UNIVERSIDAD
CATÓLICA
BOLIVIANA
SANTA CRUZ

SIMULADOR DE ARQUITECTURA X86

Simulador de una arquitectura de computadoras tipo x86

Docente: Ing. Paulo Cesar Loayza Carrasco

Estudiantes: Nicole Lozada León

Dariana Pol Aramayo

Krishna Ariany Lopez Melgar

Santa Cruz, Bolivia
2025

Resumen

El presente trabajo de investigación detalla el diseño y desarrollo de un simulador educativo para una arquitectura de computadoras de tipo x86. El objetivo principal es crear una herramienta didáctica e interactiva que facilite la comprensión conceptual de los componentes y procesos fundamentales de un sistema de cómputo moderno. Para lograrlo, se prioriza la abstracción de conceptos complejos y la visualización de procesos internos, utilizando un enfoque de simulación que modela el comportamiento observable del sistema.

La metodología se basó en el desarrollo de una plataforma unificada y modular implementada en Visual Basic for Applications (VBA) sobre Microsoft Excel, garantizando su accesibilidad. El proyecto abarcó la creación de modelos funcionales para los componentes centrales de la CPU: se implementó un conjunto completo de registros x86 de 32 bits , una Unidad Aritmético-Lógica (ALU) capaz de realizar operaciones aritméticas y lógicas con actualización de flags , y una Unidad de Control (CU) que orquesta el ciclo de instrucción Fetch-Decode-Execute.

Tabla de contenidos

1. Introducción

En el ámbito de la arquitectura de computadoras, la comprensión de los principios fundamentales que rigen el funcionamiento de los procesadores representa un desafío educativo significativo. Los conceptos abstractos como el pipeline de instrucciones, la jerarquía de memoria y la ejecución especulativa suelen resultar particularmente complejos para los estudiantes, quienes se enfrentan a la dificultad de visualizar procesos que ocurren a nivel microscópico dentro del hardware. Esta problemática motivó el desarrollo del presente proyecto: un simulador didáctico de arquitectura x86 que busca transformar conceptos teóricos en experiencias interactivas y observables.

La arquitectura x86, siendo el estándar predominante en computadoras personales y servidores durante décadas, ofrece un marco ideal para el estudio de los principios arquitectónicos modernos. Sin embargo, su complejidad inherente, con modos de direccionamiento variados, registros especializados y mecanismos de ejecución segmentada, requiere de herramientas que permitan desglosar progresivamente cada componente.

Este simulador surge como respuesta a esta necesidad educativa, implementando un modelo funcional que replica el comportamiento de un procesador x86 simplificado pero estructuralmente fiel. Desarrollado en Visual Basic for Applications (VBA) sobre Microsoft Excel, la plataforma aprovecha la facilidad de estas herramientas para garantizar su accesibilidad, mientras que su diseño modular permite explorar secuencialmente cada aspecto de la arquitectura. El sistema integra representaciones visuales de los registros de propósito general y de segmento, una unidad aritmético-lógica (ALU) operativa con actualización de flags, y una unidad de control que orquesta el ciclo fetch-decode-execute característico del modelo von Neumann.

La innovación central del proyecto reside en su capacidad para simular concurrentemente múltiples aspectos arquitectónicos: desde la gestión básica de memoria RAM hasta políticas avanzadas de caché mediante mapeo directo y asociativo por conjuntos con reemplazo LRU, pasando por la implementación de un pipeline de cinco etapas con detección automática de riesgos de datos. Cada módulo ha sido diseñado con un enfoque pedagógico progresivo, permitiendo a los usuarios observar cómo las instrucciones en lenguaje ensamblador se transforman en señales de control, operaciones aritméticas y transferencias de datos, actualizando en tiempo real el estado del sistema.

El presente trabajo constituye un recurso para la experimentación controlada, permitiendo modificar parámetros operativos, introducir código en ensamblador personalizado y observar las consecuencias de diferentes escenarios de ejecución. El simulador no solo facilita la comprensión conceptual sino que fomenta el desarrollo de competencias analíticas esenciales para cualquier profesional en el área de sistemas computacionales.

