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

19/12/2025

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.

2. 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.

3. 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.

3.1. 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()”.

3.2. Componentes del Sistema

- **CPU**: Incluye la Unidad de Control, la ALU y los registros.
- **Memoria y Gestión de Paginación:** El sistema simula una jerarquía de memoria compuesta por:
    - **Memoria RAM Principal:** Representada en el rango C33:E44. Es el único espacio donde la CPU puede ejecutar instrucciones directamente.
    - **Memoria Virtual:** Representada en el rango C47:E54. Almacena el código que excede la capacidad de la RAM.
    - **Mecanismo de Swapping:** Se implementa un sistema de intercambio circular. Cuando el Program Counter solicita una instrucción que reside en la Memoria Virtual, el sistema traslada dicha instrucción a la última posición física de la RAM (fila 44) y mueve la instrucción que ocupaba ese lugar hacia la Memoria Virtual, activando la Swap Flag para indicar el fallo de página.
- **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.
- **Teclado(Simulado)-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.

- **Monitor(Simulado)-Componentes:**
    - **Memoria de Video:** Es donde se almacenan cada uno de los píxeles 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.

4. 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).
- Buffer de Teclado:M38-N40
- Memoria de Video: R41-S41

5. 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-006	Como usuario, quiero visualizar la memoria RAM.	Se muestra el contenido de C33:E54, 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

6. 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.

▪ 6.1. Funciones Importantes del Simulador:

- **Resetear_Simulacion():** Restablece todos los registros (EAX, EBX, etc.), contadores y flags a su estado inicial. Además, reconstruye las fórmulas de vinculación en la RAM para asegurar que, tras un intercambio de memoria, la interfaz visual vuelva a reflejar el estado original del programa cargado.
- **Simular_ALU_Paso()**: Es el núcleo del ciclo de instrucción. Gestiona las etapas de Fetch y Execute, actualizando los acumuladores y registros. Antes de cada ejecución, verifica la posición del PC para determinar si es necesario realizar un acceso a memoria virtual.
- **Swapping_Individual_Circular(PC_Actual, ws):** Implementa la lógica de paginación. Si el PC apunta a una dirección fuera de la RAM física (instrucción > 20), este procedimiento:
  - Localiza la instrucción requerida en la Memoria Virtual.
  - Realiza un intercambio (swap) entre esta instrucción y la contenida en la última fila de la RAM (buffer de intercambio).
  - Activa visualmente la bandera Swap Flag (P17) en verde para alertar al usuario del movimiento de datos.
  - Incluye validaciones de seguridad para evitar cargar instrucciones vacías (ceros) que podrían corromper el flujo del programa.
- **Cargar_Caché_Controlada():** Simula el comportamiento de la memoria caché.
- **Resaltar_RAM_Activa():** Controla la retroalimentación visual. Ha sido optimizada para mapear correctamente las instrucciones virtuales a su ubicación física temporal en la RAM (última fila), asegurando que el usuario siempre vea iluminada la celda donde realmente ocurre la ejecución física.
- **SimularPipelineDinamico():** Genera la animación en la matriz U9:Y28. Representa visualmente cómo múltiples instrucciones atraviesan las 5 etapas (IF, ID, EX, MEM, WB) a lo largo del tiempo, ilustrando el concepto de paralelismo, aunque su ejecución es asíncrona respecto al ciclo paso a paso de la ALU.

▪ 6.2. Logica de Simulacion de la CPU:

El núcleo del simulador reside en el procedimiento Simular_ALU_Paso(), el cual modela el comportamiento de la Unidad de Control y la ALU en un ciclo secuencial. A diferencia de un procesador real que realiza esto en micro-operaciones eléctricas, el simulador utiliza la lógica de programación estructurada de VBA para replicar las siguientes fases:

- **Fase de Búsqueda (Fetch)**
  - **Control del PC:** El sistema utiliza la variable global G_ContadorPC como puntero de instrucción.
  - **Validación de Memoria:** Antes de leer la instrucción, el sistema verifica si la dirección solicitada reside en la RAM física o en la Memoria Virtual. Si es necesario, invoca la subrutina de *Swapping* para traer la instrucción al espacio direccionable de la RAM (filas 33-44).

