SIMULADOR DE ARQUITECTURA X86

Luis M. Aguirre, Jesús B. Oliva, Leandro J. Jaldín

Ingeniería de Software, Facultad de Ciencias, Universidad Católica Boliviana

Arquitectura de Computadoras: SIS-131

Ing. Paulo César Loayza Carrasco
1 de octubre del 2025

RESUMEN (Abstract)

Este proyecto presenta un simulador de CPU implementado en Microsoft Excel (archivo .xlsm) que modela un pipeline de 5 etapas (IF, ID, EX, MEM, WB) y una memoria RAM simulada. Se describen el diseño, la implementación en VBA y los resultados de pruebas con programas de ejemplo. El simulador permite avanzar ciclo a ciclo, visualizar registros y memoria, y detectar efectos básicos de acceso a memoria. Para este proyecto nos dividimos en 3 roles. Uno que se encargaría para implementar las instrucciones de tipo MOV, ADD, SUB, INC, DEC. Simulaba los registros utilizando diccionario o variables y que mostrara visualmente el contenido de cada uno. El siguiente veía la simulación de la Memoria (RAM, Caché y Virtual), creando etiquetas, estados (válido, LRU, Memory Virtual) y los datos. Y el último realizaba el Pipeline, la Interfaz Visual del Simulador y Coordinador general del proyecto, se encargaba de realizar pruebas si se actualizaban correctamente los cambios en las hojas visuales, ver el estado del simulador y comprobar la sincronización de componentes CPU, Memoria y Pipeline para después crear el diseño final para la presentación.

INSTRODUCCIÓN

El desarrollo de simuladores de unidades centrales de procesamiento (CPU) representa una herramienta fundamental en la enseñanza de la arquitectura computacional, ya que permite visualizar de manera práctica los procesos internos de ejecución de instrucciones. En este proyecto se presenta el diseño e implementación de un simulador de CPU elaborado en Microsoft Excel, utilizando macros en VBA (Visual Basic for Applications) para automatizar la gestión de instrucciones, registros y memoria. El propósito principal de este trabajo es ofrecer una representación interactiva del funcionamiento del ciclo de instrucción, incluyendo las etapas de *fetch*, *decode*, *execute*, *memory* y *write-back*, tal como ocurre en un pipeline real de procesadores modernos.

El simulador no solo facilita la comprensión del flujo de datos y control dentro del CPU, sino que también permite analizar el uso de la memoria RAM y la evolución de los registros durante la ejecución de cada ciclo. De esta manera, se integra la teoría con la práctica mediante una plataforma accesible, potenciando el aprendizaje de conceptos como segmentación de instrucciones, paralelismo y manejo de recursos computacionales.

En síntesis, el presente proyecto busca proporcionar una herramienta educativa didáctica y técnica, que refleje los principios básicos del procesamiento secuencial y concurrente, sirviendo como apoyo tanto en cursos de arquitectura de computadoras como en entornos de autoaprendizaje sobre diseño de procesadores y simulación digital.

OBJETIVOS

Objetivo general: Construir un simulador de CPU real en Excel que muestre cada parte que es lo que hace para cada instrucción incluyendo sus 5 etapas y entender sus principios, sus comportamientos de las políticas que tienen y ver su tipo de arquitectura que tiene.

Objetivos específicos: Implementar pipeline con sus 5 etapas de la CPU; visualizar RAM; permitir avance por ciclo; documentar y registrar los avances en un Backlog y resumir los avances del proyecto en un documento Word en formato APA 7.

MARCO TEÓRICO

El diseño y simulación de una Unidad Central de Procesamiento (CPU) es un tema fundamental en el campo de la Ingeniería de Software y la Arquitectura de Computadores. Una CPU se compone de diversos subsistemas interrelacionados que permiten la ejecución de instrucciones, el manejo eficiente de datos y la interacción con la memoria principal. Entre los componentes más relevantes se encuentran el **ciclo de instrucción**, las **fases del pipeline** y la **jerarquía de memoria**.

1. Jerarquía de memoria
   La jerarquía de memoria es una organización estructural que busca optimizar el acceso a los datos según su frecuencia de uso y velocidad de recuperación. Esta se compone de distintos niveles: **registros, caché, memoria principal (RAM)** y **almacenamiento secundario o memoria virtual** (Patterson y Hennessy, 2021).
   Cada nivel presenta una compensación entre velocidad, capacidad y costo, lo que obliga al sistema a emplear **políticas de reemplazo** y **métodos de actualización** (como LRU o Write-Through) para mantener la coherencia y eficiencia en el acceso a los datos.