El resultado es una plataforma que no solo ilustra el funcionamiento interno de un computador, sino que inspira una comprensión más profunda y duradera de los principios que hacen posible la computación moderna.

2. Marco Teórico

2.1. Arquitectura x86

La arquitectura x86 es una arquitectura de conjunto de instrucciones (ISA) desarrollada inicialmente por Intel, caracterizada por su compatibilidad hacia atrás y su uso masivo en computadoras personales y servidores. Fue introducida con el procesador Intel 8086 en 1978 y ha evolucionado para incluir modos de 32 bits (x86) y 64 bits (x86-64). Esta arquitectura soporta instrucciones complejas, modos de direccionamiento variados y es base para la mayoría de sistemas operativos actuales. Según Microsoft (Windows Drivers Documentation), x86 se destaca por su versatilidad y amplia adopción en la industria. (Microsoft, 2025)

2.2. Componentes de la CPU

2.2.3 Unidad de Control

La Unidad de Control es la encargada de dirigir y coordinar todas las operaciones de la CPU, interpretando las instrucciones del programa y enviando señales de control a las demás unidades. Según Amazon Web Services (AWS), la CU gestiona el flujo de datos dentro del CPU y entre el CPU y la memoria. (Amazon, n.d.)

2.2.4 Unidad Aritmético-Lógica (ALU)

La ALU es un circuito combinacional, y es la parte del procesador que realiza operaciones aritméticas, lógicas y otras operaciones relacionadas necesarias. A veces hay un componente separado, shifter , que se utiliza para realizar las operaciones de desplazamiento en los elementos de datos . El programador de abstracción 2 generalmente considera que las actividades del shifter son parte de la ALU. El ábaco, la máquina de Blaise Pascal y otros antecesores de la computadora eran realmente solo ALU, y en su mayor parte los demás componentes de los sistemas informáticos modernos simplemente están ahí para contener datos o transferirlos a la ALU. Por lo tanto, la ALU podría considerarse el centro del sistema informático . (Joe Grimes, 2003)

2.2.5 Registros (Registers).

Los registros (Registers) son unidades de almacenamiento pequeñas y de alta velocidad ubicadas dentro de la Unidad Central de Procesamiento (CPU) que se utilizan para mantener temporalmente datos e instrucciones durante el procesamiento. Su función principal es proporcionar acceso rápido a la información necesaria para la ejecución de instrucciones, minimizando el tiempo de espera en comparación con acceder a la memoria principal. Este acceso inmediato a datos opera dentro del ciclo fundamental Fetch-Decode-Execute, permitiendo a la CPU ejecutar instrucciones eficientemente. (Dev.to, 2024)

2.3. Jerarquía de Memoria:

2.3.1 Memoria RAM, Caché y Memoria Virtual.

La Memoria RAM (Random Access Memory) es una memoria volátil de alta velocidad que almacena temporalmente datos e instrucciones que la CPU necesita durante la ejecución de programas. Según un análisis académico de la Universidad Autónoma de México (UNAM), la RAM permite un acceso rápido y aleatorio a la información, aunque pierde todos sus datos

una vez que el equipo se apaga; este equilibrio la hace fundamental para la operación de los sistemas informáticos modernos. (Coban Alta Verapaz, 2013)

La Memoria Caché es una memoria pequeña pero extremadamente rápida, ubicada más cerca del procesador que la RAM, usada para almacenar copias de las instrucciones y datos más utilizados. Así, reduce la latencia de acceso al acelerar la recuperación de información; este concepto está ampliamente descrito en estudios universitarios sobre jerarquía de memoria. (Coban Alta Verapaz, 2013)

La Memoria Virtual es una técnica que extiende la memoria física disponible mediante la utilización del espacio en disco duro o SSD como si fuera memoria adicional. Funciona mediante la creación de áreas llamadas páginas de intercambio que almacenan temporalmente datos que no se están usando activamente en la RAM, permitiendo así la ejecución de programas más grandes que la memoria física instalada. (Coban Alta Verapaz, 2013)

2.3.2 Políticas de reemplazo de caché (LRU).

Las políticas de reemplazo de caché son estrategias utilizadas para determinar qué datos deben ser eliminados de la memoria caché cuando esta está llena y se requiere espacio para almacenar nueva información.

