UNIVERSIDAD NACIONAL ABIERTA
VICERRECTORADO ACADÉMICO
ÁREA: INGENIERÍA

TRABAJO PRÁCTICO

ASIGNATURA: Arquitectura del Computador

CÓDIGO: 333

FECHA DE PUBLICACIÓN EN BLOG DEL SUBPROGRAMA DISEÑO ACADÉMICO: 13/07/24

FECHA DE DEVOLUCIÓN POR PARTE DEL ESTUDIANTE: El estudiante contará hasta el día **02/11/2024 sin prórroga,** para su realización y envío.

NOMBRE DEL ESTUDIANTE: César Torres Chang

CÉDULA DE IDENTIDAD: V-20.246.713

CORREO ELECTRÓNICO DEL ESTUDIANTE: cesarchang23@gmail.com

TELÉFONO:

CENTRO LOCAL: Metropolitano CARRERA: 236

LAPSO ACADÉMICO: 2024-2

NUMERO DE ORIGINALES:

FIRMA DEL ESTUDIANTE:

UTILICE ESTA PÁGINA COMO CARÁTULA DE SU TRABAJO

RESULTADOS DE CORRECCIÓN

OBJ N°		I.2	III.2
0:NL	1:L		

INDICE

Tareas Realizadas en el Emulador de Arquitectura del Computador EMU8086

Unidad de Control

Uso Intensivo de la UC

Unidad Aritmético-Lógica (ALU)

Operaciones Aritméticas y Lógica Usando la ALU

Memoria

Cargar y Almacenar Datos en Diferentes Tipos de Memoria (RAM y ROM)

Bus de Datos

Transferencia de Datos entre la Memoria y la CPU a través del Bus de Datos

Ciclo de Instrucción

Ciclo Completo de Instrucción (fetch – decode – execute)

Registros

Realizar Operaciones Rápidas y Eficientes

Interconexiones y Módulo de Entrada/Salida (E/S)

Comunicación entre la CPU y un Dispositivo E/S

Almacenamiento de Programas

Almacenamiento en Memoria Principal

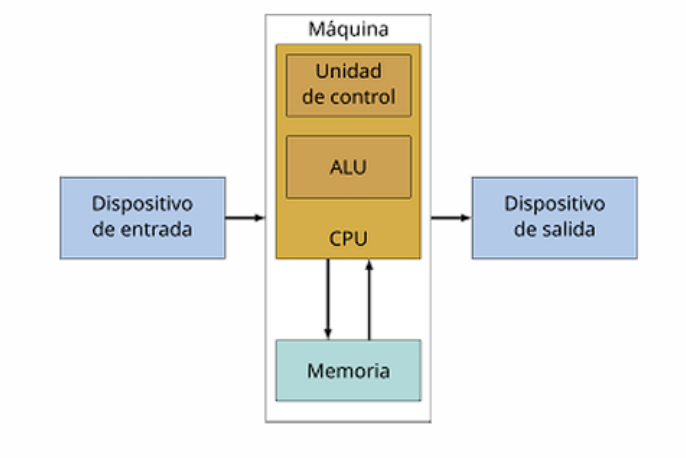
INTRODUCCIÓN

El modelo de Von Neumann, propuesto por el matemático y físico John von Neumann en la década de 1940, establece los principios fundamentales sobre los cuales se construyen las arquitecturas de computadoras modernas. Este modelo postula que tanto las instrucciones del programa como los datos necesarios para su ejecución son almacenados en la misma memoria, creando una estructura unificada que permite un acceso eficiente y simplificado a los recursos del sistema. Esta característica fundamental no solo reduce la complejidad del diseño de hardware, sino que también optimiza el flujo de información entre la CPU y la memoria, facilitando el ciclo de ejecución de instrucciones.

A lo largo de este trabajo, se abordarán varios aspectos críticos del modelo de von Neumann, comenzando con la función de la Unidad de Control (UC), que coordina el flujo de instrucciones y asegura que cada operación se ejecute en el orden correcto. Además, se discutirá la importancia de los registros en la ejecución rápida de operaciones aritméticas y lógicas, así como la relevancia de la memoria RAM y ROM en términos de velocidad y accesibilidad de los datos. También se examinarán los patrones de transferencia de datos entre la CPU y la memoria, y cómo estos afectan el rendimiento general del sistema. A medida que se profundiza en estos temas, se hará evidente cómo el modelo de Von Neumann continúa siendo una guía esencial en el diseño y funcionamiento de las computadoras actuales, influyendo en la manera en que se estructuran y operan los sistemas informáticos en la actualidad.

PRIMERA PARTE

Modelo De Von Neumann



¿Qué es?

La arquitectura de Von Neumann es un diseño fundamental que define cómo una computadora procesa y maneja la información. En este modelo, los componentes principales son la CPU, la memoria, y los dispositivos de entrada y salida. Cada uno cumple funciones específicas que contribuyen al funcionamiento eficiente de la computadora.

Componentes del modelo

Unidad Central de Procesamiento (CPU)

Es el cerebro de la computadora y ejecuta las instrucciones de los programas. En su interior se encuentran dos unidades clave:

- ***Unidad de Control (UC)***: Interpreta las instrucciones y coordina los demás componentes para que cumplan sus funciones.
- ***Unidad Aritmético-Lógica (ALU)***: realiza cálculos matemáticos y operaciones lógicas requeridas para el procesamiento.

Memoria

Es el espacio donde se almacenan tanto las instrucciones de los programas como los datos necesarios para que estos se ejecuten. En el modelo de Von Neumann, la misma memoria se utiliza para ambos propósitos.

Dispositivos de entrada y salida (E/S)

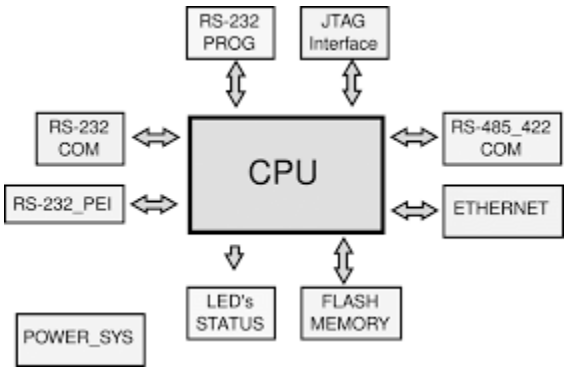
También llamados *dispositivos* periféricos, permiten la interacción con el exterior. Los dispositivos de entrada, como teclados y sensores, proporcionan datos a la CPU, mientras que los de salida, como pantallas e impresoras, muestran resultados.

Protocolos de Comunicación

Los protocolos de comunicación son conjuntos de reglas que definen cómo se envían y reciben datos entre la CPU y los periféricos, estableciendo normas para la velocidad de transmisión, formato de datos y señalización. La estandarización es esencial para asegurar que diferentes dispositivos puedan interactuar sin conflictos y se minimicen los errores.

Figura 1

Diagrama que ilustra el modo de comunicación de los periféricos asociados al microcontrolador.



Fuente: Carralero, K., Cedeño, A., Espí, R., Marrero, I., Moreno, A., Parra, Y. (2016). Pasarela de datos natural sobre estándar TETRA.

Establecimiento de protocolos

Los protocolos de comunicación entre la CPU y un periférico son esenciales para asegurar que ambos puedan intercambiar datos de manera eficiente y sin errores, estos se realizan de la siguiente manera:

1. **Defininición del Protocolo**: se establece un conjunto de reglas y estándares que determinan cómo se deben estructurar, codificar, transmitir, recibir y procesar los datos. Esto incluye el formato de los datos, la secuencia de transmisión y las respuestas esperadas.
2. **Interfaz Física**: la conexión física entre la CPU y el periférico se realiza a través de interfaces como USB, PCI, o serial. Estas interfaces definen los pines y señales eléctricas necesarias para la comunicación.
3. **Controladores de Dispositivos**: son programas que actúan como intermediarios entre la CPU y el periférico. Gestionan la comunicación siguiendo el protocolo definido, asegurando que los datos se envíen y reciban correctamente.

4. **Interrupciones y Acceso Directo a Memoria (DMA)**: las interrupciones permiten que el periférico notifique a la CPU cuando necesita atención, mientras que el DMA permite la transferencia de datos sin intervención constante de la CPU.
5. **Sincronización y Control de Flujo**: se implementan mecanismos para sincronizar la transmisión de datos y controlar el flujo, evitando que se pierdan datos o se produzcan errores de sincronización.
6. **Protocolos de Alto Nivel**: además de los protocolos de bajo nivel (físicos y de enlace), se utilizan protocolos de alto nivel como TCP/IP, HTTP, o FTP para la transferencia de datos más complejos y estructurados.

Importancia de establecer protocolos estándar

Definir un protocolo estándar en las interfaces de comunicación entre la CPU y los periféricos es crucial por varias razones:

- **Compatibilidad**: los protocolos estándar aseguran que diferentes dispositivos y componentes puedan comunicarse entre sí sin problemas, independientemente del fabricante. Esto facilita la integración de nuevos periféricos en sistemas existentes.
- **Interoperabilidad**: permite que dispositivos de diferentes fabricantes funcionen juntos sin necesidad de adaptadores o modificaciones especiales. Esto es esencial en entornos donde se utilizan múltiples dispositivos y sistemas.
- **Eficiencia**: los protocolos estándar optimizan la transferencia de datos, reduciendo la latencia y mejorando el rendimiento general del sistema. Esto es especialmente importante en aplicaciones que requieren alta velocidad y baja latencia, como el procesamiento de video o juegos.
- **Facilidad de Desarrollo**: los desarrolladores pueden diseñar hardware y software sabiendo que sus productos serán compatibles con otros dispositivos que siguen el mismo protocolo. Esto reduce el tiempo y los costos de desarrollo.
- **Seguridad**: los protocolos estándar suelen incluir medidas de seguridad para proteger los datos durante la transmisión. Esto es vital para prevenir accesos no autorizados y garantizar la integridad de los datos.
- **Mantenimiento y actualización**: es más fácil mantener y actualizar sistemas que utilizan protocolos estándar, ya que las actualizaciones pueden aplicarse de manera uniforme y predecible.

- **Escalabilidad**: los sistemas basados en protocolos estándar pueden escalarse más fácilmente, permitiendo la adición de nuevos dispositivos y funcionalidades sin necesidad de rediseñar la infraestructura existente.

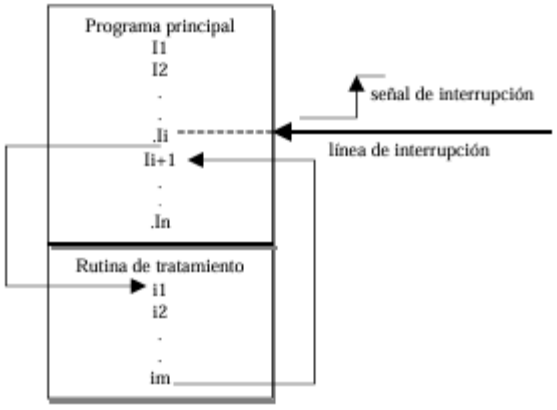
Manejo de Interrupciones

Las interrupciones son señales enviadas por un periférico a la CPU para indicar que requieren atención inmediata. Esto permite que la CPU pause su tarea actual, atienda el periférico y luego retome su tarea original. La CPU utiliza una tabla de interrupciones para identificar el tipo de interrupción y ejecutar el código necesario.

Básicamente una interrupción viene determinada por la ocurrencia de una señal externa que provoca la bifurcación a una dirección específica de memoria, interrumpiendo momentáneamente la ejecución del programa. A partir de esa dirección se encuentra la rutina de tratamiento que se encarga de realizar la operación de E/S propiamente dicha, devolviendo después el control al punto interrumpido del programa. El sistema de interrupciones permite una mejor sincronización de la E/S con el exterior y la posibilidad de compartir la CPU por más de un programa.

Figura 2

Diagrama que muestra la rutina de tratamiento de interrupciones.



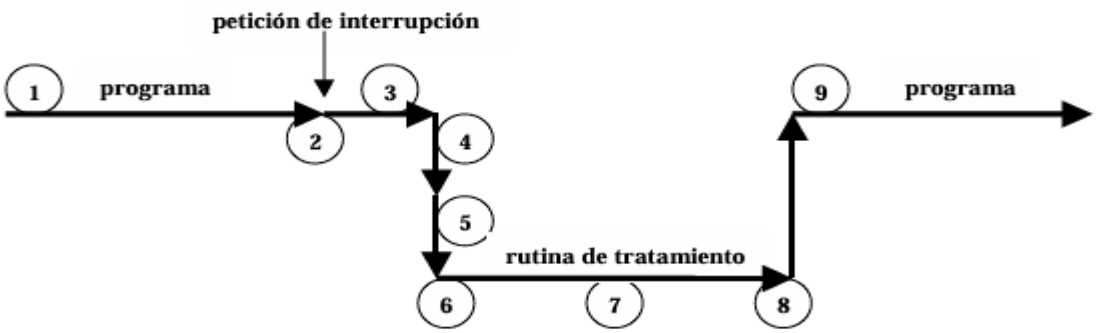
Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

El mecanismo de interrupción de un procesador debe incorporar todas las medidas necesarias para desviar la ejecución hacia la rutina de servicio y restaurar el estado del

programa interrumpido una vez que dicha rutina ha finalizado. En el siguiente esquema se detallan las fases que tienen lugar en la atención a una interrupción producida por algún dispositivo periférico:

Figura 3

Fases que tienen lugar en la atención a una interrupción producida por algún dispositivo periférico

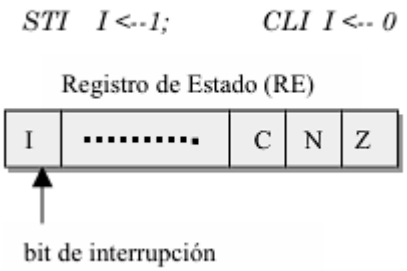


Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

Describamos cada uno de las fases a continuación:

1. El programa en ejecución (CPU) activa el sistema de interrupciones utilizando instrucciones que operan (ponen a 1 y a 0) sobre el bit de capacitación de las interrupciones I del registro de estado (RE):

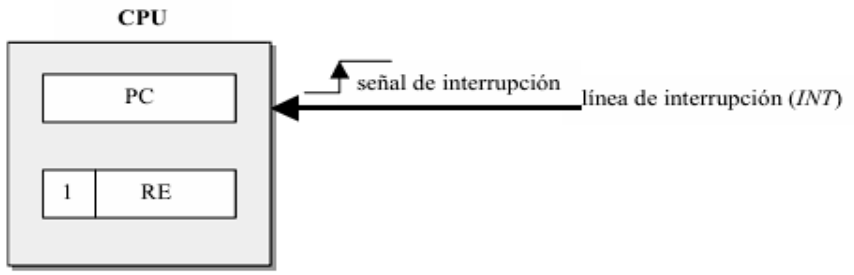
Figura 4



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

2. Se produce la petición de interrupción por parte de algún dispositivo periférico en un instante de tiempo impredecible para la CPU.