2. Memoria caché
   La memoria caché actúa como un intermediario entre el procesador y la memoria principal, almacenando temporalmente las instrucciones y datos más utilizados. Las políticas **LRU (Least Recently Used)** y **Write-Through** son mecanismos clásicos de gestión que permiten decidir qué datos conservar o reemplazar al producirse un fallo de caché (Hwang & Briggs, 2017). En este simulador, la caché implementa un sistema **2-Way Set Associative**, que ofrece un equilibrio entre rapidez de búsqueda y complejidad del hardware.

3. Paginación y memoria virtual
   La **memoria virtual** amplía el espacio de direcciones accesible por el procesador mediante un sistema de **paginación**, en el que las direcciones virtuales se traducen en direcciones físicas a través de una **unidad de manejo de memoria (MMU)**. Este mecanismo facilita la gestión eficiente de los recursos físicos de memoria y permite la coexistencia de múltiples procesos sin interferencias directas (Silberschatz, Galvin y Gagne, 2020).

4. Pipeline del procesador
   El **pipeline** es una técnica de paralelismo que divide la ejecución de las instrucciones en fases, permitiendo que varias instrucciones se procesen de manera simultánea en diferentes etapas. Generalmente, las fases son: **Fetch, Decode, Execute, Memory Access y Write Back**. Este método mejora significativamente el rendimiento del CPU al reducir los ciclos ociosos y aprovechar mejor los recursos internos (Stallings, 2021).

5. Simulación digital y entornos educativos
   El uso de **simuladores digitales** permite comprender los conceptos abstractos del funcionamiento interno de un procesador. Herramientas como hojas de cálculo con macros VBA o simuladores visuales son recursos educativos que facilitan la enseñanza de arquitectura de computadores mediante un entorno accesible y experimental (Tanembaum & Austin, 2013).
   En este contexto, el simulador desarrollado en Excel busca representar de manera

interactiva los procesos internos de la CPU y su memoria jerárquica, mostrando visualmente las transiciones de datos y los estados de cada componente.

METODOLOGÍA

Etapas del desarrollo

1. **Análisis conceptual:**
   Se estudiaron los componentes principales del procesador (ALU, registros, bus de datos, memoria y pipeline) y sus interacciones. También se definieron las políticas de reemplazo de caché (LRU) y de paginación (FIFO o LRU).
2. **Diseño del modelo de simulación:**
   Se estructuraron las hojas de Excel para representar:
   - **CPU:** Ejecución de instrucciones paso a paso.
   - **Memoria:** Visualización de caché, RAM y memoria virtual.
   - **Pipeline:** Diagrama de flujo que muestra las fases simultáneas de ejecución.
3. **Implementación en VBA:**
   Se desarrollaron módulos en VBA que controlan la simulación, incluyendo funciones de:
   - Carga y lectura de memoria.
   - Reemplazo de páginas y líneas de caché.
   - Ejecución de instrucciones y registro de resultados.
   - Control de estados mediante botones y eventos.
4. **Pruebas y validación:**
   Se realizaron ejecuciones de prueba con diferentes conjuntos de instrucciones para verificar la coherencia de los resultados y la correcta visualización del pipeline y la jerarquía de memoria.
5. **Optimización visual:**
   Se organizaron los controles y botones en una interfaz tipo menú, se aplicaron fuentes técnicas (como *Consolas* o *Segoe UI*) y colores diferenciados para cada componente (CPU, caché, RAM y pipeline).

   (Incluir código o fragmentos del Excel) de lo que se tenga hasta ahora.)

IMPLEMENTACIÓN

Módulo de Gestión de Memoria Virtual y Caché

La implementación del sistema de memoria se realizó mediante un módulo VBA que simula tres componentes esenciales de la arquitectura de computadoras: la memoria RAM, el sistema de caché y la unidad de gestión de memoria (MMU). El sistema fue diseñado para operar sobre una hoja de cálculo de Excel, proporcionando una representación visual e interactiva del comportamiento de la memoria.