Una de las políticas más reconocidas y utilizadas es la LRU (Least Recently Used, o menos recientemente usada). Esta política elimina el bloque de datos que lleva más tiempo sin ser utilizado, partiendo del supuesto de que los datos que no se han usado recientemente tienen menos probabilidades de ser requeridos en un futuro inmediato. De acuerdo con documentos académicos y material universitario, la política LRU busca optimizar la eficiencia del sistema manteniendo en caché la información más relevante y frecuente para la CPU, ayudando a mejorar el rendimiento general del sistema. (Carlos E. Quesada Sánchez & Esteban Meneses, 2006)

2.4. Modelos de Arquitectura

2.4.1 Diferencias entre Von Neumann y Harvard.

La arquitectura Von Neumann se caracteriza por tener una memoria única que almacena tanto las instrucciones del programa como los datos. Según la Universidad Nacional Autónoma de México (UNAM) y otras fuentes académicas, esta memoria compartida es accesible por un único sistema de buses, lo cual implica que la CPU no puede leer una instrucción y acceder a datos simultáneamente, generando potencialmente un cuello de botella en el rendimiento (conocido como el problema del bus de Von Neumann). Esta arquitectura es la base de la mayoría de los ordenadores personales y servidores modernos, incluido el microprocesador x86. (Charles W. Kann III, 2016)

En contraste, la arquitectura Harvard posee memorias separadas físicamente para datos e instrucciones, cada una con su propio sistema de buses. Esta característica permite que la CPU acceder simultáneamente a las instrucciones y a los datos, mejorando significativamente

el rendimiento en tareas que requieren accesos concurrentes. Harvard es común en sistemas embebidos, procesadores de señal digital (DSP) y microcontroladores, donde la eficiencia y velocidad en el acceso paralelo son cruciales. Esta arquitectura fue propuesta inicialmente en la Universidad de Harvard en la década de 1940 y es especialmente útil en aplicaciones donde se requiere procesamiento en tiempo real. (Charles W. Kann III, 2016)

2.5. Ejecución de Programas

2.5.1 El Ciclo de Instrucción.

El ciclo de instrucción en arquitectura de computadoras se define como el período o secuencia de acciones que la unidad central de proceso (CPU) realiza para ejecutar una instrucción de lenguaje de máquina. Este ciclo comprende varias etapas clave, generalmente buscar la instrucción en la memoria principal, decodificarla y ejecutarla, siendo fundamental para el funcionamiento y rendimiento del procesador.

3. Diseño y Desarrollo

3.1 Implementación de la Unidad de Control (CU), ALU y Registros

3.1.1 Arquitectura de Registros

Se implementó un conjunto completo de registros x86 de 32 bits utilizando variables nombradas individuales para maximizar la legibilidad y fidelidad con la arquitectura real:

```
VBA

' Registros de propósito general
Public EAX As Long, EBX As Long, ECX As Long, EDX As Long
' Registros de segmento
Public CS As Integer, DS As Integer, SS As Integer, ES As Integer
' Registros de puntero e índice
Public EIP As Long, ESP As Long, EBP As Long, ESI As Long, EDI As Long
' Flags de estado
Public ZF As Boolean, SF As Boolean, CF As Boolean, OF As Boolean, PF As Boolean, AF As Boolean

```

Esta aproximación permite una representación visual clara durante la depuración y se alinea directamente con la nomenclatura estándar de ensamblador x86.