- **Resaltado Visual:** Se invoca a Resaltar_RAM_Activa() para indicar gráficamente al usuario qué instrucción se está procesando en la memoria.

- **Fase de Decodificación (Decode)**
  - **Interpretación de Operandos:** El simulador lee las celdas de *Comando*, *Origen* y *Destino* de la instrucción activa.
  - **Abstracción de Hardware:** Utiliza funciones auxiliares (ObtenerValorOrigen) para traducir nombres simbólicos (como "R1", "ACUMULADOR" o "ENTRADA_A") a sus valores numéricos actuales almacenados en las celdas correspondientes (ej. M25, M9, C5).
- **Fase de Ejecución y Escritura (Execute & Writeback)**
  - **Procesamiento en la ALU:** Mediante una estructura de control Select Case, el sistema determina la operación aritmética o lógica a realizar (SUMA, RESTA, MOVE, AND, OR).
  - **Cálculo:** Se realizan las operaciones matemáticas sobre las variables internas.
  - **Actualización de Estado (Flags):** Inmediatamente después del cálculo, se llama al procedimiento Actualizar_Flags, el cual evalúa el resultado para encender o apagar los indicadores de estado:
    - *Zero Flag (ZF):* Si el resultado es 0.
    - *Negative Flag (NF):* Si el resultado es menor a 0.
    - *Carry Flag (CF):* Si ocurre un desbordamiento en la operación.
  - **Escritura (Writeback):** Finalmente, el resultado se escribe en la celda destino correspondiente (ej. actualizando el valor de la celda del Acumulador en la hoja de cálculo), completando así el ciclo.

▪ 6.3. Funcionamiento Detallado y Funciones del Pipeline:

A diferencia de la ejecución paso a paso de la ALU (que es lógica y matemática), el módulo del Pipeline es eminentemente visual y algorítmico. Su objetivo es representar el paralelismo de instrucciones (ILP) en una matriz de tiempo-espacio.

- **Función Principal**: SimularPipelineDinamico() Esta es la función "motor" del módulo. No se limita a calcular un resultado, sino que orquesta la animación en la hoja de cálculo. Su lógica interna opera en los siguientes pasos:
  - **Inicialización y Limpieza:** Al invocarse, la función primero limpia el rango visual U9:Y28 para eliminar rastros de ejecuciones anteriores, asegurando que la matriz de estados esté vacía.
  - **Bucle de Instrucciones** (Eje Vertical): La función recorre las filas del Código Fuente (filas 9 a 28). Por cada instrucción válida encontrada (ej. MOVE, SUMA), asigna una fila correspondiente en la matriz del pipeline.
  - **Bucle de Ciclos de Reloj** (Eje Horizontal): Para simular el paso del tiempo, la función desplaza la "pintura" de las etapas hacia la derecha.
    - *Ciclo T1:* Instrucción 1 entra en IF.
    - *Ciclo T2:* Instrucción 1 pasa a ID, Instrucción 2 entra en IF.
    - *Ciclo T3:* Instrucción 1 pasa a EX, Instrucción 2 pasa a ID, Instrucción 3 entra en IF. Esta cascada se logra mediante bucles anidados que actualizan el color y texto de las celdas U:Y.