Arquitectura de Memoria

La memoria RAM se implementó con una capacidad de 32 posiciones, organizadas en 8 páginas virtuales de 4 posiciones cada una. Esta estructura permite simular el mecanismo de paginación

utilizado en sistemas operativos modernos. La memoria física se representa en las columnas O y P de la hoja de cálculo, donde la columna O almacena las direcciones y la columna P contiene los datos correspondientes. Se definieron etiquetas predefinidas para las primeras posiciones de memoria (VAR_A, VAR_B, RESULT, TEMP) facilitando la identificación de variables en el contexto educativo.

El sistema de caché se implementó utilizando una arquitectura asociativa por conjuntos (set-associative) con 4 conjuntos y 2 vías, resultando en 8 líneas de caché totales. Esta configuración permite un balance entre el rendimiento de una caché completamente asociativa y la simplicidad de una caché de mapeo directo. La política de reemplazo seleccionada fue LRU (Least Recently Used), implementada mediante un sistema de bits que mantiene el estado de uso de cada línea dentro de su conjunto. Las latencias simuladas fueron de 1 ciclo para aciertos (hits) y 10 ciclos para fallos (misses), reflejando la diferencia de velocidad entre el acceso a caché y a memoria principal.

La MMU se implementó con una tabla de páginas que gestiona la traducción de direcciones virtuales a físicas. El sistema soporta 8 páginas virtuales que pueden mapearse a 8 marcos físicos, utilizando páginas de 4 posiciones cada una. Cuando ocurre un fallo de página (page fault), el sistema carga la página solicitada desde un área de intercambio (swap) simulada en la columna AC, con una latencia de 200 ciclos que representa el acceso a disco. El algoritmo LRU también se aplicó para el reemplazo de páginas, manteniendo contadores incrementales que registran el tiempo relativo desde el último acceso a cada página.

Funciones Principales

La función *Leer_Memoria()* constituye la interfaz principal del sistema. Esta función recibe una dirección virtual como parámetro y ejecuta el siguiente proceso: primero, traduce la dirección virtual en número de página y desplazamiento mediante operaciones de división y módulo; segundo, consulta la tabla de páginas para verificar si la página está cargada en memoria física; tercero, en caso de page fault, invoca el algoritmo de reemplazo LRU para seleccionar un marco víctima y carga la página desde el área de swap; cuarto, calcula la dirección física correspondiente; y finalmente, busca el dato en la caché, cargándolo desde RAM si es necesario. Durante todo este proceso, el sistema actualiza las señales visuales en las celdas X6 y Y7 para indicar el estado de la caché y la MMU respectivamente.

La función *Escribir_Memoria()* permite modificar el contenido de la memoria virtual. Esta función primero verifica que la página de destino esté cargada en memoria física, traduciendo la dirección virtual a física mediante la tabla de páginas. Posteriormente, actualiza directamente el valor en la RAM y, si el dato está presente en la caché, actualiza también la línea correspondiente manteniendo la coherencia entre ambos niveles de memoria. Este comportamiento simula una política de escritura write-through.

La función *Reset_Simulador()* inicializa todas las estructuras del sistema. Esta función establece todos los valores de RAM en cero, inicializa las etiquetas predefinidas, marca todas las páginas como inválidas en la tabla de páginas, vacía completamente la caché estableciendo todos los tags en -1, carga datos de prueba en el área de swap (valores múltiplos de 100), y resetea los puertos de entrada/salida junto con las señales de estado.

Algoritmos de Reemplazo

El algoritmo LRU para la caché se implementó mediante un sistema simple de bits para conjuntos de 2 vías. Cada línea mantiene un bit de estado que indica si es la más recientemente usada (MRU, valor 0) o la menos recientemente usada (LRU, valor 1). Al acceder a una línea, se establece su bit en 0 y el de su compañera en el mismo conjunto en 1. Durante un reemplazo, se selecciona la línea cuyo bit tenga valor 1. Esta implementación es eficiente y suficiente para conjuntos de 2 vías.

Para la paginación, el algoritmo LRU se implementó mediante contadores incrementales. Cada página válida mantiene un contador que se incrementa en cada acceso a memoria. Cuando se accede a una página específica, su contador se resetea a 0. Al necesitar reemplazar una página, el algoritmo busca primero marcos no utilizados; si todos están ocupados, selecciona la página con el contador más alto, indicando que ha sido la menos recientemente accedida. Este método proporciona una aproximación precisa del comportamiento LRU verdadero.

