché: H38:K41.
• Pipeline visual: U:Y (filas 9–28).

4. Historias de Usuario y Planificación del Proyecto

Este apartado presenta las historias de usuario que guiaron el desarrollo del simulador de arquitectura x86.
Las historias están agrupadas por *épicas*, que representan conjuntos de funcionalidades relacionadas.

Epic 1 – Configuración del Proyecto y Entorno de Trabajo

ID	Historia de Usuario	Criterios de Aceptación
HU-001	Como desarrollador, quiero configurar un repositorio de GitHub para colaborar y usar control de versiones.	Repositorio creado y compartido con todos los miembros; estructura de ramas establecida; acceso otorgado al docente.
HU-002	Como desarrollador, quiero configurar la plataforma de desarrollo (VBA en Excel), para comenzar a programar el simulador.	Proyecto base .xlsm configurado y cargado al repositorio.

Epic 2 – Desarrollo del Simulador

ID	Historia de Usuario	Criterios de Aceptación
HU-003	Como usuario, quiero ingresar código ensamblador x86 en una interfaz.	El simulador puede leer el contenido de las celdas del programa (filas 9–28).
HU-004	Como usuario, quiero ejecutar el código paso a paso.	El botón “Ejecutar Paso” procesa una instrucción por vez y actualiza registros y memoria.
HU-005	Como usuario, quiero visualizar los registros de la CPU.	Se muestran R1–R4, AC, y F5 actualizados tras cada instrucción.
HU-	Como usuario, quiero visualizar la	Se muestra el contenido de C33:E54,

ID	Historia de Usuario	Criterios de Aceptación
006	memoria RAM.	actualizándose tras operaciones de memoria.

Epic 3 – Documentación y Presentación

ID	Historia de Usuario	Criterios de Aceptación
HU-007	Como equipo, quiero preparar los contenidos teóricos de la presentación.	Se redactan los apartados teóricos: CPU, Memoria, Ciclo de Instrucción, Pipeline.
HU-008	Como equipo, quiero redactar la documentación del proyecto.	Se crea y completa el documento Word con la estructura APA.

Epic 4 – Funcionalidades Avanzadas de Visualización

ID	Historia de Usuario	Criterios de Aceptación
HU-009	Como usuario, quiero ver cómo la ALU procesa operaciones aritméticas y lógicas.	El simulador muestra operandos, resultado y actualiza los flags.
HU-010	Como usuario, quiero visualizar el ciclo de instrucción completo.	Cada etapa del ciclo se representa visualmente.
HU-011	Como usuario, quiero visualizar la memoria caché y sus políticas.	La caché (H38:K41) muestra “hit” o “miss” con política LRU o FIFO.
HU-012	Como usuario, quiero ver el pipeline de instrucciones.	Se observa la ejecución paralela de instrucciones y riesgos simples.

Asignación del Trabajo

Desarrollador	Responsabilidades Principales	Historias Asignadas
Andrés Mallea	Configuración de entorno y carga de código	HU-9, HU-10, HU-3, HU-8
Mijael Callejas	Lógica de ejecución y registros	HU-1, HU-2, HU-5, HU-6, HU-7
Israel Gutiérrez	Simulación de memoria y documentación	HU-3, HU-4, HU-11, HU-12

5. Desarrollo e Implementación

El código principal del simulador está escrito en VBA. Las macros controlan tanto la lógica del CPU como la representación visual de la memoria y pipeline.

Las funciones más importantes son:

- Resetear_Simulacion(): Inicializa todos los registros, memoria y flags.
- Simular_ALU_Paso(): Ejecuta una instrucción por ciclo.
- Cargar_Caché_Controlada(): Simula el comportamiento de la memoria caché.
- Resaltar_RAM_Activa(): Destaca la sección de RAM en uso.
- SimularPipelineDinamico(): Realiza la animación del pipeline de instrucciones.

6. Resultados

El simulador desarrollado permite ejecutar de manera controlada e independiente las principales funciones de una arquitectura x86 simplificada.
A través del primer botón de "Ejecutar", los usuarios pueden observar cómo las instrucciones —como MOVE, SUMA y RESTA— modifican los valores del acumulador (AC), los registros, la memoria RAM y los flags asociados.

De forma complementaria, mediante un segundo control o botón, se activa la animación del pipeline, la cual representa de manera visual el recorrido de las instrucciones a través de las etapas del procesador.
Aunque ambas funciones operan de manera separada, su ejecución conjunta aunque no se puede sincronizar perfectamente por la velocidad de la animación del pipeline permite comprender la relación entre la lógica interna del CPU y el flujo teórico de instrucciones dentro del pipeline.

Durante las pruebas, se verificó que la actualización de registro sea coherente con las operaciones ejecutadas, y que la representación visual del pipeline contribuye a reforzar la comprensión conceptual del paralelismo y las etapas de ejecución, aun cuando no exista sincronización directa entre ambas simulaciones.