- **Funciones Auxiliares y Relacionadas:** Aunque SimularPipelineDinamico realiza el trabajo pesado visual, depende del estado generado por otras funciones del sistema:
  - **Lectura de Estado (Simular_ALU_Paso):** Aunque el pipeline corre su propia animación, conceptualmente refleja lo que ocurre en Simular_ALU_Paso. El pipeline visualiza la *clase* de operación que la ALU ya ha validado (por ejemplo, si es una operación aritmética para mostrar en la etapa EX).
  - **Interacción con Memoria:** En la etapa MEM, la función del pipeline consulta implícitamente si la instrucción involucra operandos de memoria (como R1 o direcciones RAM) para decidir si esa etapa se "ilumina" como activa o pasa como inactiva (burbuja o NOP).
- **Mapeo de Etapas en la Función**: La función traduce las operaciones de ensamblador a las 5 etapas clásicas de la siguiente manera:
  1. IF (Instruction Fetch - Columna U): Simula la lectura de la celda de código (ej. C9).
  2. ID (Instruction Decode - Columna V): Interpreta el comando (ej. "SUMA") y lee los registros origen.
  3. EX (Execute - Columna W): Es la etapa crítica. Si la instrucción es SUMA o RESTA, la función marca esta celda para indicar uso de la ALU. Si es MOVE, esta etapa es de tránsito simple.
  4. MEM (Memory Access - Columna X): Se activa visualmente si la instrucción implica leer/escribir en la RAM simulada (C33:E44).
  5. WB (Write Back - Columna Y): Muestra visualmente el destino final del dato (ej. escribir en ACUMULADOR o R1).

6.3. Implementacion Tecnica en Exel:

La visualización del pipeline se gestiona mediante la macro SimularPipelineDinamico(). Esta funcionalidad opera de la siguiente manera:

- **Matriz de Visualización:** Se utiliza el rango de celdas U9:Y28 en la hoja de cálculo como una matriz de tiempo-espacio.
  - Las filas representan las instrucciones en orden de ejecución.
  - Las columnas (U a Y) representan las 5 etapas (IF, ID, EX, MEM, WB).
- **Animación Secuencial:** El algoritmo recorre las instrucciones activas y "pinta" progresivamente las celdas hacia la derecha, simulando el paso del tiempo (ciclos de reloj).
- **Independencia de Ejecución:** Debido a las limitaciones de sincronización en VBA, la animación del pipeline funciona como una representación educativa complementaria a la ejecución paso a paso de la ALU. Esto permite al estudiante observar el flujo teórico sin interrumpir el cálculo lógico de los registros.

6.4. Implementaicon de Dispositivo de Entrada

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

- **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.
- **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.

6.5. Implementaicon de Dispositivo de Salida

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

- **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.
- **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.

6.6. Representación de Datos:

Para cada instrucción procesada (ej. MOVE, SUMA), el simulador rellena las celdas correspondientes en la matriz del pipeline con los datos relevantes de esa etapa:

- En **ID**, muestra el código de operación.
- En **EX**, indica el uso de la ALU.
- En **WB**, muestra el registro o dirección donde se guardará el dato final.

7. 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.

8. 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.

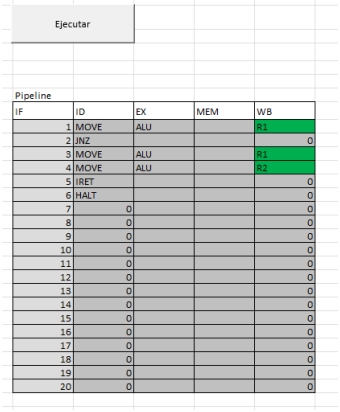
La aplicación no solo cumple los objetivos técnicos planteados, sino que también fortalece el proceso de aprendizaje, al ofrecer una experiencia práctica que permite observar el flujo de ejecución y la relación entre hardware y software.

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.