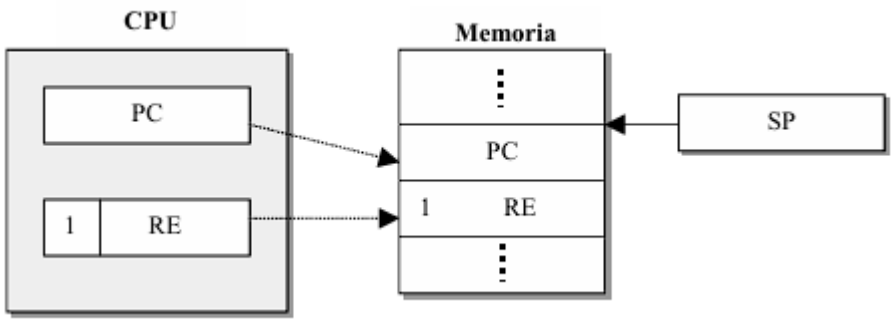
Figura 5



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

3. La CPU finaliza la ejecución de la instrucción en curso.
4. La CPU salva automáticamente el estado en la pila, es decir, el contador de programa (PC) y el registro de estado (RE):

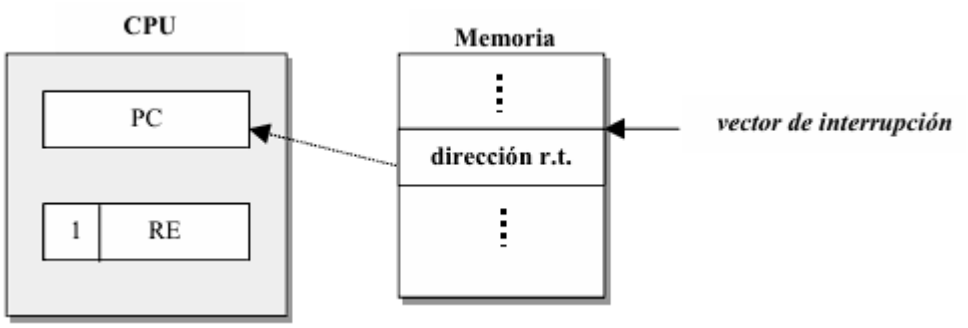
Figura 6



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

5. La CPU obtiene la dirección de la rutina de tratamiento a partir del vector de interrupción (VI), que usualmente se ubica en memoria.

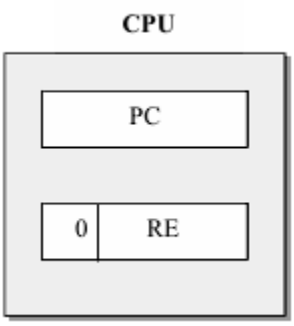
Figura 7



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

6. La CPU descapacita las interrupciones (I = 0) para que durante la ejecución de la primera instrucción de la rutina de tratamiento no se vuelva a detectar la misma interrupción y provoque un bucle infinito.

Figura 8

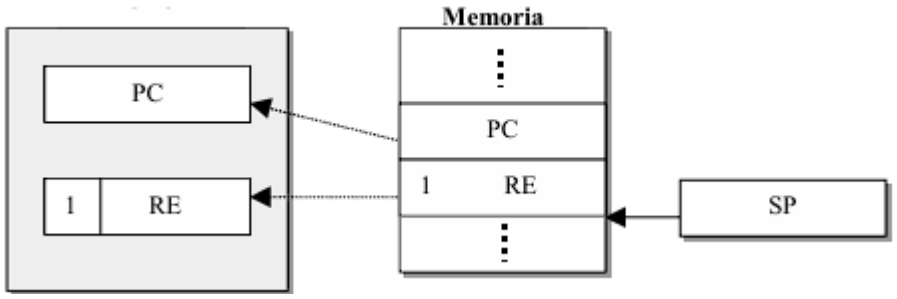


Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

7. La CPU ejecuta la rutina de tratamiento de la interrupción que realiza lo siguiente:
    - Salva en la pila los registros a utilizar.
    - Realiza la operación de Entrada/Salida
    - Restaura desde la pila los registros utilizados

8. Finaliza la rutina de tratamiento con la ejecución de la instrucción de retorno de interrupción (RTI), que restaura automáticamente el estado de la CPU desde la pila y vuelve al programa interrumpido:

Figura 9



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

9. La CPU continúa la ejecución del programa interrumpido, quedando las interrupciones capacitadas automáticamente al recuperarse el valor I = 1 del RE.

Transferencia de Datos

Para que un computador pueda ejecutar un programa debe ser ubicado previamente en la memoria, junto con los datos sobre los que opera, y para ello debe existir una unidad funcional de entrada de información capaz de escribir en la memoria desde el exterior. Análogamente, para conocer los resultados de la ejecución de los programas, los usuarios deberán poder leer el contenido de la memoria a través de otra unidad de salida de datos. La unidad de Entrada/Salida (E/S) soporta estas funciones, realizando las comunicaciones del computador (memoria) con el mundo exterior (periféricos).

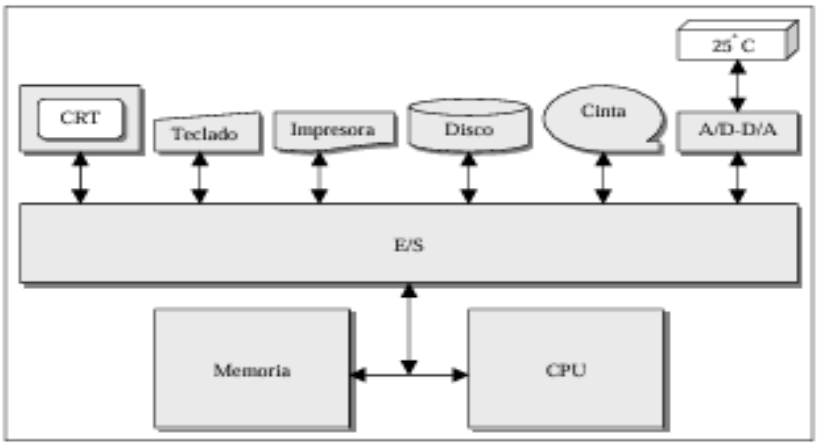
Clasificación de Dispositivos Periféricos

- **Dispositivos de Presentación de Datos:** son dispositivos con los que interactúan los usuarios, portando datos entre éstos y la máquina, por ejemplo, ratón, teclado, pantalla, impresora, etc.
- **Dispositivos de Almacenamiento de Datos:** son dispositivos que forman parte de la jerarquía de memoria del computador. Interactúan de forma autónoma con la máquina, aunque también sirven para el intercambio de datos con el usuario, por ejemplo, los discos magnéticos.

- **Dispositivos de Comunicación con Otros Procesadores:** permiten la comunicación con procesadores remotos a través de redes, por ejemplo, las redes de área local o global.
- **Dispositivos de Adquisición de Datos:** permiten la comunicación con sensores y actuadores que operan de forma autónoma en el entorno del computador. Se utilizan en sistemas de control automático de procesos por computador y suelen incorporar conversores de señales *analógico-digital* (A/D) y *digital-analógico* (D/A).

Figura 10

Transferencia de datos entre CPU y periféricos.



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

Los dispositivos de transporte y presentación de datos representan una carga muy baja de trabajo para el procesador comparados con los dispositivos de almacenamiento.

Figura 11

Velocidades de transferencia típicas para diferentes dispositivos.

Dispositivos	Velocidad
Sensores	1 Bps – 1 KBps
Teclado	10 Bps
Línea de comunicaciones	30 Bps – 20 MBps
Pantalla (CRT)	2 KBps
Impresora de línea	1 – 5 KBps
Cinta (cartridge)	0.5 – 2 MBps
Disco	4.5 MBps
Cinta	3-6 MBps

La velocidad de transferencia de los dispositivos de presentación de datos ha sido tradicionalmente menor en comparación con la de los dispositivos de almacenamiento. Sin embargo, en tiempos recientes, esta situación ha comenzado a cambiar. Los computadores se utilizan cada vez más para gestionar documentos multimedia que incluyen gráficos, videos y voz. Los gráficos demandan una gran capacidad de procesamiento de datos, hasta el punto de que se han desarrollado procesadores especializados para manejar eficientemente las representaciones gráficas, conocidos como GPU (Graphic Processor Unit). El video presenta un problema similar, ya que implica la animación de gráficos y requiere la creación de una nueva imagen cada 1/30 de segundo (33 milisegundos). El procesamiento de la voz también exige una alta capacidad, pues implica la creación o reconocimiento de fonemas en tiempo real, siendo el medio que más procesamiento requiere debido a que es el que menos tolera retrasos por parte del usuario.

Los dispositivos periféricos que pueden conectarse a un computador para realizar entrada y salida de información presentan, pues, las siguientes características:

- Tienen formas de funcionamiento muy diferentes entre sí, debido a las diferentes funciones que realizan y a los principios físicos en los que se basan.
- La velocidad de transferencia de datos es también diferente entre sí y diferente de la presentada por la CPU y la memoria.
- Suelen utilizar datos con formatos y longitudes de palabra diferentes

No obstante estas diferencias, existen una serie de funciones básicas comunes a todo dispositivo de E/S:

- Identificación única del dispositivo por parte de la CPU.
- Capacidad de envío y recepción de datos.
- Sincronización comunes a todo de la transmisión, exigida por la diferencia de velocidad de los dispositivos de E/S con la CPU.

La identificación del dispositivo se realiza con un decodificador de direcciones. El envío y la recepción de datos tiene lugar a través de registros de entrada y salida de datos. Los circuitos de sincronización se manipulan por medio de registros de estado y control. Los circuitos de sincronización aseguran que los datos se transfieran en el momento correcto, compensando la diferencia de velocidad entre los dispositivos y la CPU, en los sistemas modernos también se utilizan métodos de transferencia de datos como el *bus mastering* y el *Direct Memory Access* (DMA), donde ciertos dispositivos pueden comunicarse directamente

con la memoria sin involucrar constantemente a la CPU. Esto aligera la carga del procesador y optimiza la eficiencia.

Transferencia Síncrona

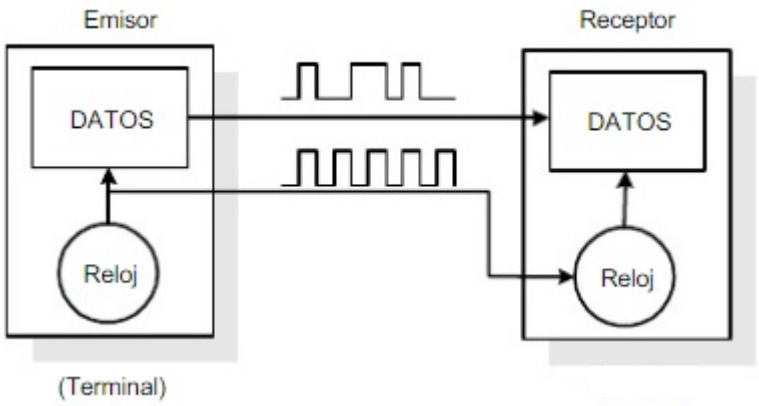
En transmisión síncrona se envía, además de los datos la señal de reloj; de esta manera el receptor se sincroniza con el emisor y determina los instantes significativos de la señal que recibe. Los datos se transmiten de manera consecutiva entre el emisor y el receptor,con un flujo constante que viene determinado por la señal del reloj de sincronismo.

Cuando se trata de transmisión de señales por pares metálicos en donde intervienen un terminal u ordenador (ETD) y un módem (ETCD), la señal o reloj de sincronismo del emisor puede generarse en cualquiera de estos dispositivos siendo común para ambos. El receptor es el encargado de generar la señal de sincronismo a partir de la señal que le llega por la línea.

En la transmisión síncrona los datos que se envían se agrupan en bloques formando tramas, que son un conjunto consecutivo de bits con un tamaño y estructura determinados. Este tipo de transmisión es más eficiente en la utilización del medio de transmisión que la asíncrona, siendo también más inmune a errores por lo que se suele usar para mayores velocidades que la asíncrona.

Figura 12

Diagrama de transferencia síncrona.



Fuente: Valvano J. W (2004), Introducción a los sistemas de microcomputadores Empotrados, México, Editorial Cengage Learning

Transferencia Asíncrona

Este modo de transmisión se caracteriza porque la base de tiempo del emisor y receptor no es la misma, empleándose un reloj para la generación de datos en la transmisión y otro distinto para la recepción.