3.1.2 Unidad Aritmético-Lógica (ALU)

La ALU se implementó mediante un módulo VBA que contiene las siguientes operaciones:

Operaciones Aritméticas:

- *EjecutarADD*: Suma con actualización completa de flags
- *EjecutarSUB*: Resta con manejo de préstamo (borrow)

- *EjecutarMUL/IMUL:* Multiplicación sin/con signo
- *EjecutarDIV/IDIV:* División con protección contra división por cero

Operaciones Lógicas:

- *EjecutarAND/OR/XOR:* Operaciones bit a bit
- *EjecutarNOT:* Complemento a uno
- *EjecutarTEST:* AND sin almacenamiento (solo flags)

Características de la ALU:

- Actualización automática de los 6 flags principales (ZF, SF, CF, OF, PF, AF)
- Soporte para operandos inmediatos y de registro
- Manejo de números hexadecimales y decimales
- Detección de overflow y underflow

3.1.3 Unidad de Control (CU)

La Unidad de Control se distribuye entre dos módulos principales:

Módulo Parser (`ModuloParser.bas`):

- Interpreta instrucciones en lenguaje ensamblador
- Descompone instrucciones en opcode y operandos
- Valida la sintaxis antes de la ejecución
- Soporta 20+ instrucciones x86 comunes

Módulo Ciclo (`ModuloCiclo.bas`):

- Coordina el flujo de ejecución de instrucciones
- Gestiona el registro EIP (Instruction Pointer)
- Controla la secuencia Fetch-Decode-Execute

3.2 Funcionamiento del Ciclo de Instrucción

3.2.1 Arquitectura del Ciclo

El ciclo de instrucción implementa el modelo clásico de von Neumann con cuatro etapas:

```
Fetch → Decode → Execute → Update EIP
```

3.2.2 Etapas Detalladas

1. **Fetch (Búsqueda):**

```
VBA

Private Sub Fetch()
    If EIP < MEM_SIZE Then
        instruccionActual = Memoria(EIP)
        EIP = EIP + 1
    End If
End Sub
```

- Lee la instrucción actual desde memoria en la dirección EIP
- Incrementa automáticamente el contador de programa

2. **Decode (Decodificación):**

```
VBA

Private Sub Decode()
    partes = Split(instruccionActual, " ")
    opcode = partes(0)
    operandos = partes(1) ' Si existe
End Sub
```

- Separa el opcode de los operandos
- Identifica el tipo de instrucción y sus parámetros
- Prepara los registros para la ejecución

3. Execute (Ejecución):

- Ejecuta la operación correspondiente en la ALU
- Actualiza registros y memoria según la instrucción
- Calcula y establece los flags de estado

4. Update EIP (Actualización):

- Verifica límites de memoria
- Prepara el ciclo para la siguiente instrucción

3.2.3 Flujo de Ejecución Completo

El método EjecutarCiclo() orquesta todo el proceso:

```
VBA

Public Sub EjecutarCiclo()
    Fetch       ' Obtener instrucción
```

```
    If Len(instruccionActual) > 0 Then
        Decode   ' Interpretar instrucción
        Execute  ' Ejecutar operación
    End If
    ActualizarEIP ' Preparar siguiente instrucción
End Sub
```

3.3 Simulador de Memoria RAM - Diseño Conceptual

3.3.1 Marco Teórico

Fundamento Conceptual: Implementación del modelo de memoria principal basado en la arquitectura von Neumann, simulando la separación física entre datos y código.

3.3.2 Diseño Arquitectónico

- Modelo de Datos: Tres arrays paralelos que representan:
    - Capa física (RAM): Almacenamiento real en bytes
    - Capa de visualización (RAM_Display): Representación hexadecimal
    - Capa de control (RAM_Loaded): Gestión de estado de carga
- Organización de Memoria:
    - Segmentación clásica: .data (0x00-0x7F) para variables, .text (0x80-0xFF) para código
    - Grid 16×16 que emula la disposición física de chips de memoria

3.3.3 Mecanismo de Carga Dinámica

Diseño Innovador:

- Sistema de carga bajo demanda que simula el comportamiento real de sistemas operativos
- Evita la carga completa inicial, replicando el fetching de instrucciones durante ejecución
- Bandera RAM_Loaded previene sobreescritura y permite tracking de uso