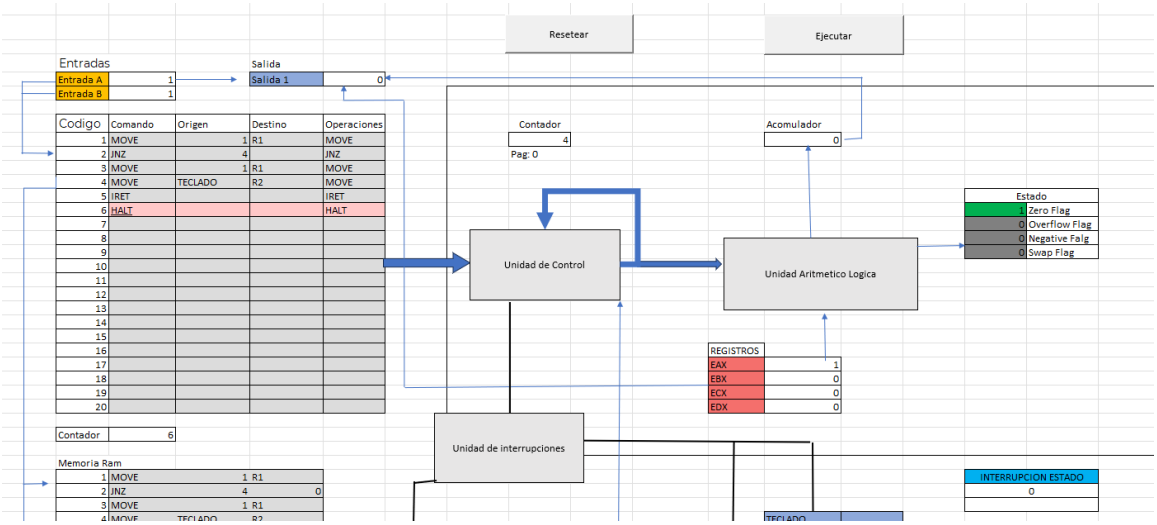
Finalmente, el simulador se consolida como un recurso educativo accesible y expandible, capaz de apoyar el estudio de la arquitectura x86 y servir de base para futuras mejoras, como la incorporación de detección de riesgos en el pipeline, políticas avanzadas de reemplazo de caché o la traducción de código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

9. Anexos

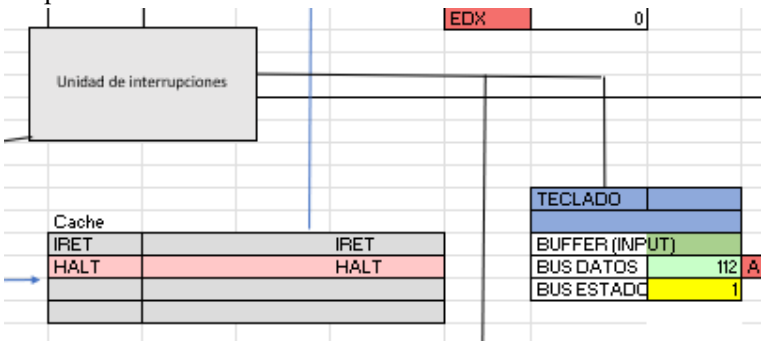
• Capturas de pantalla del pipeline y flags.