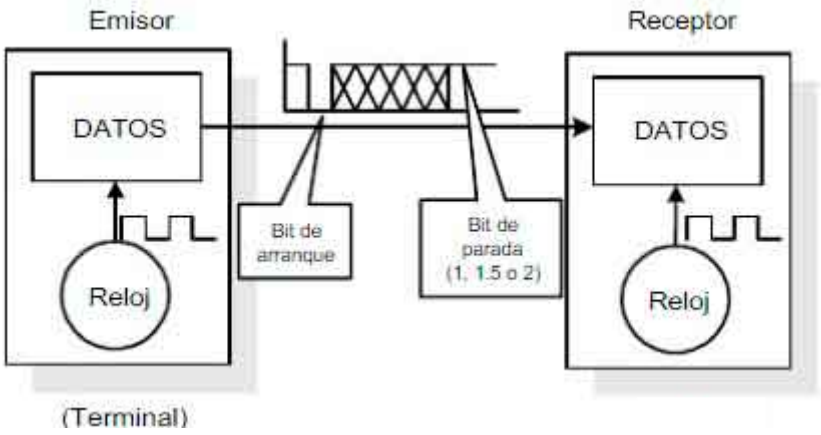
En este tipo de transmisión la información se transmite por palabras, bytes o conjunto de bits, estando precedidos estos bits por un bit de arranque o “start” y finalizando con al menos un bit de parada o “stop” pudiendo ser también 1,5 o 2 bits.

A este Conjunto de bits se le denomina carácter, pudiéndose transmitir en cualquier momento, es decir que entre dos informaciones consecutivas (al contrario de lo que ocurre en la transmisión síncrona) no tiene porqué haber un tiempo que sea múltiplo de un elemento unitario “bit”.

En este tipo de transmisión, el receptor sincroniza su reloj con el transmisor usando el bit de arranque que llega con cada carácter. Este método es más flexible, pues permite la comunicación con periféricos que tienen velocidades de transferencia variables o que operan a intervalos irregulares, aunque puede implicar mayores tiempos de espera y mecanismos de control adicionales para evitar errores.

Figura 13

Diagrama de transferencia asíncrona.



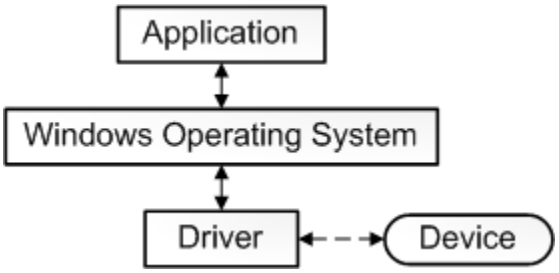
Fuente: Valvano J. W (2004), Introducción a los sistemas de microcomputadores Empotrados, México, Editorial Cengage Learning

Controladores de Dispositivos

Un controlador es un componente de software que facilita la comunicación entre el sistema operativo y un dispositivo. Por ejemplo, cuando una aplicación requiere leer datos de un dispositivo, realiza una solicitud a través de una función proporcionada por el sistema operativo. Este, a su vez, llama a una función dentro del controlador, que ha sido desarrollado generalmente por el fabricante del dispositivo y contiene las instrucciones necesarias para interactuar con el hardware. Una vez que el controlador obtiene los datos, los envía al sistema operativo, el cual finalmente los entrega a la aplicación.

Figura 14

Diagrama de funcionamiento de un controlador de dispositivos o driver.



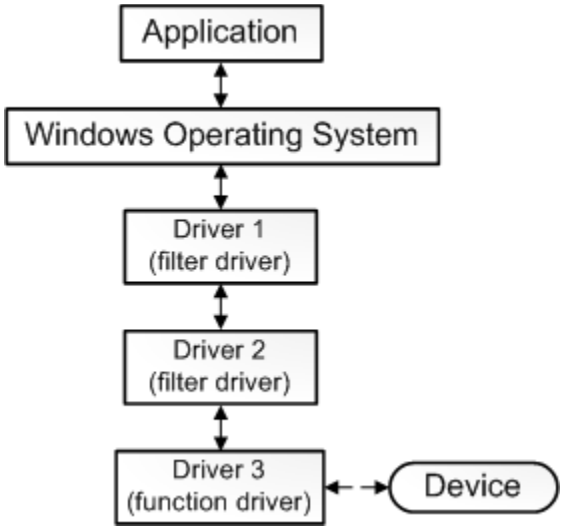
Fuente: @aviviano, @alexbuckgit, @mhopkins-msft (2024) https://learn.microsoft.com

No siempre es necesario que los controladores sean desarrollados por el fabricante del dispositivo. Si un dispositivo cumple con un estándar de hardware publicado, los desarrolladores del sistema operativo (OS) pueden crear el controlador, eliminando la necesidad de que el diseñador del dispositivo proporcione uno.

Además, no todos los controladores se comunican directamente con un dispositivo. A menudo, varios controladores superpuestos en una pila de controladores participan en una solicitud de E/S. La forma convencional de visualizar la pila es con el primer participante en la parte superior y el último participante de la parte inferior. Algunos controladores de la pila cambian la solicitud de un formato a otro. Estos controladores no se comunican directamente con el dispositivo. En su lugar, cambian la solicitud y la pasan a los controladores que están más bajos en la pila.

Figura 15

Diagrama de pila de controladores.



Fuente: @aviviano, @alexbuckgit, @mhopkins-msft (2024) https://learn.microsoft.com

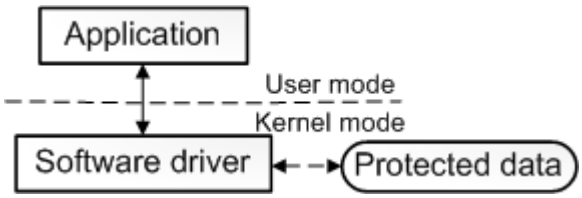
De allí que podamos diferenciar entre el *controlador de función,* aquel que se comunica directamente con el dispositivo y el *controlador de filtro*, que realizan el procesamiento auxiliar. Algunos controladores de filtro observan y registran información sobre las solicitudes de E/S, pero no participan activamente en ellas. Por ejemplo, algunos controladores de filtro actúan como comprobadores para asegurarse de que los demás controladores de la pila controlan correctamente la solicitud de E/S.

Existen controladores que no están vinculados a ningún dispositivo de hardware. Por ejemplo, si se desea desarrollar una herramienta que acceda a las estructuras de datos clave del sistema operativo, puede dividirse en dos partes: una que se ejecute en modo de usuario y brinde la interfaz de usuario, y otra que opere en modo kernel y acceda directamente a los datos del sistema operativo. El componente en modo de usuario es la aplicación, mientras que el que opera en modo kernel es un controlador de software. Este tipo de controlador no está asociado a ningún dispositivo de hardware.

El diagrama a continuación ilustra una aplicación en modo de usuario que interactúa con un controlador de software en modo kernel:

Figura 16

Diagrama de funcionamiento de un controlador de software en modo kernel.



Fuente: @aviviano, @alexbuckgit, @mhopkins-msft (2024) https://learn.microsoft.com

Los controladores de software operan exclusivamente en modo kernel, ya que están diseñados principalmente para acceder a datos protegidos que solo están disponibles en este nivel. Sin embargo, no todos los controladores de dispositivos requieren acceso a los recursos y datos de modo kernel; por ello, algunos controladores de dispositivo funcionan en modo de usuario.

Gestión de Compatibilidad

La gestión de la compatibilidad de controladores de dispositivos para diferentes periféricos se basa en varios componentes y estándares clave que permiten que el sistema operativo y los dispositivos puedan comunicarse eficazmente. Esta compatibilidad se asegura mediante:

- **Modelos de Controladores Abstratos**: los sistemas operativos, como Windows o Linux, emplean modelos de controladores que abstraen las operaciones de los dispositivos. Por ejemplo, el modelo WDM (Windows Driver Model) en Windows permite que los controladores desarrollados para versiones de Windows sean compatibles entre ellas, minimizando la necesidad de escribir controladores específicos para cada versión del sistema.
- **API y Especificaciones del Sistema Operativo**: los sistemas operativos proveen API específicas que los controladores deben utilizar para comunicarse con el sistema. Por ejemplo, en el caso de un dispositivo USB, el controlador debe implementar funciones específicas de la API USB de Windows, que permite al sistema operativo gestionar operaciones como la identificación del dispositivo, transferencia de datos y gestión de energía de forma estandarizada.
- **Uso de Controladores Genéricos y Específicos**: un controlador genérico puede funcionar con cualquier dispositivo que cumpla con ciertos estándares (por ejemplo, dispositivos de almacenamiento USB). Sin embargo, los controladores

específicos ofrecen optimizaciones y acceso a funciones avanzadas propias del hardware. Por ejemplo, en una impresora, el controlador genérico puede gestionar funciones de impresión básica, pero el controlador específico del fabricante permite configuraciones avanzadas, como ajustes de calidad y mantenimiento.

- **Frameworks y Capas de Abstracción**: en Linux, el uso del Kernel Mode Driver Framework (KMDF) y User Mode Driver Framework (UMDF) ayuda a que los controladores puedan operar en diferentes modos (kernel y usuario) dependiendo de los requerimientos de seguridad y acceso al hardware. En sistemas basados en Unix, frameworks como CUSE (Character Device in Userspace) permiten la creación de controladores de dispositivos en modo usuario para ciertos tipos de dispositivos.

Podemos dar un ejemplo con un dispositivo de almacenamiento USB: al conectar un dispositivo de almacenamiento USB a un sistema Windows, el sistema operativo primero identifica el tipo de dispositivo y busca un controlador compatible. Si el dispositivo cumple con la especificación USB Mass Storage Class, el sistema puede cargar el controlador genérico de almacenamiento USB (usbstor.sys) proporcionado por Windows, esto ocurriría en cuatro (4) pasos:

1. **Identificación del dispositivo:** el sistema envía una solicitud de enumeración al dispositivo USB. Este responde con sus identificadores de clase y subclase (USB Mass Storage Class).
2. **Carga del controlador genérico:** al reconocer la clase del dispositivo, el sistema carga el controlador usbstor.sys.
3. **Interacción con el sistema de archivos:** el controlador USB genérico actúa como puente entre el dispositivo de almacenamiento y el sistema de archivos del sistema operativo, facilitando la lectura y escritura de datos en el dispositivo.
4. **Gestión de operaciones específicas:** si el dispositivo requiere funciones avanzadas, el fabricante puede proporcionar un controlador específico que se superponga al genérico, proporcionando funcionalidades adicionales o personalizadas.

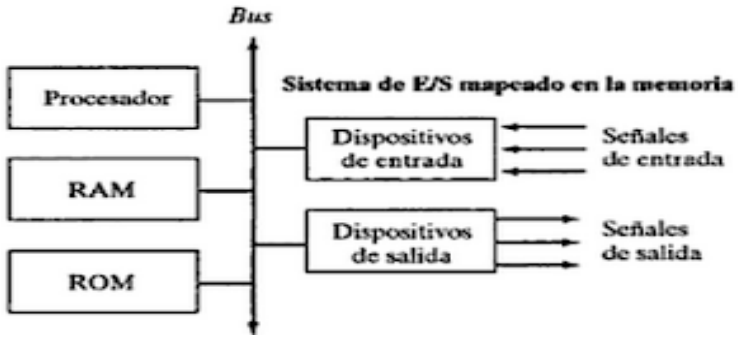
Mapeo de Direcciones

El mapeo de direcciones en la interfaz entre la CPU y los periféricos es el proceso mediante el cual la CPU accede y gestiona los dispositivos de entrada y salida (E/S). Este proceso puede llevarse a cabo mediante dos enfoques principales:

- **Mapeo de E/S con Direccionamiento de Memoria (E/S Mapeada en Memoria)**: en este enfoque, se reserva una parte del espacio de direcciones de memoria para los periféricos, como si fueran posiciones de memoria. Los periféricos se tratan como si fueran ubicaciones de memoria, permitiendo a la CPU leer y escribir directamente en ellos mediante las instrucciones normales de acceso a memoria (como LOAD y STORE). Las direcciones asignadas a los periféricos deben ser exclusivas, y a menudo se definen en el firmware o el sistema operativo. La ventaja de este enfoque es que permite utilizar las mismas instrucciones de acceso a memoria, simplificando el diseño de la CPU y la programación. E/S Mapeada en Memoria es más sencilla y puede hacer que el acceso sea más rápido, ya que no requiere instrucciones especiales, pero puede consumir espacio de direcciones.

Figura 17

E/S mapeada en memoria.



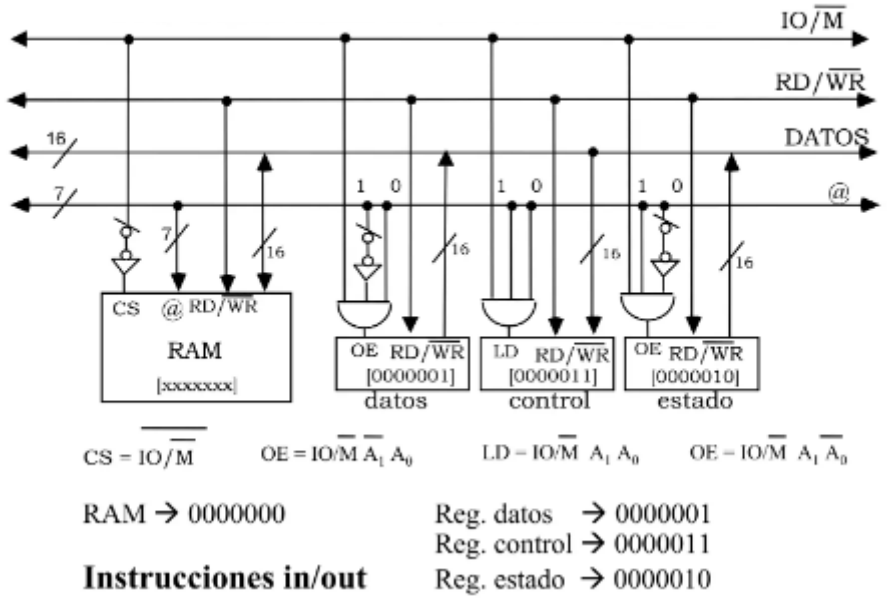
Fuente: Valvano J. W (2004), Introducción a los sistemas de microcomputadores Empotrados, México, Editorial Cengage Learning

- **Mapeo de E/S Aislada**: aquí, se reserva un espacio de direcciones específico para los periféricos, separado del espacio de direcciones de memoria principal. En lugar de utilizar las instrucciones de acceso a memoria, la CPU utiliza instrucciones especiales de entrada y salida (como IN y OUT en algunas arquitecturas) para comunicarse con los periféricos. Este enfoque permite mantener el espacio de direcciones de memoria libre para uso de datos y programas, ya que los periféricos no ocupan espacio en el mismo. La E/S aislada ahorra espacio de direcciones en la memoria principal y evita posibles

conflictos, aunque puede requerir un conjunto adicional de instrucciones y aumenta la complejidad del diseño de la CPU.