Integración con Excel

El sistema aprovecha la interfaz visual de Excel para proporcionar retroalimentación en tiempo real. Las celdas se organizan en regiones específicas: la tabla de RAM ocupa las filas 3-34 en las columnas O-P; la representación de la caché se ubica en las filas 3-10 con columnas S-U mostrando tag, dato y estado LRU; la tabla de páginas se visualiza en las filas 7-14 con columnas W-Y indicando número de página, marco asignado y bit de validez; el área de swap se encuentra en la columna AC; y las señales de estado se muestran en celdas específicas (X6 para caché, Y7 para MMU). Los puertos de entrada/salida se ubican en las filas 3-4 de la columna W, permitiendo la comunicación con el módulo de CPU.

Módulo de Simulación de CPU

El módulo de CPU implementa un procesador simplificado capaz de ejecutar un conjunto básico de instrucciones en lenguaje ensamblador. El diseño sigue una arquitectura de registros con tres registros de propósito general y un contador de programa, operando sobre la memoria virtual proporcionada por el módulo anterior.

Arquitectura del Procesador

El procesador cuenta con tres registros principales ubicados en celdas específicas de la hoja de cálculo. El Registro1 (celda C8) y el Registro2 (celda G8) funcionan como registros de propósito general para almacenamiento temporal de datos. El Acumulador (celda E22) se reserva exclusivamente para almacenar resultados de operaciones aritméticas. El Contador de Programa (PC, celda J8) mantiene la dirección de la siguiente instrucción a ejecutar, inicializándose en la fila 38 donde comienza el programa.

El sistema incluye puertos de entrada/salida mapeados a celdas específicas. Los puertos de entrada (Entrada1 en C30 y Entrada2 en C31) permiten leer datos desde dispositivos externos o valores predefinidos por el usuario. Los puertos de salida (Salida1 en D30 y Salida2 en D31) almacenan resultados que pueden representar la comunicación con periféricos. Esta separación entre entrada y salida simula el modelo de E/S por puertos utilizado en arquitecturas reales.

Conjunto de Instrucciones

Se implementaron dos instrucciones fundamentales que permiten operaciones básicas de transferencia y cálculo. La instrucción MOVER realiza transferencia de datos entre diferentes ubicaciones del sistema. Su formato es "MOVER origen destino" donde ambos operandos pueden ser registros, puertos de E/S o direcciones de memoria virtual. La implementación distingue entre diferentes tipos de operandos mediante análisis de texto: nombres de registros y puertos se reconocen directamente, mientras que las referencias a memoria virtual utilizan la sintaxis MV[dirección] que es parseada extrayendo el número entre corchetes.

La instrucción SUMAR ejecuta la operación aritmética de suma sobre dos operandos. Su formato es "SUMAR operando1 operando2" y el resultado siempre se almacena en el Acumulador. Los operandos pueden ser cualquier registro o puerto de entrada, pero no se permite el acceso directo a memoria virtual en esta instrucción. Esta limitación simplifica la implementación mientras mantiene la funcionalidad esencial para operaciones aritméticas.

Ciclo de Ejecución

El ciclo de instrucción sigue el modelo clásico fetch-decode-execute implementado en la función *EjecutarInstruccionPaso()*. En la fase de fetch, el sistema lee la instrucción ubicada en la fila indicada por el PC, específicamente en la columna G. La fase de decode analiza el texto de la instrucción dividiéndola en tokens mediante espacios, identificando el código de operación (primera palabra) y los operandos (palabras subsecuentes). La fase de execute invoca la función correspondiente según el tipo de instrucción detectada, pasando los operandos necesarios. Finalmente, el PC se incrementa en 1 para apuntar a la siguiente instrucción.

Durante la ejecución, el sistema proporciona retroalimentación visual resaltando en amarillo la celda de la instrucción actual. Este resaltado se limpia antes de cada nueva instrucción, asegurando que solo la instrucción en ejecución esté marcada. Esta característica facilita el seguimiento del flujo de ejecución durante la depuración o demostración del programa.