3.4 Simulador de Memoria Caché - Diseño Conceptual

3.4.1 Fundamentos Teóricos

Jerarquía de Memoria: Implementación de dos políticas de caché para análisis comparativo:

- Mapeo Directo: Simplicidad y velocidad predictiva
- Asociativo por Conjuntos: Balance entre flexibilidad y complejidad

3.4.2 Arquitectura de Caché Dual

Diseño Comparativo:

- Caché 1 (Mapeo Directo):
    - 16 líneas × 4 bytes = 64B total
    - Algoritmo: índice = (dirección / tamaño_línea) % número_líneas
- Caché 2 (2-Way Set Associative):
    - 32 conjuntos × 2 vías × 16 bytes = 1KB total
    - Política LRU: Seguimiento de uso por contador temporal

3.4.3 Sistema de Monitoreo Integral

Diseño de Observabilidad:

- 5 perspectivas complementarias:
    - Vista estado actual (Cache)
    - Vista memoria principal (RAM)
    - Métricas rendimiento (Estadísticas)
    - Traza temporal (LogCaché)
    - Estado interno detallado (EstadoCaché)

3.5 Simulador de Memoria Virtual - Diseño Conceptual

3.5.1 Modelo de Sistema Completo

Arquitectura Integrada: Simulación de un microprocesador educativo con:

- Espacio de direccionamiento unificado de 256 bytes
- Conjunto de registros propósito general (AX, BX, CX, DX)
- Registros de segmento y propósito especial (SI, DI, BP, SP)
- Unidad de ejecución con pipeline simplificado

3.5.2 Diseño de Gestión de Memoria

Segmentación Lógica:

- INSTR: Segmento de código (direcciones bajas)
- DATA: Segmento de datos (área media)
- STACK: Pila del sistema (32 posiciones superiores)
- FREE: Espacio no asignado

3.5.3 Motor de Ejecución

Diseño de Ciclo de Instrucción:

1. Fetch: `VirtualMemory(CurrentInstructionVM)`
2. Decode: Análisis sintáctico en `ParseAndExecuteVM`
3. Execute: Ejecución semántica según opcode
4. Update: Actualización estado del sistema

4. Pipeline

4.1 Simulador de Pipeline - Diseño Conceptual

4.1.1 Modelo de Procesamiento Segmentado

Arquitectura Pipeline Clásica: Implementación de las 5 etapas estándar:

- IF (Instruction Fetch): Búsqueda desde memoria
- ID (Instruction Decode): Decodificación y preparación operandos
- EX (Execute): Ejecución en ALU
- MEM (Memory Access): Acceso a datos en memoria
- WB (Write Back): Almacenamiento resultados

4.1.2 Mecanismo de Paradas (Stalls)

Diseño de Gestión de Riesgos:

- Detección automática de dependencias RAW (Read After Write)
- Propagación de burbujas en el pipeline
- Sistema visual de indicación de paradas

4.1.3 Visualización de Paralelismo

Diseño Instruccional:

- Esquema de colores único por instrucción
- Representación temporal en eje Y vs etapas en eje X
- Indicadores de progreso y completado

5. Diseño

5.1 Integración y Coherencia del Diseño

5.1.1 Progresión Pedagógica

Diseño Curricular Integrado:

1. Nivel Memoria: Organización física (RAM)
2. Nivel Aceleración: Jerarquía y caching
3. Nivel Sistema: Memoria virtual y ejecución
4. Nivel Paralelismo: Pipeline y optimización

5.1.2 Plataforma Unificada

Diseño Técnico:

- Implementación en VBA/Excel para accesibilidad
- Interfaz consistente entre módulos
- Sistema de control unificado (paso a paso vs ejecución completa)

5.1.3 Simulación vs Emulación

Enfoque de Diseño:

- Simulación: Comportamiento observable sin implementación real
- Modelado Educativo: Énfasis en conceptos sobre precisión
- Balance: Exactitud suficiente para aprendizaje conceptual