Figura 18

Ejemplo de E/S aislada



Fuente: Larraza, Edurne (2014)The Computer Input/Output Subsystem Education in an undergraduate introductory course: a Multiperspective Study.

Las partes del diagrama anterior son:

1. **Señales de control:**

- IO/M: esta señal indica si la operación es de memoria o de E/S. Cuando es cero (0) se accede a la memoria. Cuando es uno () se accede a ES.
- RD/WR: indica si la operación es de lectura (RD) o escritura (WR).

2. **Buses de datos y direcciones:**

- Bus de datos: tiene una anchura de 16 bits, lo que permite transferir datos de hasta 16 bits entre la CPU y los dispositivos de E/S o la memoria.
- Bus de direcciones: tiene una anchura de 7 bits, lo cual es suficiente para direccionar diferentes registros de control y de datos en el espacio de E/S.

3. **Memoria RAM:** se muestra una sección de la memoria RAM que está direccionada cuando la señal I/O/M = 0 (lo cual indica acceso a memoria) y

RD/WR según si es lectura o escritura. La dirección en la memoria RAM es 0000000 (7 bits en el bus de direcciones).

4. **Registro de E/S:**

- Registro de datos (0000001): registro para enviar o recibir datos desde un periférico.
- Registro de control (0000011): registro de control, utilizado para enviar señales de control a los periféricos.
- Registro de estado (0000010): registro de estado, que permite a la CPU leer el estado actual del periférico (por ejemplo, si está listo para recibir o enviar datos).

Cada uno de estos registros tiene una dirección específica (indicada en binario) dentro del espacio de E/S. Estas direcciones permiten que la CPU interactúe con estos registros usando las instrucciones IN/OUT.

5. **Lógica de decodificación:**

- LD y OE: se utilizan puertas lógicas para activar la lectura o escritura en cada registro específico dependiendo de las señales de control y las direcciones.
- Las señales de control OE (Output Enable) y LD (Load) determinan si se va a realizar una operación de lectura o escritura y a qué registro específico se accede en función de las líneas de dirección A_1 y A_0.

El funcionamiento general del sistema es el siguiente:

1. Acceso a la memoria: si la señal IO/M = 0, entonces la CPU accede a la memoria RAM usando las direcciones de memoria habituales y no a los registros de E/S.
2. Acceso a E/S: si la señal IO/M = 1, entonces la CPU accede a los registros de E/S en lugar de la memoria. Dependiendo de las señales RD/WR y del valor en el bus de direcciones (A_1 y A_0), se selecciona el registro específico:
    - 0000001 para el registro de datos.
    - 0000011 para el registro de control.
    - 0000010 para el registro de estado.

En este tipo de configuración de E/S aislada, la CPU utiliza instrucciones especiales, como IN/OUT para acceder a los registros de E/S, estas instrucciones son específicas para manejar dispositivos de E/S y funcionan solo cuando IO/M = 1.

Importancia de Tener un Espacio de Direcciones para Periféricos

Tener un espacio de direcciones para periféricos es crucial en la arquitectura de un sistema, ya que permite a la CPU acceder, gestionar y controlar los dispositivos de entrada y salida (E/S) de manera organizada y eficiente. La importancia de este espacio de direcciones para periféricos se puede desglosar en varios puntos clave:

1. **Optimización del espacio de memoria:** se evita que estos dispositivos ocupen direcciones de memoria destinadas a almacenar programas y datos. Esto es especialmente importante en sistemas con espacio de memoria limitado. Si los periféricos ocuparan el mismo espacio que la memoria de datos y programas, se reduciría el espacio disponible para la ejecución de aplicaciones y almacenamiento de datos, limitando la capacidad del sistema.
2. **Acceso directo y eficiente a los periféricos:** permite a la CPU interactuar con los periféricos de manera rápida y directa. Esto facilita la comunicación entre la CPU y los dispositivos externos, ya que estos dispositivos pueden ser referenciados directamente mediante direcciones específicas, acelerando el acceso y reduciendo la complejidad en la programación.
3. **Evita conflictos en la asignación de direcciones: u**n espacio separado para periféricos (como en el E/S Aislada) previene posibles conflictos entre las direcciones de memoria de datos/programas y las direcciones asignadas a los dispositivos de E/S. Al reservar un conjunto exclusivo de direcciones, se elimina la posibilidad de que la CPU acceda a la dirección de un periférico cuando debería estar accediendo a datos o viceversa.
4. **Flexibilidad en el diseño del sistema:** permite ampliar el sistema y agregar nuevos dispositivos sin modificar el espacio de direcciones de la memoria principal. Esto facilita la integración de nuevos periféricos o la re-configuración de dispositivos existentes, algo especialmente útil en sistemas embebidos o de propósito general que requieren una gestión de E/S flexible.
5. **Simplificación en la programación y control de dispositivos:** los programadores pueden referirse a ellos de manera consistente, sin tener que manejar configuraciones complejas de acceso. Esto reduce la posibilidad de errores en la programación y facilita la creación de controladores de dispositivos (drivers), que son fundamentales para el correcto funcionamiento de los periféricos.

E/S Mapeada en Memoria vs. E/S Aislada

	E/S Mapeada en Memoria	E/S Aislada
Espacio de Direcciones	Usa el mismo espacio de direcciones de la memoria principal.	Tiene un espacio de direcciones separado para los periféricos.
Instrucciones	Utiliza las mismas instrucciones de acceso a memoria (*LOAD*, *STORE*).	Requiere instrucciones específicas de E/S (como *IN* y *OUT*).
Facilidad de Programación	Simplifica la programación, ya que no se requieren instrucciones especiales para E/S.	Puede ser más complejo, ya que se necesitan instrucciones adicionales para E/S
Acceso a Periféricos	Los periféricos se acceden como si fueran posiciones de memoria, lo cual facilita su manipulación.	Los periféricos se acceden mediante puertos de E/S, lo que puede ser menos intuitivo.
Velocidad de Acceso	Permite un acceso más rápido, ya que usa instrucciones normales de memoria.	Puede ser ligeramente más lento debido al uso de instrucciones específicas de E/S.
Riesgo de Conflictos	Existe riesgo de conflictos entre la memoria y los periféricos, especialmente en sistemas con pocos bits.	Minimiza el riesgo de conflictos entre direcciones de memoria y periféricos.
Uso de Espacio de Direcciones	Consume espacio de direcciones de memoria, reduciendo el espacio disponible para programas y datos.	Conserva el espacio de direcciones de la memoria para datos y programas.
Aplicación Ideal	Sistemas con espacio de direcciones amplio (32 o 64 bits) y donde la simplicidad es prioritaria.	Sistemas con espacio de direcciones reducido (8 o 16 bits) o en aplicaciones embebidas.
Control de Acceso	Menor control sobre el acceso a periféricos, ya que estos comparten el espacio de direcciones de memoria.	Permite un mayor control y aislamiento de dispositivos, útil en sistemas críticos.
Tipo de Sistemas Preferido	Sistemas que no requieren instrucciones de E/S especiales y permiten la programación sencilla.	Sistemas embebidos o críticos donde el control de acceso y el aislamiento son necesarios.

Acceso Directo a Memoria (DMA)

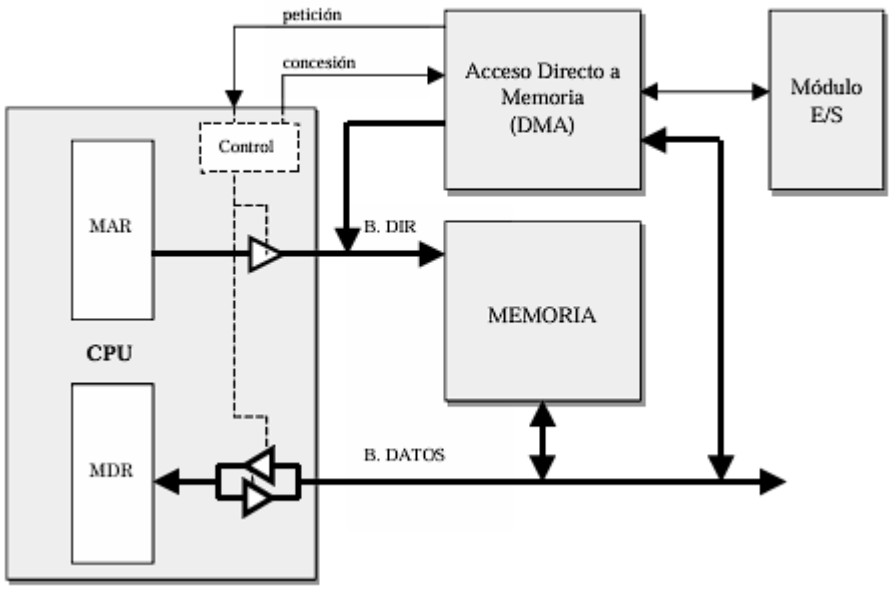
La E/S mediante interrupciones consume menos tiempo de CPU en comparación con la E/S controlada por programa. Sin embargo, en ambos métodos, las transferencias de datos deben pasar a través de la CPU. Esto implica que la velocidad de las transferencias está limitada por la rapidez con la que la CPU puede atender al dispositivo periférico, ya que debe gestionarlas ejecutando una serie de instrucciones. En la E/S controlada por programa, la CPU se dedica exclusivamente a las operaciones de E/S, logrando transferir los datos a una velocidad razonable, pero a cambio de estar ocupada en esta tarea de manera continua. Por otro lado, el uso de interrupciones libera en gran medida a la CPU de la gestión directa de las transferencias, aunque esto puede reducir su velocidad, ya que la rutina de servicio de interrupciones generalmente incluye instrucciones adicionales que no están directamente relacionadas con la transferencia de datos. En conclusión, ambos métodos presentan ciertas limitaciones que afectan tanto la actividad de la CPU como la velocidad de transferencia.

Por tanto, cuando se tienen que transferir grandes cantidades de datos y a una elevada velocidad, es necesario disponer de una técnica que realice de forma más directa las transferencias entre la memoria y el dispositivo periférico, limitando al máximo la intervención de la CPU. Esta técnica se conoce con el nombre de *acceso directo a memoria (DMA: Direct Memory Access).*

Se trata de un módulo capaz de leer y escribir directamente en la memoria los datos que provienen o se envían a los dispositivos periféricos. Para realizar esto, el módulo DMA (Acceso Directo a Memoria) solicita a la CPU el acceso a la memoria. Antes de concederle este acceso, la CPU coloca su conexión con los buses del sistema (datos, direcciones y R/W) en estado de alta impedancia, lo cual equivale a desconectarse temporalmente de la memoria mientras esta es gestionada por el DMA. Una vez que se completa la operación de E/S, el DMA genera una interrupción, permitiendo que la CPU recupere el control de la memoria. De este modo, la velocidad de transferencia queda limitada únicamente por el ancho de banda de la memoria.

Figura 19

DMA



Fuente: (S/F) Estructura de Computadores, Facultad de Informática, Universidad Complutense de Madrid.

Estructura y Funcionamiento de un Controlador de DMA

Para gestionar las transferencias de información, un controlador de DMA dispone de 3 registros: datos, dirección y contador de palabras.

El registro de dirección guarda la dirección de la próxima palabra a transmitir y se incrementa automáticamente después de cada transferencia. El contador de palabras almacena la cantidad de palabras restantes por transmitir y también se decrementa automáticamente tras cada transferencia. La lógica de control monitorea el contenido del contador de palabras y termina la operación cuando este llega a 0. Un decodificador se encarga de identificar la dirección de memoria asignada al módulo DMA. Para iniciar una operación de E/S, la CPU debe proporcionar al DMA la siguiente información:

- El tipo de operación de E/S: lectura o escritura.
- La dirección del periférico.
- La posición de memoria donde comienza el bloque de datos a leer o escribir.
- La cantidad de palabras que contiene el bloque.

Una vez que se envía esta información, la CPU continúa con otras tareas, delegando la operación de E/S completamente al DMA. El DMA transfiere directamente el bloque de datos,

palabra por palabra, entre el periférico y la memoria, sin necesidad de pasar por la CPU. Al finalizar la transferencia, el DMA envía una señal de interrupción a la CPU.

Para poder transferir a/desde la memoria, el DMA necesita controlar el bus durante un tiempo suficiente para completar la transferencia. Sin embargo, este tiempo no tiene que ser continuo, puede fraccionarse en pequeños intervalos que se alternan con la CPU. Existen diferentes alternativas en la forma de controlar el bus. Cada alternativa supone un compromiso diferente entre velocidad de transferencia y actividad de la CPU. El empleo de una alternativa en concreto dependerá de las prestaciones que se deseen y de las características del procesador que se utilice. Estas son:

- Ráfagas: el DMA toma control del bus y no lo libera hasta haber transmitido un bloque de datos completo. Con este método se consigue la mayor velocidad de transferencia pero puede dejar sin actividad a la CPU durante períodos grandes de tiempo.