Modos de Ejecución

El simulador ofrece dos modos de ejecución distintos. El modo paso a paso, implementado en *EjecutarInstruccionPaso()*, ejecuta una sola instrucción por invocación, permitiendo al usuario observar detalladamente cada cambio en el estado del procesador. Este modo es invaluable para propósitos educativos y depuración, ya que permite analizar el efecto de cada instrucción individual sobre los registros y la memoria.

El modo de ejecución continua, implementado en *EjecutarTodo()*, ejecuta instrucciones secuencialmente hasta encontrar una celda vacía en la columna de instrucciones. Este modo utiliza un bucle que repetidamente invoca *EjecutarInstruccionPaso()* y emplea la función DoEvents de VBA para mantener la interfaz de Excel responsiva durante la ejecución. La condición de terminación se detecta al intentar leer una instrucción y encontrar una celda vacía, momento en el cual el sistema muestra un mensaje indicando la finalización del programa.

Integración con el Sistema de Memoria

La integración entre la CPU y el sistema de memoria se realiza mediante llamadas directas a las funciones *Leer_Memoria()* y *Escribir_Memoria()* cuando la instrucción MOVER utiliza la sintaxis MV[dirección]. Al detectar esta sintaxis mediante análisis del operando (verificando el prefijo "mv["), el sistema extrae el número de dirección, elimina los caracteres no numéricos (corchetes), y convierte el resultado a un valor entero que se pasa como parámetro a las funciones de memoria.

Esta integración activa automáticamente todos los mecanismos del sistema de memoria: la traducción de dirección virtual a física mediante la tabla de páginas, la gestión de page faults con carga desde el área de swap, la búsqueda en caché con política LRU, y la actualización de todas las estructuras de datos y señales visuales. De esta forma, cada instrucción que accede a memoria virtual dispara una simulación completa del comportamiento de la jerarquía de memoria, incluyendo las latencias correspondientes que se visualizan en las celdas de señales.

Inicialización y Reset

La función *InicializarCPU()* prepara el procesador para comenzar la ejecución estableciendo todos los registros en cero y posicionando el PC en la fila 38. Esta función también limpia cualquier resaltado previo de instrucciones, asegurando un estado visual limpio. La función *InicializarTodo()* extiende esta funcionalidad invocando tanto *InicializarCPU()* como *Reset_Simulador()* del módulo de memoria, proporcionando un reinicio completo de todo el sistema simulado. Esta función es esencial al comenzar una nueva sesión de prueba o al cargar un programa diferente.

Validación y Manejo de Errores

El sistema implementa validación básica de instrucciones verificando la sintaxis y la existencia de operandos requeridos. Cuando se detecta una instrucción con número incorrecto de operandos, el sistema muestra un mensaje de error específico indicando el problema y la fila donde ocurrió. Los operandos desconocidos también generan mensajes de error, aunque en estos casos la función devuelve un valor predeterminado de cero para evitar la interrupción completa de la ejecución.

Las celdas vacías en la columna de instrucciones se interpretan como fin de programa en modo de ejecución continua, pero generan advertencias en modo paso a paso. Este comportamiento permite al usuario saber cuándo ha alcanzado el final del programa sin provocar errores críticos. La implementación no incluye manejo de excepciones para errores de memoria (como direcciones fuera de rango), delegando esta responsabilidad al módulo de memoria que sí implementa dichas validaciones.

Traducción de C++ a Ensamblador

Como complemento al simulador desarrollado, se presenta la traducción de un programa simple en C++ a lenguaje ensamblador x86, demostrando la lógica de compilación que fundamenta el diseño de nuestro simulador.

Implentación de Código C++

```cpp
#include <iostream>

// Función principal del programa
int main() {
    // Declaramos dos variables para almacenar los números de entrada
    int numero1;
    int numero2;
    // Declaramos una variable para almacenar el resultado de la suma
    int suma;

    // Pedir al usuario el primer número
    std::cout << "Por favor, introduce el primer numero entero: ";
    // Leer el primer número de la entrada del usuario y guardarlo en numero1
    std::cin >> numero1;

    // Pedir al usuario el segundo número
    std::cout << "Por favor, introduce el segundo numero entero: ";
    // Leer el segundo número de la entrada del usuario y guardarlo en numero2
    std::cin >> numero2;

    // Calcular la suma
    suma = numero1 + numero2;

    // Mostrar el resultado de la suma
    std::cout << "\nLa suma de " << numero1 << " y " << numero2 << " es: " << suma << std::endl;

    // Indicar que el programa ha finalizado correctamente
    return 0;
}
```