7. Conclusiones

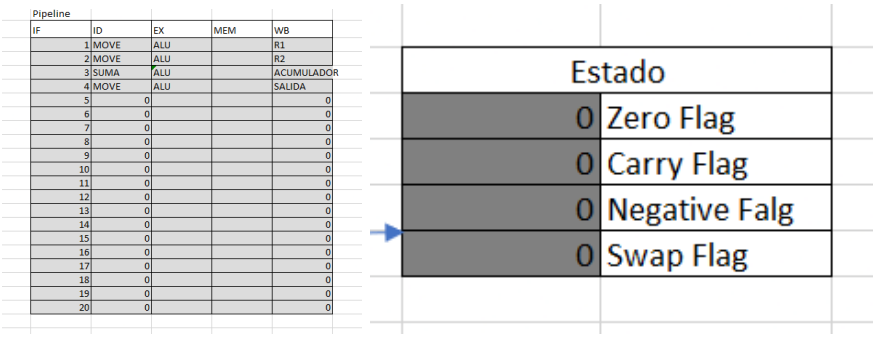
El desarrollo del Simulador de Arquitectura x86 en Excel VBA permitió demostrar que es posible representar de forma didáctica y funcional los principios fundamentales de la arquitectura de computadoras.
A través de su implementación, el equipo integró conceptos teóricos como el ciclo de instrucción, la organización de la CPU, la ALU, los registros, la memoria RAM, la caché y el pipeline, transformándolos en una herramienta visual e interactiva.

La aplicación no solo cumple los objetivos técnicos planteados, sino que también fortalece el proceso de aprendizaje, al ofrecer una experiencia práctica que permite observar el flujo de ejecución y la relación entre hardware y software.
Asimismo, el proyecto fomenta la colaboración interdisciplinaria, el uso de herramientas modernas de control de versiones (GitHub) y la aplicación de metodologías ágiles mediante historias de usuario, reflejando un enfoque profesional en su desarrollo.

Finalmente, el simulador se consolida como un recurso educativo accesible y expandible, capaz de apoyar el estudio de la arquitectura x86 y servir de base para futuras mejoras, como la incorporación de detección de riesgos en el pipeline, políticas avanzadas de reemplazo de caché o la traducción de código C a ensamblador.

8. Anexos

• Capturas de pantalla del pipeline y flags.

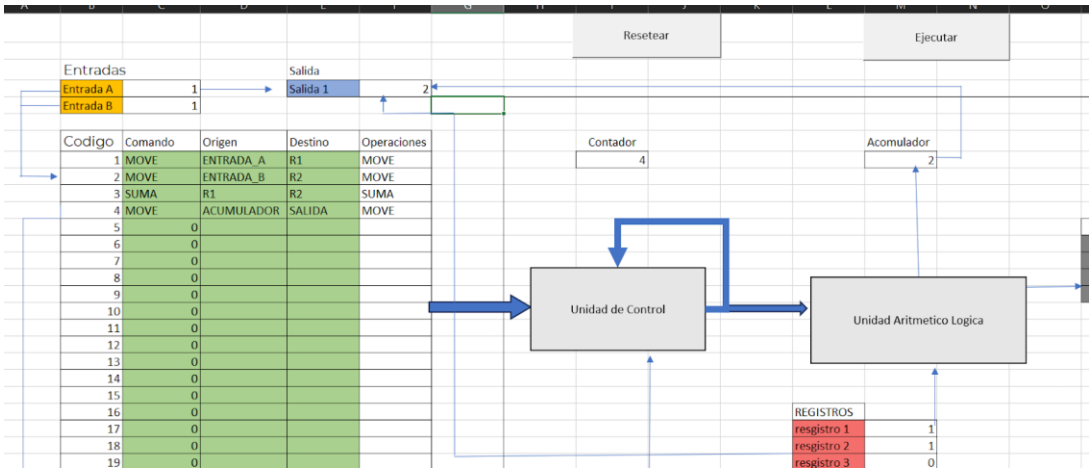


Pipeline

	IF	ID	EX	MEM	WB
1	MOVE	ALU			R1
2	MOVE	ALU			R2
3	SUMA	ALU			ACUMULADOR
4	MOVE	ALU			SALIDA
5	0				0
6	0				0
7	0				0
8	0				0
9	0				0
10	0				0
11	0				0
12	0				0
13	0				0
14	0				0
15	0				0
16	0				0
17	0				0
18	0				0
19	0				0
20	0				0

Estado	
0	Zero Flag
0	Carry Flag
0	Negative Falg
0	Swap Flag

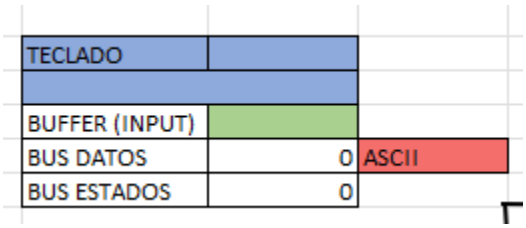
• Tablas de resultados de las pruebas de ejecución.