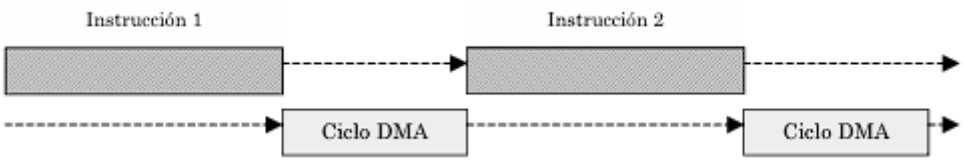
Figura 20

Ráfaga

- Robo de ciclos: el DMA toma control del bus y lo retiene durante un solo ciclo. Transmite una palabra y libera el bus. Es la forma más usual de transferencia y en ella se dice que el DMA roba ciclos a la CPU.

Figura 21

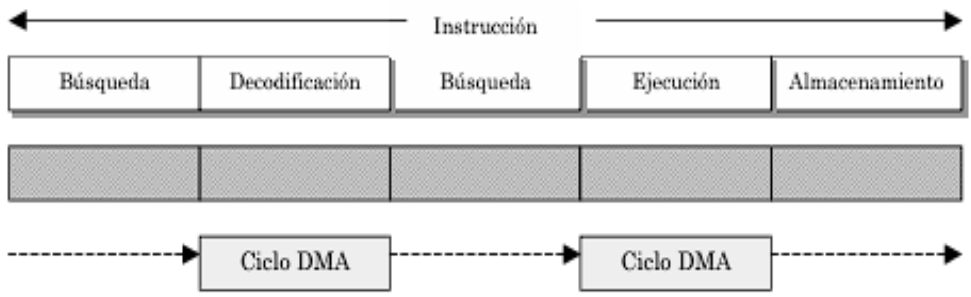
Robo de ciclos



- Transparente: el DMA accede al bus sólo en los ciclos en los que la CPU no lo utiliza. Y esto ocurre en diferentes fases de ejecución de las instrucciones. De esta forma la ejecución del programa no se ve afectado en su velocidad de ejecución.

Figura 22

Transparente



Los pasos que se siguen en la transferencia mediante DMA son:

1. La CPU ejecuta las instrucciones de E/S que cargan los registros de dirección y contador de palabras del controlador de DMA. El registro de dirección debe contener la dirección base de la zona de memoria principal que se va a utilizar en la transferencia de datos. El registro contador de palabra almacena el número de palabras que se transfieren desde/hacia la memoria.
2. Cuando el controlador de DMA está preparado para transmitir datos, activa la línea de petición. La CPU espera a un punto de concesión del DMA, renuncia al control de los buses de datos y direcciones y activa la línea de reconocimiento de DMA.
3. El DMA transfiere directamente los datos a/desde memoria principal por alguno de los métodos que se acaban de ver. Después de transferir una palabra, el registro de dirección y el registro contador de palabras del controlador se incrementa y decrementa respectivamente.
4. Si el contenido del registro contador de palabra no es 0, pero el periférico no está preparado para enviar o recibir los siguientes datos, el DMA devuelve el control a la CPU liberando el bus del sistema y desactivando la línea de petición. La CPU responde desactivando la línea de reconocimiento de DMA y continuando con su operación normal.

5. Si el contenido del contador de palabras es 0, el controlador de DMA renuncia al control del bus del sistema y envía una señal de interrupción a la CPU.

El DMA se puede configurar de diferentes formas:

1. Bus único: todos los módulos comparten el bus del sistema. El DMA, que actúa en sustitución de la CPU, intercambia datos entre la memoria y el periférico utilizando un procedimiento análogo al de E/S controlada por programa, es decir, hace de intermediario entre ambas unidades. Esta configuración, aunque puede ser muy económica, es claramente poco eficaz, ya que cada transferencia de una palabra consume 2 ciclos del bus.
2. Integración de funciones DMA-E/S: Reduce a 1 el número de ciclos de utilización del bus. Esto significa que hay un camino entre el controlador de DMA y uno o más controladores de E/S que no incluyen al bus del sistema. La lógica del DMA puede ser una parte de un controlador de E/S o puede ser un módulo independiente que controla a uno o más controladores de E/S.
3. Bus de E/S conectado al DMA: se puede generalizar si se utiliza un bus de E/S para conectar los controladores de E/S al DMA. Esta alternativa reduce a una el número de interfaces de E/S en el DMA, y proporciona una configuración fácilmente ampliable.

Beneficios del Uso de DMA

1. Liberación de la CPU: la CPU no tiene que gestionar directamente las transferencias de datos entre la memoria y los periféricos. Esto permite que la CPU se ocupe de otras tareas mientras el DMA realiza la transferencia de datos de forma autónoma.
2. Transferencias más rápidas: el DMA puede mover datos directamente entre la memoria y los periféricos sin intervención de la CPU, lo que reduce la latencia y mejora la velocidad de transferencia. La velocidad está limitada solo por el ancho de banda de la memoria y los periféricos, no por el procesamiento de la CPU.
3. Optimización del rendimiento del sistema: la carga de trabajo de la CPU disminuye, ya que no necesita ejecutar instrucciones de E/S repetitivas. Esto permite un uso más eficiente del procesador, especialmente en sistemas que requieren un procesamiento intensivo.

4. Facilita la transferencia de bloques de datos grandes: DMA es especialmente útil para mover grandes cantidades de datos (como en aplicaciones de audio, video o almacenamiento masivo) de manera rápida y eficiente, evitando el consumo de recursos de la CPU que requeriría la E/S controlada por programa.
5. Reducción de errores en la transferencia: al automatizar la transferencia de datos, se reduce la probabilidad de errores humanos o de programación en el proceso de E/S, lo cual puede mejorar la confiabilidad del sistema.

Desafíos del Uso de DMA

1. Complejidad en el diseño del hardware: la implementación de DMA en el hardware del sistema requiere una lógica de control adicional para gestionar la transferencia de datos. Esto puede aumentar la complejidad y el costo de diseño de la arquitectura.
2. Competencia por el acceso a los buses: Como el DMA utiliza los buses de datos y direcciones para transferir datos, puede generar competencia con la CPU por el acceso a estos recursos. Esta competencia puede ocasionar breves períodos de espera (llamados *ciclos robados* o *bus contention*), lo que puede ralentizar otras operaciones en el sistema.
3. Coordinación y sincronización: la CPU y el DMA deben coordinarse para evitar conflictos. La CPU debe poner sus buses en estado de alta impedancia mientras el DMA tiene el control, y cualquier error de sincronización podría resultar en una pérdida o corrupción de datos.
4. Latencia en el cambio de control: después de que el DMA completa una transferencia, envía una señal de interrupción a la CPU. Este cambio de control puede generar una pequeña latencia, que puede ser crítica en aplicaciones que requieren respuestas en tiempo real.
5. Consumo adicional de energía: al añadir un módulo de DMA al sistema, se incrementa el consumo de energía, lo que puede ser un problema en dispositivos con recursos limitados, como los dispositivos móviles.
6. Limitaciones en la flexibilidad de transferencia: el DMA es ideal para transferencias de datos secuenciales de grandes bloques, pero puede no ser tan eficiente para transferencias no secuenciales o pequeñas. Esto puede ser una limitación en algunos tipos de aplicaciones de E/S.

Latencia y Ancho de Banda

La *latencia* es el tiempo que tarda una señal en viajar desde el emisor hasta el receptor. En sistemas de comunicación, es el tiempo de respuesta entre la solicitud y la recepción de los datos. La latencia se mide en milisegundos (ms) y puede ser afectada por factores como la distancia, la congestión de la red y el tipo de medio de transmisión.

Por otra parte, el *ancho de banda* es la capacidad máxima de transferencia de datos de una red o conexión en un período de tiempo determinado. Se mide en bits por segundo (bps) y define la cantidad de información que se puede transmitir simultáneamente. Un mayor ancho de banda permite transmitir más datos en menos tiempo.

	Latencia	**Ancho de Banda**
Unidades de Medida	Generalmente se mide en milisegundos (ms).	Se mide en bits por segundo (bps), como Mbps (megabits por segundo) o Gbps (gigabits por segundo).
Impacto en el Rendimiento	Una alta latencia puede resultar en demoras perceptibles en la respuesta, afectando el rendimiento de aplicaciones que requieren tiempo real.	Un bajo ancho de banda limita la cantidad de datos transferidos, lo cual puede hacer más lentas las aplicaciones que manejan grandes volúmenes.
Ideal para...	Procesos que necesitan respuestas inmediatas con baja transferencia de datos, como periféricos de control y entrada.	Procesos que manejan grandes cantidades de datos o transferencias continuas, como dispositivos de almacenamiento y transmisiones de multimedia.
Afectación por Alta Latencia	Puede provocar retrasos en la interacción, especialmente notables en aplicaciones interactivas, como videojuegos o interfaces de control en tiempo real.	Una baja velocidad de respuesta que retrase la transferencia de archivos grandes, lo cual impacta en aplicaciones como la edición de video y almacenamiento en red.
Afectación por Bajo Ancho de Banda	La baja capacidad de transmisión podría no afectar tanto los periféricos de baja transferencia, pero limita considerablemente el desempeño en sistemas de alto tráfico.	Un bajo ancho de banda puede causar pérdida de calidad en transmisiones en tiempo real y mayor tiempo de espera para transferencias de archivos grandes.

Factores de Influencia	Distancia entre dispositivos, calidad de conexión y tipo de interfaz (USB, PCIe, Ethernet).	Capacidad de conexión de la red, calidad de hardware y tipo de interfaz de conexión.
Optimización	Conexiones directas y de baja latencia (como PCIe en vez de USB o Ethernet).	Uso de conexiones de alta velocidad (USB 3.0, Thunderbolt, PCIe) y optimización de red para maximizar la transmisión simultánea de datos.
Ejemplo en la Interfaz CPU-Periférico	En la comunicación CPU-teclado, la latencia afecta la velocidad de respuesta de la interfaz al usuario.	En la comunicación CPU-disco duro, el ancho de banda afecta la rapidez con la que se pueden leer/escribir grandes volúmenes de datos.

Periféricos que Requieren Baja Latencia

- Mouse y teclado: necesitan una respuesta inmediata para evitar retrasos en la interacción.
- Dispositivos de entrada en tiempo real: como los controles de juegos o interfaces de realidad virtual.
- Sensores en sistemas de tiempo real: dispositivos médicos.

Periféricos que Requieren Alto Ancho de Banda

- Tarjetas de red y adaptadores de red: necesitan alta capacidad para transmitir grandes volúmenes de datos.
- Dispositivos de almacenamiento externo: como discos duros, SSD, etc., requieren ancho de banda elevado para transferencias rápidas de archivos grandes.
- Cámaras para videoconferencia.

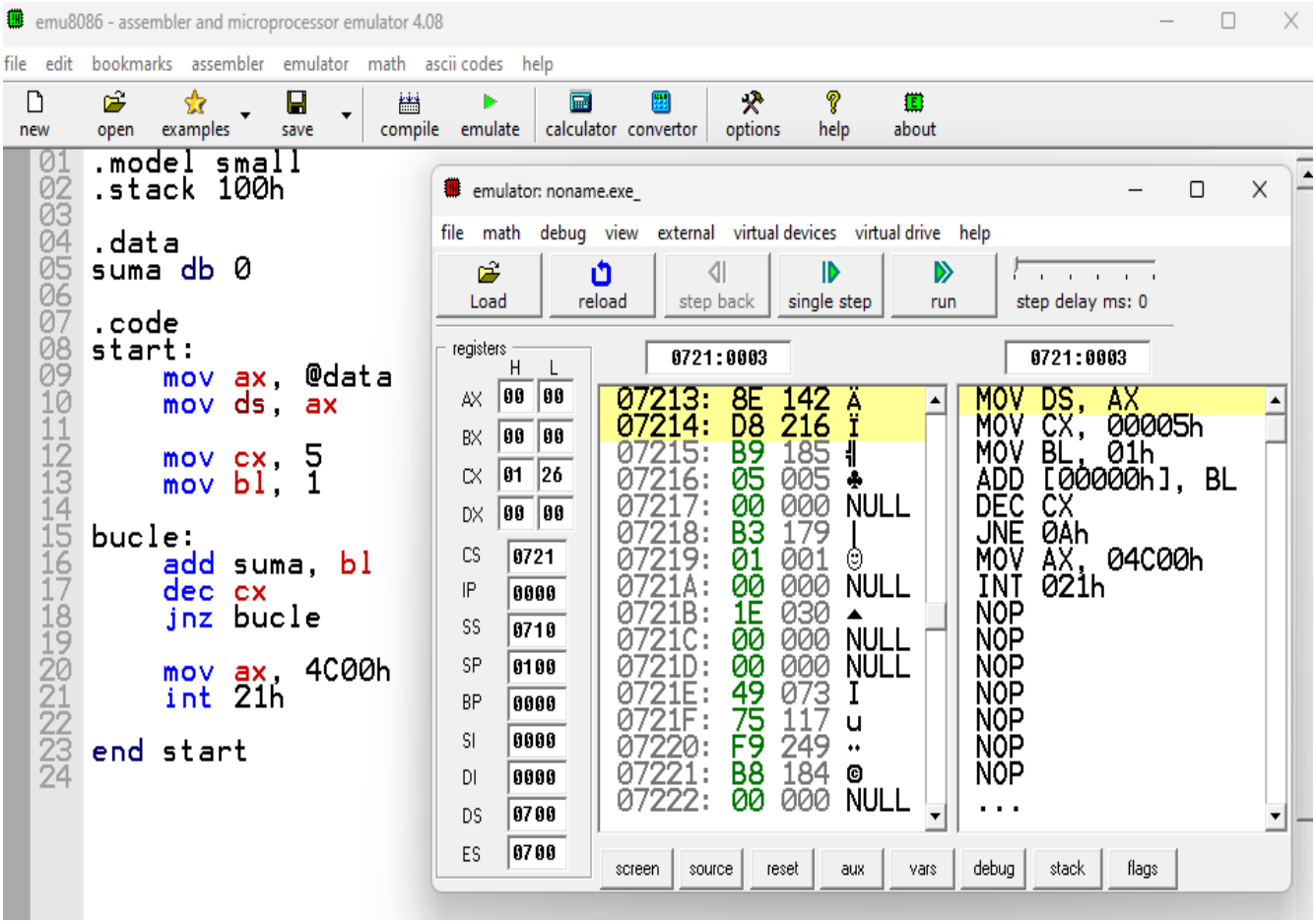
Tareas Realizadas en el Emulador de Arquitectura del Computador EMU8086