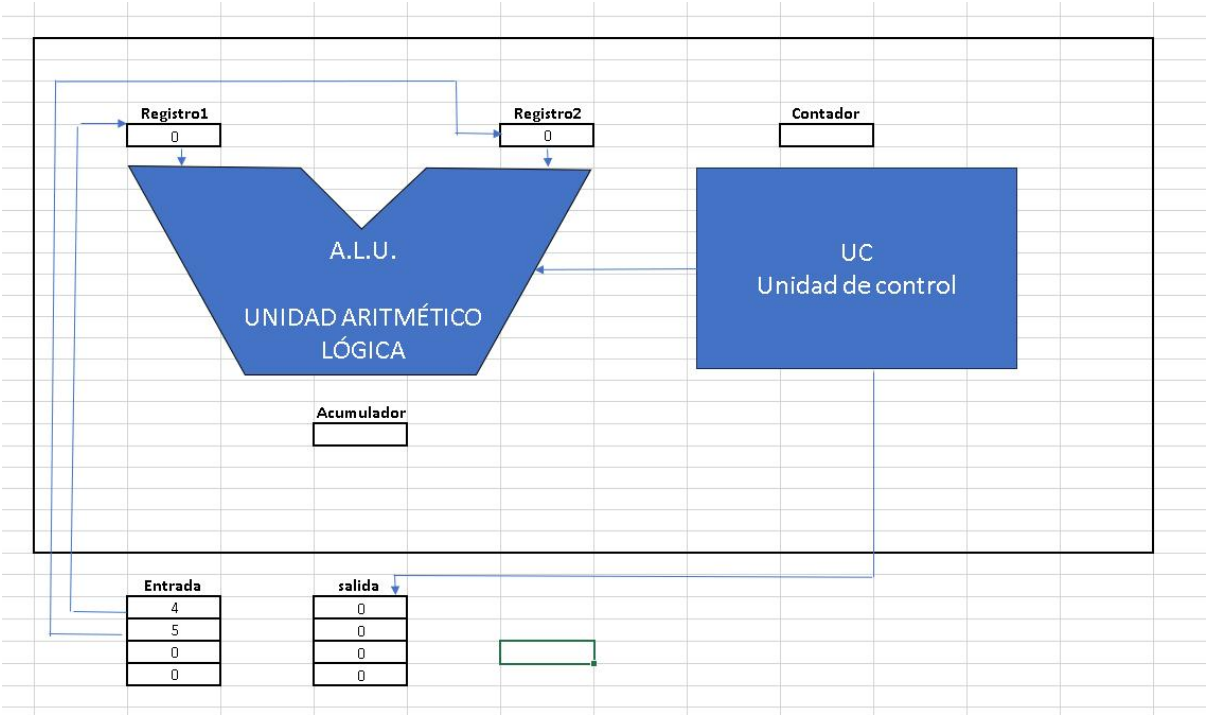
Traducción a lenguaje ensamblador x86

```
section .data
    ; Declaramos las variables y les asignamos valores iniciales (simulando la entrada)
    numero1 dd 10    ; 'dd' define un Double-word (4 bytes, como un 'int' en C++)
    numero2 dd 25
    suma    dd 0     ; Variable para almacenar el resultado

section .text
    global _start

_start:
    ; 1. Cargar el primer número (numero1) en el registro EAX
```