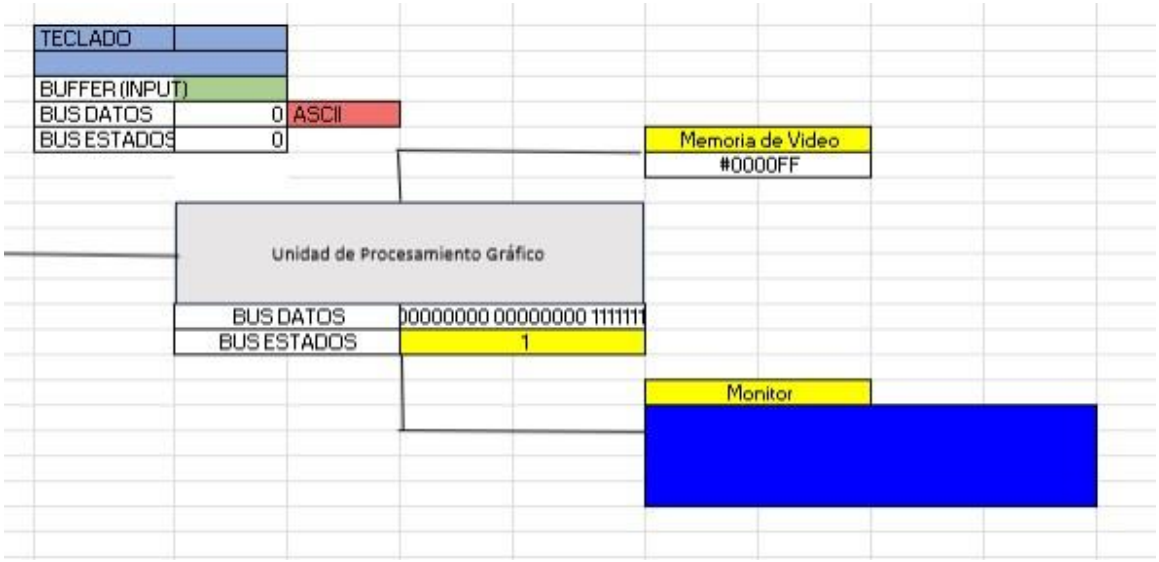
• Tablas de resultados de las pruebas de ejecución.



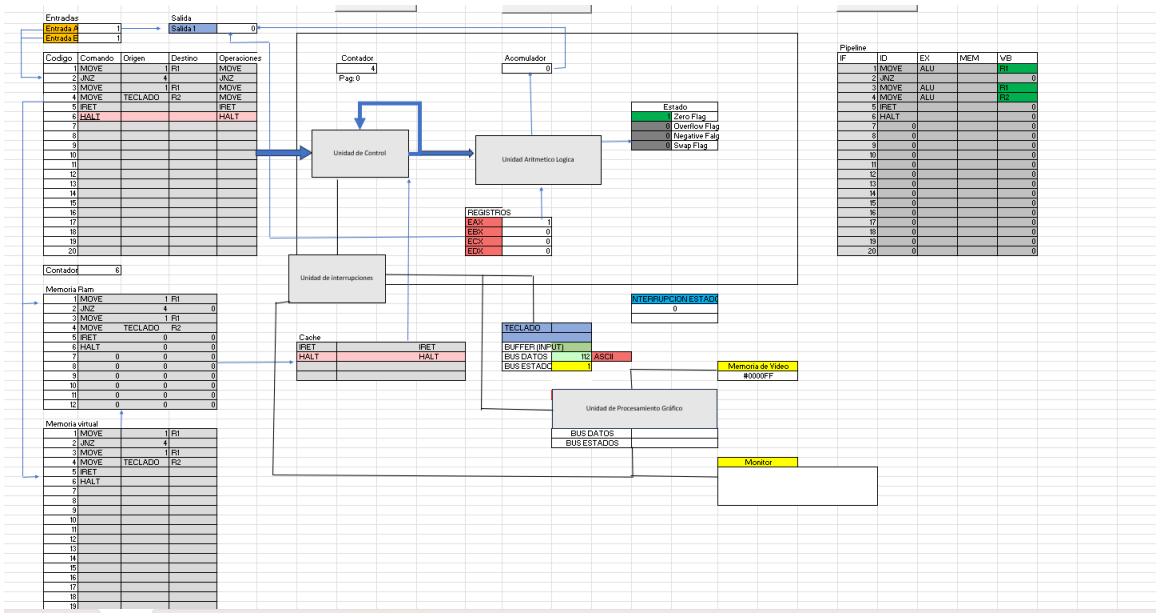
- Dispositivo de entrada Teclado:



- Dispositivo de salida Monitor:



- Sistema Completo:



• Enlace al repositorio de GitHub.
https://github.com/Mijael-Callejas/Proyecto-Simulador-de-Arquitectura-x86-UCB