Unidad de Control (UC)

Uso Intensivo de la UC

Utilizando el lenguaje ensamblador (asm):

```
.model small
.stack 100h

.data
suma db 0; Variable para almacenar la suma

.code
start:
    mov ax, @data; Inicializa el segmento de datos
    mov ds, ax

    mov cx, 5; Establece un contador en 5
    mov bl, 1; Establece un valor a sumar (1 en este caso)

bucle:
    add suma, bl; Suma el valor de bl a suma
    dec cx; Decrementa el contador
    jnz bucle; Si cx no es cero, repite el bucle

    mov ax, 4C00h   ; Código de salida
    int 21h         ; Interrupción de DOS para salir

end start
```



Proceso de la Unidad de Control en el Código

1. Fetch: la UC comienza por obtener la primera instrucción desde la dirección de memoria donde se encuentra el segmento de código **start:**, la instrucción es **mov ax**, **@data**, que inicializa el segmento de datos. Después de recuperar la instrucción, la UC incrementa automáticamente el Contador de Programa (PC) para que apunte a la siguiente instrucción.
2. Decode: La UC decodifica la instrucción **mov ax**, **@data**, comprendiendo que debe cargar la dirección del segmento de datos en el registro **ax**. Este paso es crucial porque prepara a la CPU para la siguiente etapa.
3. Ejecución: la UC envía señales de control a los componentes correspondientes para que se ejecute la instrucción. En este caso, se establece **ds** con el valor de **ax**, permitiendo que el programa acceda correctamente a las variables en la sección de datos.

4. Repetición del bucle: el bucle se activa con la instrucción **bucle:**, donde se ejecutan las siguientes operaciones:
   - **add suma, bl**: La UC dirige a la ALU para que sume el valor de **bl** a la variable **suma**.
   - **dec cx**: decrementa el contador **cx**. La UC asegura que este paso ocurra inmediatamente después de la suma.
   - **jnz bucle**: la UC evalúa el estado del registro **cx**. Si **cx** no es cero, salta de vuelta a la etiqueta **bucle**, repitiendo las instrucciones de suma y decremento hasta que **cx** alcance cero.
5. Finalización: Una vez que el bucle se completa (cuando cx llega a cero), la siguiente instrucción **mov ax, 4C00h** se ejecuta para preparar la salida del programa. La UC dirige la ejecución hacia el código de salida.

Importancia de la Unidad de Control en el Ciclo de Instrucción

Aspecto	Descripción
Coordinación de Operaciones	La UC asegura que las operaciones se realicen en el orden correcto, fundamental para la ejecución de bucles y estructuras de control.
Control de Flujo	Maneja las instrucciones de salto y condiciones, permitiendo que el programa altere su flujo de ejecución basado en condiciones específicas.
Gestión de Recursos	Dirige el acceso a recursos de la CPU, como registros y la ALU, asegurando que estén listos para las operaciones requeridas.
Eficiencia en la Ejecución	Optimiza el ciclo de instrucción (fetch, decode, execute) para minimizar el tiempo de procesamiento y mejorar el rendimiento del sistema.
Facilitación de la Lógica de Programación	Permite la implementación de bucles y condiciones, lo que habilita la creación de programas más complejos y funcionales.

Unidad Aritmético-Lógica (ALU)

Operaciones Aritméticas y Lógica Usando la ALU

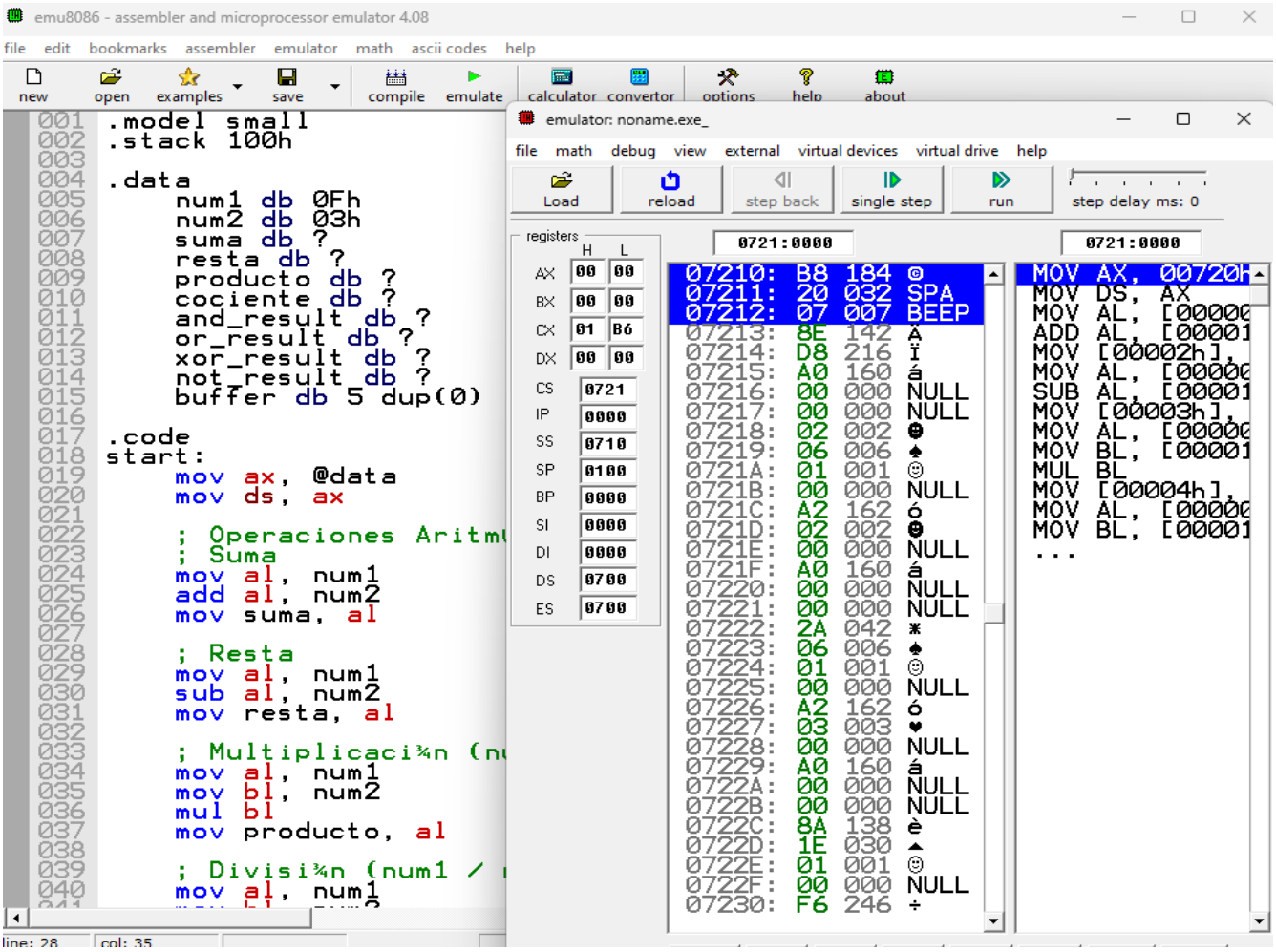
Siguiendo con el lenguaje ensamblador (asm) se realiza el siguiente código:

```
.model small
.stack 100h

.data
    num1 db 0Fh ; Primer número (15 en decimal)
    num2 db 03h ; Segundo número (3 en decimal)
    suma db ? ; Variable para almacenar la suma
    resta db ?  ; Variable para almacenar la resta
    producto db ? ; Variable para almacenar el producto
    cociente db ? ; Variable para almacenar el cociente
    and_result db ? ; Resultado de AND
    or_result db ? ; Resultado de OR
    xor_result db ? ; Resultado de XOR
    not_result db ? ; Resultado de NOT
    buffer db 5 dup(0) ; Buffer para mostrar resultados

.code
start:
    mov ax, @data ; Inicializa el segmento de datos
    mov ds, ax

    ; Operaciones Aritméticas
    ; Suma
    mov al, num1 ; Carga num1 en AL
    add al, num2 ; Suma num2 a AL
    mov suma, al ; Almacena el resultado en suma
```

```
    ; Resta
    mov al, num1 ; Carga num1 en AL
    sub al, num2 ; Resta num2 de AL
    mov resta, al ; Almacena el resultado en resta

; Multiplicación (num1 * num2)
    mov al, num1 ; Carga num1 en AL
    mov bl, num2 ; Carga num2 en BL
    mul bl ; Multiplica AL por BL (resultado en AX)
    mov producto, al ; Almacena el resultado (8 bits)

    ; División (num1 / num2)
    mov al, num1 ; Carga num1 en AL
    mov bl, num2 ; Carga num2 en BL
    xor ah, ah ; Limpia AH para la división
    div bl ; Divide AX entre BL (cociente en AL, resto en AH)
    mov cociente, al ; Almacena el cociente

    ; Operaciones Lógicas
    ; AND
    mov al, num1 ; Carga num1 en AL
    and al, num2 ; Realiza AND con num2
    mov and_result, al ; Almacena el resultado

    ; OR
    mov al, num1 ; Carga num1 en AL
    or al, num2 ; Realiza OR con num2
    mov or_result, al ; Almacena el resultado

    ; XOR
    mov al, num1 ; Carga num1 en AL
    xor al, num2 ; Realiza XOR con num2
    mov xor_result, al ; Almacena el resultado
```

```
; NOT
mov al, num1 ; Carga num1 en AL
not al ; Realiza NOT
mov not_result, al ; Almacena el resultado
; Mostrar Resultados
; Suma
mov al, suma ; Cargar el resultado de la suma
call PrintResult

; Resta
mov al, resta ; Cargar el resultado de la resta
call PrintResult

; Producto
mov al, producto ; Cargar el resultado del producto
call PrintResult

; Cociente
mov al, cociente ; Cargar el resultado del cociente
call PrintResult

; AND
mov al, and_result ; Cargar el resultado del AND
call PrintResult

; OR
mov al, or_result ; Cargar el resultado del OR
call PrintResult

; XOR
mov al, xor_result ; Cargar el resultado del XOR
call PrintResult
```

```
    ; NOT
    mov al, not_result ; Cargar el resultado del NOT
    call PrintResult

    ; Finaliza el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

; Subrutina para imprimir el resultado
PrintResult:
    ; Convierte el valor de AL a una cadena en ASCII
    mov bx, 10 ; Divisor para conversión decimal
    xor cx, cx ; Limpia el contador

convert_loop:
    xor dx, dx ; Limpia DX antes de dividir
    div bx ; Divide AL por 10, cociente en AL, resto en DX
    push dx ; Almacena el resto en la pila
    inc cx ; Incrementa el contador
    test al, al ; Verifica si AL es cero
    jnz convert_loop ; Si no es cero, sigue convirtiendo

; Imprimir el resultado en la pantalla
print_loop:
    pop dx ; Recupera el valor del tope de la pila
    add dl, '0' ; Convierte el dígito a ASCII
    mov ah, 02h ; Función de impresión de carácter
    int 21h ; Interrupción de DOS

    ret
end start
```



Pasos de la ALU para Realizar una Operación de Suma y una Operación Lógica

Aspecto	Operación de Suma	Operación Lógica (AND)
Carga de Operandos	Carga de los operandos en registros (ej. AL, BL).	Carga de los operandos en registros (ej. AL, BL).
Activación de la UC	La Unidad de Control envía una señal con el opcode para suma.	La Unidad de Control envía una señal con el opcode para AND.
Ejecución de la Operación	Se suma el contenido de los registros; puede haber acarreo.	Se evalúa bit a bit; cada bit se compara y se aplica la operación lógica.
Almacenamiento del Resultado	El resultado se almacena en un registro (ej. AL o AX).	El resultado se almacena en un registro.
Actualización de Bandera	Se actualizan las banderas de estado (como cero, acarreo).	Se actualizan las banderas de estado (como cero).