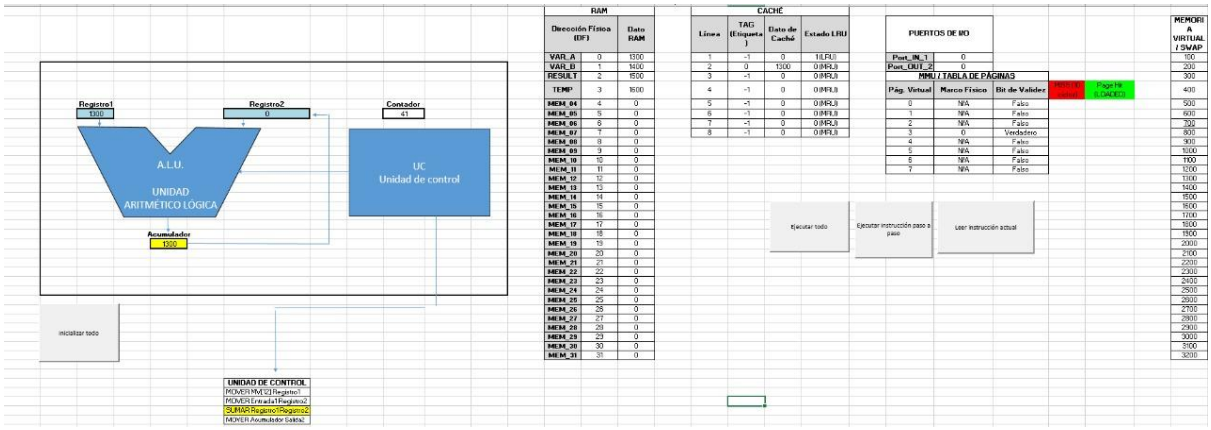
```
    mov eax, [numero1]  ; Mueve el valor de la dirección de memoria 'numero1' a EAX

    ; 2. Sumar el segundo número (numero2) a EAX
    add eax, [numero2]  ; Suma el valor de 'numero2' al valor actual de EAX (EAX = 10 + 25)

    ; 3. Almacenar el resultado (ahora en EAX) en la variable 'suma'
    mov [suma], eax     ; Mueve el valor de EAX a la dirección de memoria 'suma'

    ; El programa aquí terminaría o seguiría con la lógica para imprimir 'suma'.
; (La lógica de salida es muy dependiente del SO y es extensa)
```

DISEÑOS

Diseño de la CPU FINAL



Diseño De la Memoria FINAL



- Los cambios de los registros se ven marcados en colores.
- Los valores que se almacenan se muestran en la RAM, CACHE y la MEMORIA VIRTUAL.
- Y la latencia se ve en los ciclos, mientras menos ciclos menos latencia.

CONCLUSIONES

El desarrollo del simulador de CPU con gestión jerárquica de memoria y visualización del pipeline permitió comprender de forma práctica el funcionamiento interno de los sistemas de procesamiento modernos. A través de la simulación implementada en **Excel con macros VBA**, se logró representar las interacciones entre la **memoria caché**, la **RAM**, la **memoria virtual** y las **fases del ciclo de instrucción**, ofreciendo una herramienta educativa accesible y visualmente comprensible.

Durante el proceso, se evidenció cómo los mecanismos de **reemplazo de caché (LRU)** y de **paginación** influyen en el rendimiento del sistema, demostrando la importancia de la jerarquía de memoria en la reducción de tiempos de acceso. Asimismo, el diseño del **pipeline** permitió analizar la ejecución paralela de instrucciones, resaltando los beneficios del paralelismo en la eficiencia de los procesadores.

La integración de todos estos elementos dentro de un entorno interactivo permitió que conceptos teóricos —como la **unidad de manejo de memoria (MMU)**, los **fallos de página** y las **políticas de escritura (Write-Through)**— pudieran observarse de manera tangible y didáctica. Esto contribuye significativamente al proceso de enseñanza-aprendizaje en el área de **arquitectura de computadores**, al transformar la teoría en una experiencia experimental.

En conclusión, el proyecto no solo cumplió con los objetivos planteados, sino que también demostró la viabilidad de utilizar herramientas comunes como **Microsoft Excel** para crear simulaciones técnicas complejas. Este enfoque fomenta la creatividad, la comprensión del hardware desde una perspectiva de software y la capacidad de representar procesos computacionales con recursos accesibles y pedagógicamente eficaces.

ANEXOS

- YouTube: Unidad Central de Proceso (CPU) - ¿Cómo Funciona?: https://www.youtube.com/watch?v=rPb2im8kWC0

- Hwang, K., & Briggs, F. A. (2017). *Computer Architecture and Parallel Processing*. McGraw-Hill.
- Patterson, D. A., & Hennessy, J. L. (2021). *Computer Organization and Design: The Hardware/Software Interface* (6th ed.). Morgan Kaufmann.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating System Concepts* (10th ed.). Wiley.
- Stallings, W. (2021). *Computer Organization and Architecture: Designing for Performance* (12th ed.). Pearson.
- Tanenbaum, A. S., & Austin, T. (2013). *Structured Computer Organization* (6th ed.). Pearson.