5.2 Innovaciones en el Diseño

5.2.1 Abstracciones Educativas

- Visualización de estados internos normalmente ocultos
- Representación de conceptos abstractos mediante metáforas visuales
- Control granular sobre la ejecución para observación detallada

5.2.2 Escalabilidad Conceptual

- Diseño modular que permite expansión futura
- Interfaces consistentes para nuevos componentes
- Balance entre complejidad y comprensibilidad

6. Sistema de Entrada y Salida (I/O)

6.1 Unidad de Entrada (Input Unit)

La Unidad de Entrada comprende los dispositivos encargados de transformar señales físicas externas en datos digitales procesables por la CPU. Como ejemplo tenemos al teclado,

funciona mediante una matriz de interruptores; al presionar una tecla, el controlador del teclado genera un código de escaneo (Scan Code) que se envía a la CPU mediante una interrupción.

En el entorno del proyecto, la hoja denominada "INPUT" emula el periférico de entrada, específicamente un Teclado. La celda designada para la escritura del usuario representa el buffer de hardware del dispositivo.

Un aspecto crítico de la simulación es la representación de la interfaz de control. El botón "Actualizar Vista" y su código VBA subyacente actúan como el Controlador de E/S. En una arquitectura real, el periférico no escribe directamente en la RAM. En su lugar, el controlador:

1. Realiza una operación de sondeo (polling) o responde a una interrupción para capturar el dato "crudo" (texto en la celda).
2. Adapta la señal (convierte a formato manejable).
3. Coloca el dato en el Bus de Datos para su transporte hacia la memoria principal, sincronizando la velocidad entre el usuario (lento) y el sistema (rápido).

6.2 Unidad de Salida (Output Unit)

La Unidad de Salida convierte la información procesada en representaciones perceptibles para el usuario.

1. Monitor: Dispositivo principal de visualización que interpreta datos de la memoria de video para controlar la intensidad y color de los píxeles en una matriz (CRT, LCD, OLED).
2. Impresora: Transfiere datos digitales a un medio físico (papel) mediante inyección de tinta o láser, utilizando un buffer interno para gestionar la cola de impresión.
3. Altavoz: Convierte señales digitales en ondas sonoras analógicas mediante un convertidor Digital-Analógico (DAC) que hace vibrar una membrana.

La simulación se lleva a cabo en la hoja "OUTPUT", donde se distinguen dos componentes arquitectónicos vitales:

1. Buffer de Video (VRAM): Representado por el rango de celdas beige (denominado BufferVRAM). Este componente ilustra la necesidad de una memoria intermedia dedicada. Dado que la CPU opera a velocidades mucho mayores que el refresco de pantalla, los datos se escriben primero en esta VRAM, desacoplando el procesamiento de la visualización.
2. Pantalla Virtual: El rango de celdas negras (ScreenDisplay) simula la matriz física del monitor.
3. Controlador de Video: El botón "Mostrar en Pantalla" ejecuta la lógica del controlador de video. Este algoritmo lee secuencialmente la VRAM y "renderiza" los caracteres en la pantalla virtual, simulando el barrido de un monitor real.

6.3 Flujo de Datos y Buses del Sistema (Data Flow)

Para gestionar la comunicación entre la CPU, la Memoria y las unidades de E/S, el simulador implementa una representación visual del ciclo de bus en la hoja "MEMORIA_IO". Este proceso demuestra la jerarquía y sincronización de las señales:

1. Bus de Direcciones (Address Bus - Visualización Verde): Este bus es unidireccional (CPU → Memoria/IO). En la simulación, se ilumina primero en color verde, indicando que la CPU ha seleccionado una dirección física específica (ej. 0x00) donde se realizará la operación. Esto asegura que ningún otro componente interfiera en la transacción.
2. Bus de Control (Control Bus - Visualización Roja): Una vez establecida la dirección, se activa el Bus de Control (visualizado en rojo). Este transmite la orden específica de la Unidad de Control; en el caso de la entrada de datos, se activa la señal de escritura (WRITE = 1), instruyendo a la memoria a prepararse para almacenar información.
3. Bus de Datos (Data Bus - Visualización Azul): Finalmente, se activa el Bus de Datos bidireccional (visualizado en azul). Es el encargado de transportar la carga útil (payload), en este caso, la cadena de caracteres proveniente del controlador de entrada.
4. Almacenamiento Matricial en RAM: Como culminación del flujo, los datos no se almacenan como un bloque monolítico, sino que se distribuyen celda por celda en la matriz de memoria. Esto simula el direccionamiento byte a byte real de la arquitectura x86, donde cada carácter ASCII ocupa una dirección de memoria única y consecutiva.

7. Conclusiones

El proyecto ha culminado exitosamente en el desarrollo de un simulador integral para la arquitectura x86, implementado de manera accesible en Visual Basic for Applications (VBA) sobre Microsoft Excel. La herramienta aborda eficazmente el desafío de enseñar conceptos abstractos de la arquitectura de computadoras, transformándolos en experiencias interactivas y visualmente comprensibles.

El simulador no sólo replica los componentes fundamentales de una CPU, sino que también modela procesos complejos, permitiendo a los estudiantes experimentar de manera controlada y progresiva.

Logros Clave del Proyecto:

- *Modelado Fiel de la CPU:* Se implementaron con éxito los componentes centrales de un procesador x86, incluyendo un conjunto completo de registros de 32 bits, una Unidad Aritmético-Lógica (ALU) funcional con actualización de flags y una Unidad de Control (CU) que orquesta el ciclo de instrucción.
- *Simulación de la Jerarquía de Memoria:* El proyecto abarca de forma completa la jerarquía de memoria, con módulos dedicados para la memoria RAM, una simulación comparativa de memoria caché (con mapeo directo y asociativo por conjuntos usando política LRU) y un sistema de memoria virtual.
- *Implementación de Conceptos Avanzados:* Se desarrolló un simulador de pipeline de cinco etapas que incluye la detección automática de riesgos de datos (RAW) y la

inserción de paradas (stalls), uno de los aspectos más complejos de la arquitectura moderna.

En definitiva, este simulador se constituye como un valioso recurso didáctico que cierra la brecha entre la teoría y la práctica. Al permitir a los estudiantes observar directamente cómo el código ensamblador se traduce en operaciones a nivel de hardware, la plataforma facilita una comprensión más profunda y duradera de los principios que rigen la computación moderna.

Referencias

Amazon. (n.d.). *¿Qué es una CPU? Explicación de la unidad central de procesamiento*. AWS. Retrieved October 6, 2025, from https://aws.amazon.com/es/what-is/cpu/

Carlos E. Quesada Sánchez, & Esteban Meneses. (2006). Políticas de reemplazo en la caché de web. 14.

Charles W. Kann III. (2016). *Implementación de una CPU de una dirección en Logisim (Kann)*. The Cupola. https://cupola.gettysburg.edu/oer/3/

Coban Alta Verapaz. (2013, June 6). *Memorias Ram | Arquitectura de Computadoras*. Arquitectura de Computadoras. Retrieved October 6, 2025, from https://arquitecturadecomputadora.wordpress.com/2013/06/06/memorias-ram/

Dev.to. (2024, September 20). *What are CPU registers*. DEV Community. Retrieved October 6, 2025, from https://dev.to/aamhamdi/what-are-cpu-registers-4275

HardZone. (2025, June 25). *Ciclo de instrucción en CPU: ¿qué es Fetch, Decode y Execute?* Hard Zone. Retrieved October 6, 2025, from https://hardzone.es/tutoriales/rendimiento/ciclo-instruccion-cpu/

Joe Grimes. (2003). *Computer Architecture*. Encyclopedia of Physical Science and Technology (Third Edition),.

Microsoft. (2025, January 6). *Arquitectura x86 - Windows drivers*. Microsoft Learn. Retrieved October 6, 2025, from https://learn.microsoft.com/es-es/windows-hardware/drivers/debugger/x86-architecture