Similitudes y Diferencias

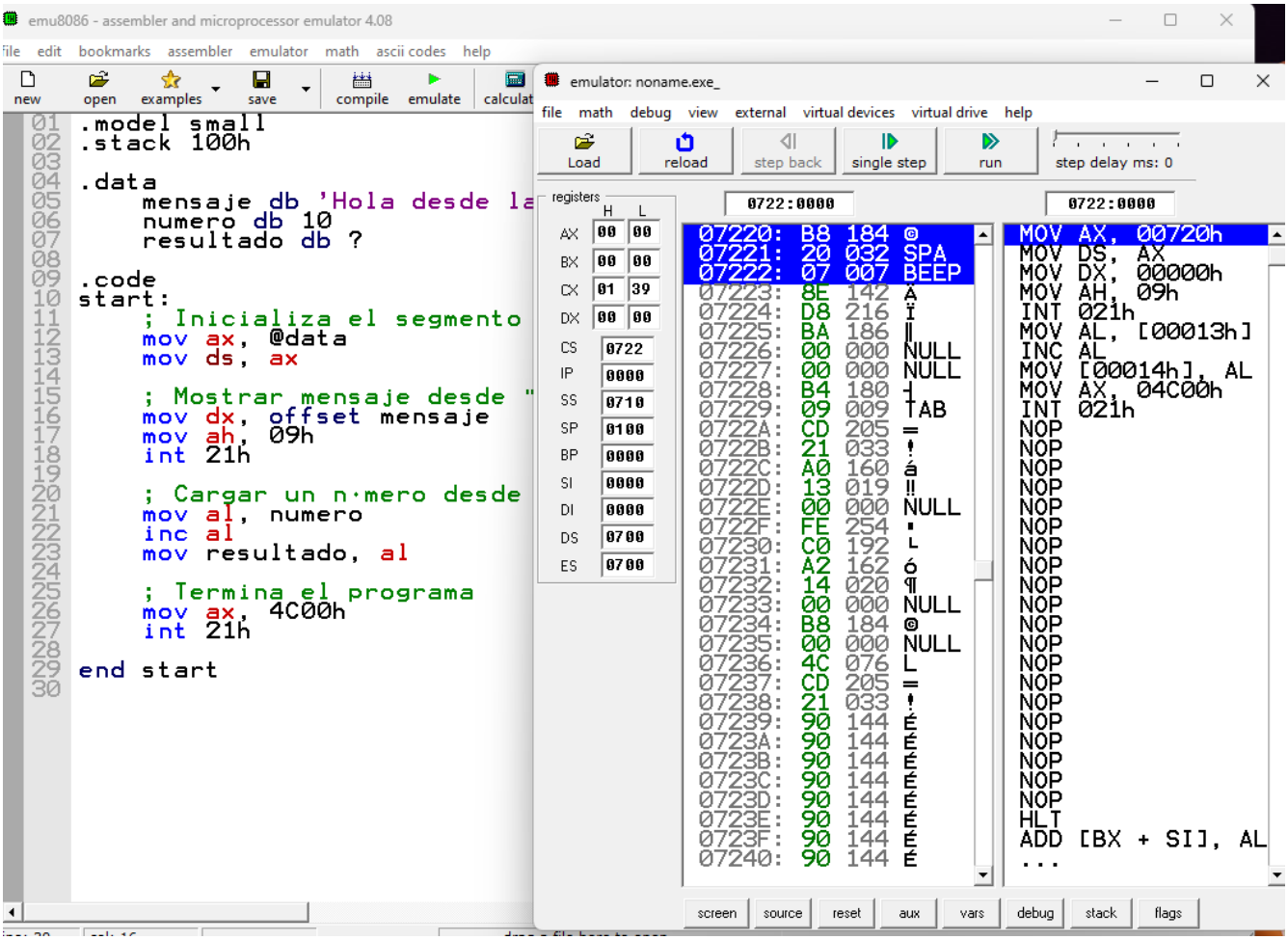
Categoría	Similitudes	Diferencias
Carga de Operandos	Ambos cargan operandos en registros antes de operar.	
Control por la UC	Ambas son iniciadas por la Unidad de Control.	
Almacenamiento de Resultados	Ambos almacenan resultados en registros.	
Actualización de Bandera	Ambas actualizan las banderas de estado.	
Tipo de Circuitos		La suma usa circuitos de suma que manejan acarreos; la lógica usa circuitos lógicos que operan bit a bit.
Naturaleza		La suma es aritmética: combina valores; AND es lógica: evalúa condiciones.
Desbordamiento		La suma puede tener desbordamiento si el resultado excede el tamaño del registro; la lógica no, pues se evalún condiciones binarias.

Memoria

Cargar y Almacenar Datos en Diferentes Tipos de Memoria (RAM y ROM)

Tenemos el siguiente código:

```
.model small
.stack 100h

.data
    mensaje db 'Hola desde la ROM!$'  ; Mensaje almacenado en la sección de datos (ROM)
    numero db 10 ; Valor inicial en RAM
    resultado db ? ; Variable para almacenar el resultado

.code
start:
    ; Inicializa el segmento de datos
    mov ax, @data
    mov ds, ax

    ; Mostrar mensaje desde "ROM"
    mov dx, offset mensaje ; Cargar la dirección del mensaje en DX
    mov ah, 09h ; Función para imprimir cadena en DOS
    int 21h ; Llamar a la interrupción para imprimir

    ; Cargar un número desde RAM, incrementarlo y almacenar el resultado en RAM
    mov al, numero ; Cargar el valor desde RAM en AL
    inc al ; Incrementar el valor
    mov resultado, al ; Almacenar el resultado de vuelta en RAM

    ; Termina el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```



Diferencias entre RAM y ROM

Característica	RAM	ROM
Velocidad	Generalmente más rápida, permite acceso aleatorio a cualquier ubicación.	Más lenta en comparación con la RAM; acceso secuencial en muchos casos.
Accesibilidad	Los datos pueden ser leídos y escritos; permite modificaciones.	Generalmente solo se puede leer; no se puede modificar fácilmente en operación.
Volatilidad	Volátil; pierde datos al apagar el sistema.	No volátil; conserva datos incluso sin energía.
Uso Común	Almacena datos temporales y variables de programas en ejecución.	Almacena firmware, instrucciones de arranque y datos permanentes.

Bus de Datos

Transferencia de Datos entre la Memoria y la CPU a través del Bus de Datos

```
.model small
.stack 100h

.data
    valor db 5 ; Variable en memoria (RAM)
    resultado db ? ; Variable para almacenar resultado

.code
start:
    ; Inicializa el segmento de datos
    mov ax, @data
    mov ds, ax

    ; Cargar valor de la memoria (RAM) en el registro AL
    mov al, valor ; Mover el contenido de 'valor' a AL

    ; Realizar una operación (incrementar el valor)
    inc al ; Incrementar el valor en AL

    ; Almacenar el resultado de vuelta en la memoria
    mov resultado, al ; Mover el contenido de AL a 'resultado'

    ; Termina el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```

Pasos, Patrones de Transferencia de Datos y Rendimiento

Paso	Descripción	Patrón de Transferencia	Impacto en el Rendimiento
Inicialización del Segmento de Datos	Se inicializa el segmento de datos cargando la dirección en **AX** y **DS**.		Configura la memoria para el acceso, estableciendo el contexto de ejecución.
Cargar Valor desde la Memoria	**`mov al, valor`:** Se carga el valor desde la dirección de memoria **`valor`** a **AL.**	Lectura desde RAM	Acceso secuencial; eficiente si el dato está en la caché.
Incrementar el Valor en el Registro	**`inc al`:** Se incrementa el valor en **AL**.		Operación interna en la CPU; muy rápida y eficiente.
Almacenar el Resultado en la Memoria	**`mov resultado, al`**: Se almacena el contenido de **AL** en **`resultado`**.	Escritura en RAM	Escritura secuencial; eficiente si se minimiza la latencia de acceso.
Terminar el Programa	**`mov ax, 4C00h`** y **`int 21h`:** Código de salida que termina la ejecución.		Finaliza la ejecución de manera ordenada, liberando recursos.

Ciclo de Instrucción

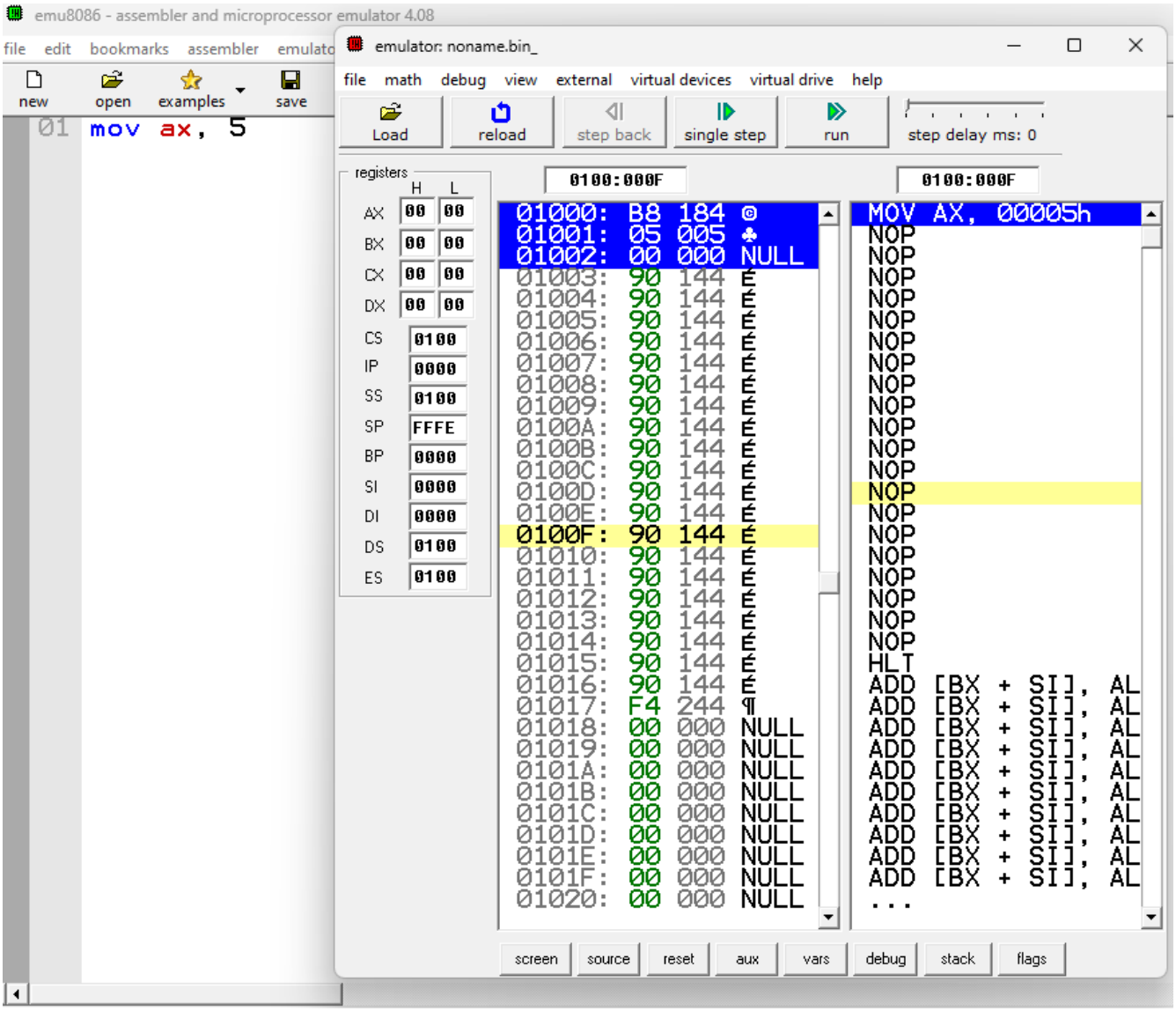
Ciclo Completo de Instrucción (fetch – decode – execute)

Fetch

En esta etapa, la CPU captura la instrucción desde la memoria. El contador de programa (PC) señala la dirección de la próxima instrucción a ejecutar. La CPU lee la instrucción en esa dirección de memoria.

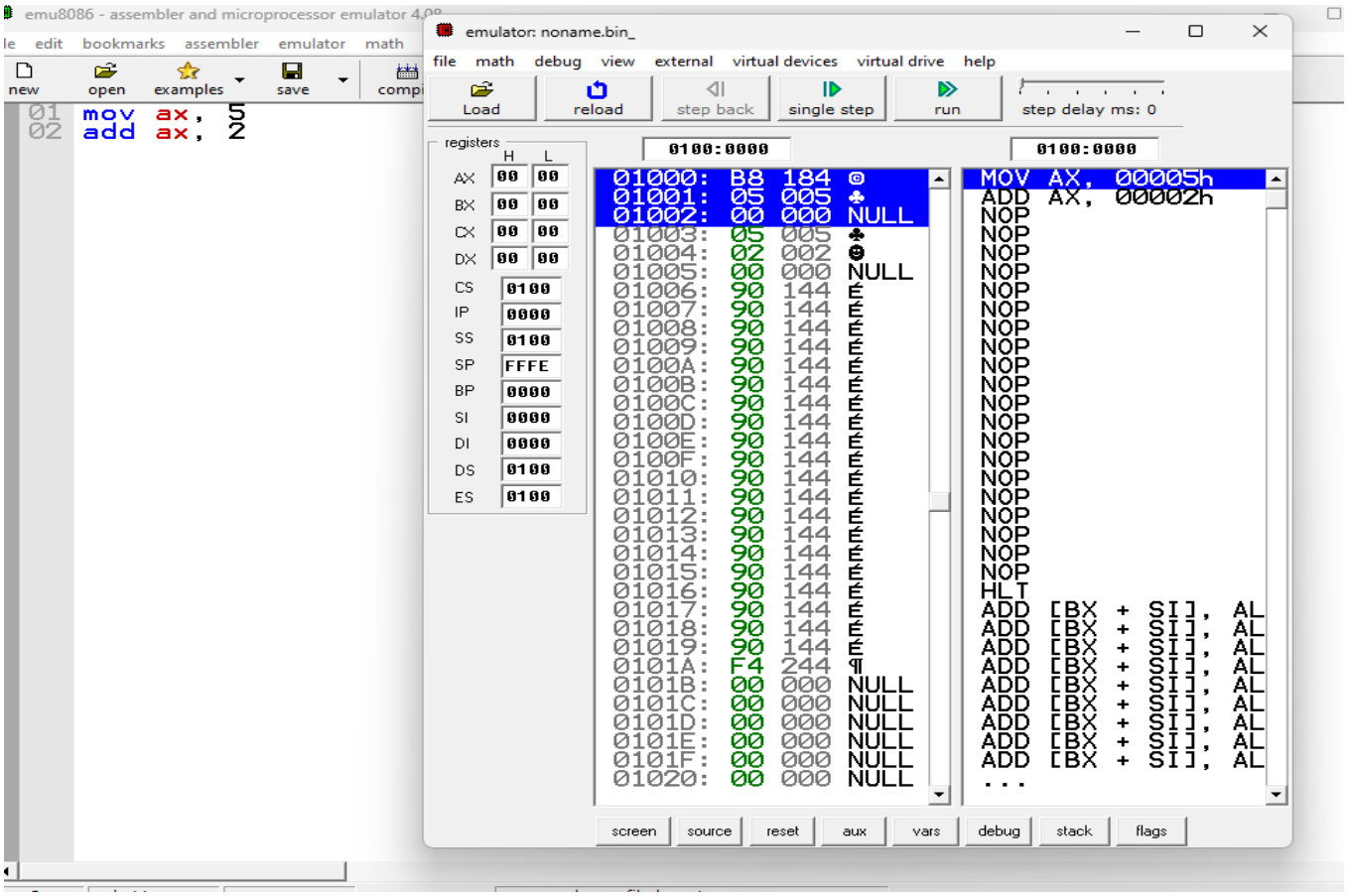
Código de ejemplo:

mov ax, 5 ; Instrucción a capturar



Decode

La CPU decodifica la instrucción capturada para determinar qué operación realizar. La Unidad de Control interpreta la instrucción. Se identifican los operandos necesarios para la operación.