Segundo Parcial

Dispositivo de Entrada:

El dispositivo seleccionado fue un teclado, el cual fue implementado de manera rustica dentro del sistema.



Componentes:

- Buffer: Es donde se ingresan los caracteres que recibirá la máquina.
- Bus de Datos: Convierte a ASCII los caracteres obtenidos (uno por uno).
- Estado: Es el estado de interrupción que se genera al recibir esta prioridad.

Funcionamiento Simulado:

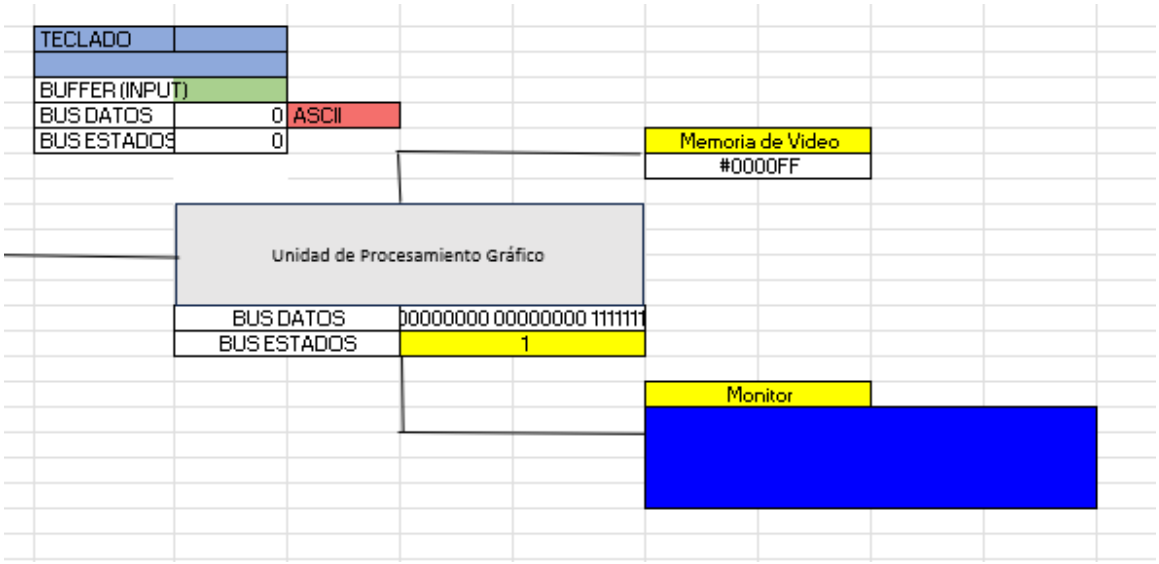
Se ingresa un carácter, el cual la CPU reconoce como interrupción, el carácter es representado como ASCII y es guardado en la memoria del sistema para su posterior uso.

Funcionamiento Real:

Un teclado funciona en un sistema operativo como un dispositivo de entrada que envía señales eléctricas al procesador para que interprete y muestre las teclas presionadas en pantalla. El sistema operativo recibe estas señales a través del controlador del teclado, interpreta los "códigos de escaneo" y los traduce en letras, números o comandos, utilizando el diseño del teclado seleccionado por el usuario. Para hacer esto, los datos del teclado (como USB o Bluetooth) viajan al procesador, que los procesa y los hace visibles en la aplicación correspondiente.

Dispositivo de Salida:

El dispositivo seleccionado fue un monitor, el cual fue implementado de manera rustica dentro del sistema.



Componentes:

- Memoria de Video: Es donde se almacenan cada uno de los pixeles que cuenta la pantalla.
- Bus de Datos: Convierte en este caso la información de la memoria de video en binario para su posterior ejecución.
- Estado: Es el estado de interrupción que se genera al recibir esta prioridad.

Funcionamiento Simulado:

Se ingresa de manera manual un tipo de color en hexadecimal en la memoria de video, luego se ejecuta el sistema provocando que la Unidad de Control determine la acción de convertir el color hexadecimal en binario mediante el Controlador de Gráficos y visualizando en el monitor.

Funcionamiento Real:

Un sistema operativo maneja un monitor como dispositivo de salida al recibir datos visuales de la tarjeta gráfica, procesarlos y enviarlos al monitor para que este los muestre en pantalla. El usuario interactúa con estas configuraciones a través del panel de control del sistema operativo, ajustando la resolución, la orientación y cómo se extienden o duplican las pantallas, e incluso seleccionando el monitor principal.

• Enlace al repositorio de GitHub.

https://github.com/Mijael-Callejas/Proyecto-Simulador-de-Arquitectura-x86-UCB