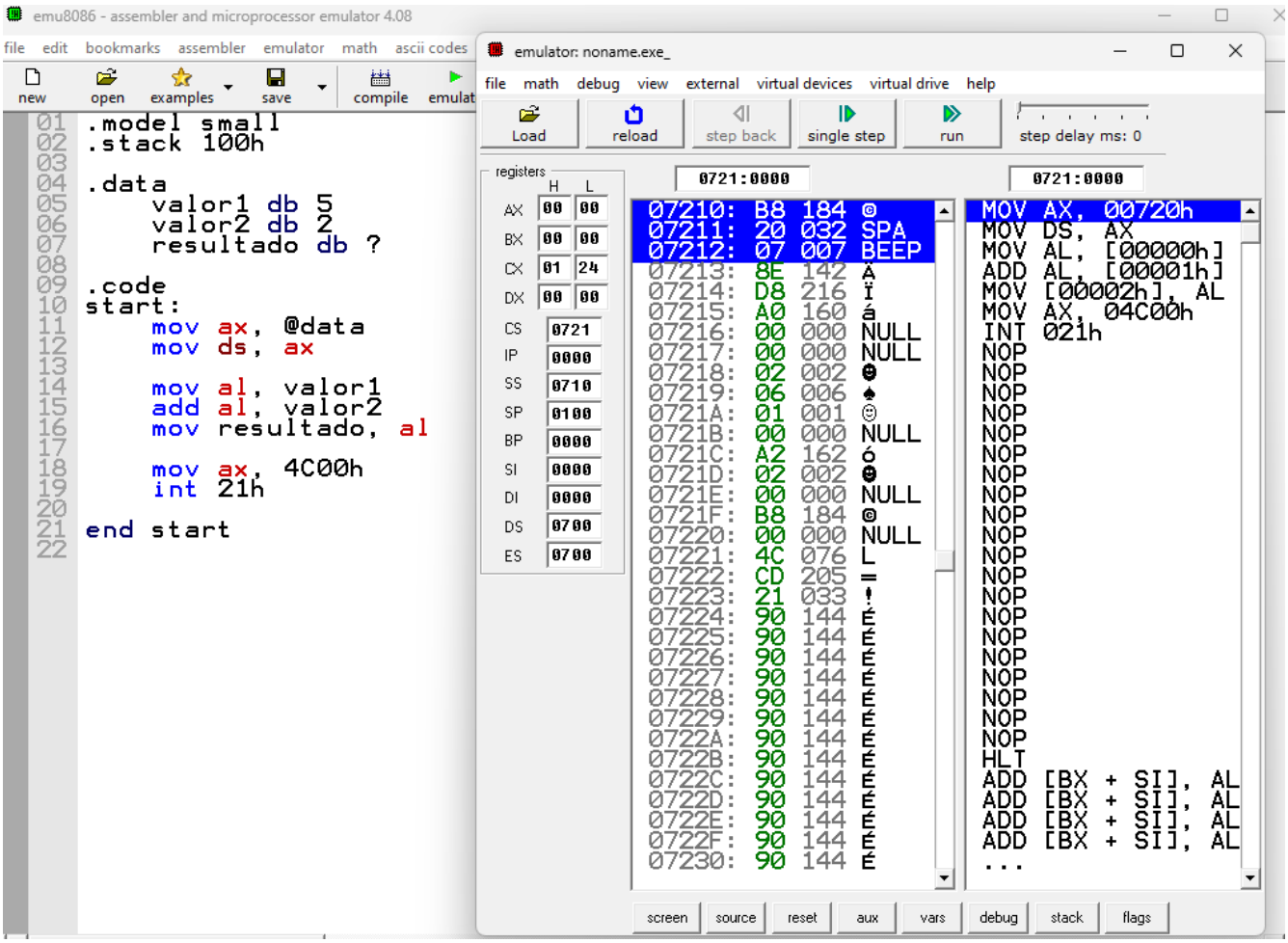


Execute

En esta etapa, la CPU ejecuta la instrucción. La ALU realiza la operación correspondiente. Se actualizan los registros y la memoria según sea necesario.

```
.model small
.stack 100h

.data
    valor1 db 5 ; Primer valor
    valor2 db 2 ; Segundo valor
    resultado db ? ; Variable para almacenar resultado
```

```
.code
start:
    mov ax, @data ; Inicializa el segmento de datos
    mov ds, ax ; Carga DS

    mov al, valor1 ; Fetch: Cargar valor1 en AL
    add al, valor2 ; Fetch: Añadir valor2 a AL
    mov resultado, al ; Almacenar resultado

    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```



Etapas del Ciclo de Instrucción y su Importancia en la Ejecución de un Programa

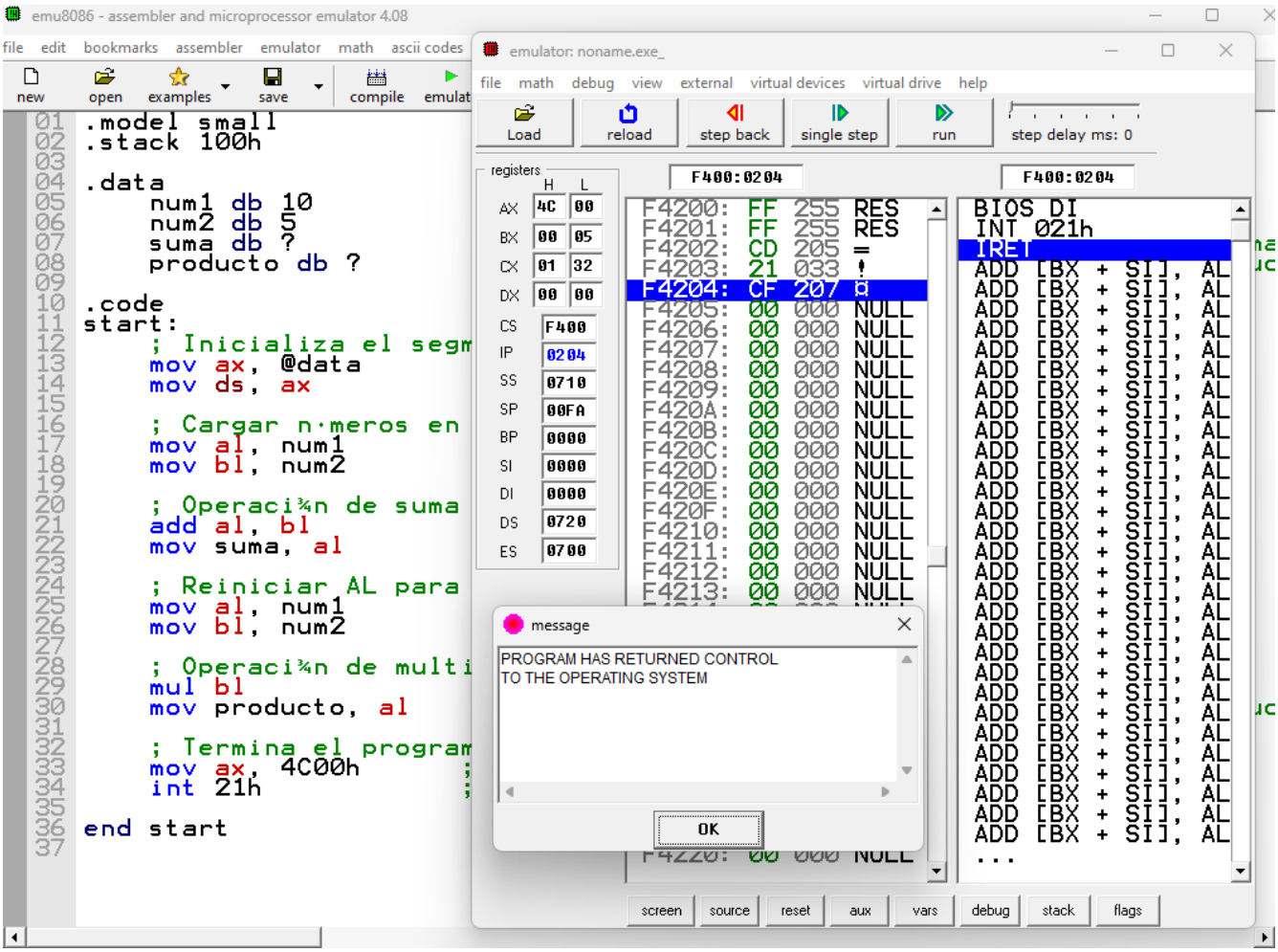
Etapa	Descripción	Importancia	Reflejo en el Simulador
Fetch	La CPU captura la instrucción desde la memoria.	Determina la próxima operación que se debe ejecutar.	Visualización del contador de programa (PC) y el registro de instrucción (IR) mostrando la instrucción capturada.
Decode	La Unidad de Control interpreta la instrucción capturada.	Identifica los tipos de operación y los operandos necesarios para la ejecución.	Indicación de la instrucción actual y preparación de registros. Posibles visualizaciones de las señales emitidas por la Unidad de Control.
Execute	Se lleva a cabo la operación correspondiente según la instrucción.	Realiza la acción requerida, actualizando datos en registros o memoria.	Actualización visual de los registros mostrando el resultado de la ejecución. Se puede observar el estado de los registros después de operaciones como suma o movimiento de datos.

Registros

Realizar Operaciones Rápidas y Eficientes

```
.model small
.stack 100h

.data
    num1 db 10 ; Primer número
    num2 db 5 ; Segundo número
    suma db ? ; Variable para almacenar el resultado de la suma
    producto db ? ; Variable para almacenar el resultado del producto

.code
start:
    ; Inicializa el segmento de datos
    mov ax, @data
    mov ds, ax

    ; Cargar números en registros
    mov al, num1 ; Cargar el primer número en AL
    mov bl, num2 ; Cargar el segundo número en BL

    ; Operación de suma
    add al, bl ; AL = AL + BL (suma)
    mov suma, al ; Almacenar el resultado de la suma en 'suma'

    ; Reiniciar AL para la multiplicación
    mov al, num1 ; Cargar el primer número de nuevo en AL
    mov bl, num2 ; Cargar el segundo número en BL

    ; Operación de multiplicación
    mul bl ; AL = AL * BL (multiplicación)
    mov producto, al ; Almacenar el resultado del producto en 'producto'
```

```
    ; Termina el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```



Influencia de los Registros en la Eficiencia de la Ejecución de Instrucciones

- Acceso rápido: los registros están ubicados dentro de la CPU, lo que permite un acceso mucho más rápido en comparación con la memoria principal (RAM). Esto reduce el tiempo que la CPU necesita para acceder a los datos, lo que mejora la velocidad de ejecución de las instrucciones.
- Minimización de accesos a memoria: al utilizar registros para almacenar datos temporales y resultados intermedios, se disminuye la necesidad de realizar múltiples

accesos a la memoria. Cada acceso a la memoria implica un tiempo de espera, mientras que las operaciones en registros son casi instantáneas.

- Operaciones de pila: los registros también se utilizan en operaciones de pila (stack), donde se almacenan direcciones de retorno y datos temporales. Esto mejora la eficiencia de las llamadas a funciones y las operaciones de retorno.
- Reducción de Instrucciones: al usar registros, muchas operaciones pueden combinarse, lo que reduce la cantidad total de instrucciones necesarias. Esto es especialmente beneficioso en bucles y cálculos iterativos.

Ejemplos de Operaciones que se Benefician de los Registros

- Suma de valores: En lugar de cargar un valor desde la memoria cada vez que se necesita realizar una operación, se carga una vez en un registro y se realiza la operación directamente. Por ejemplo:

```
mov al, [valor1] ; Cargar valor1 en AL desde memoria
add al, [valor2] ; Sumar valor2 desde memoria (más lento)
mov [resultado], al ; Almacenar resultado en memoria
```

Mejorado usando registros:

```
mov al, valor1 ; Cargar valor1 en AL
mov bl, valor2 ; Cargar valor2 en BL
add al, bl ; Sumar AL y BL (rápido)
mov resultado, al ; Almacenar resultado en memoria
```

- Contador de bucles: Cuando se implementan bucles, el uso de registros como contadores (por ejemplo, CX) mejora la eficiencia al permitir decrementos directos en lugar de acceder repetidamente a la memoria. Ejemplo:

```
mov cx, 10 ; Cargar contador en CX
bucle:
  ; Realizar operaciones
  loop bucle ; Decrementar CX y repetir (rápido)
```

- Operaciones lógicas: Las operaciones lógicas (AND, OR, XOR) se pueden realizar directamente en registros, lo que es más eficiente que realizar esas operaciones en memoria. Ejemplo:

```
mov al, 0Fh ; Cargar 0Fh en AL
and al, 0Fh ; Realizar AND (rápido)
```

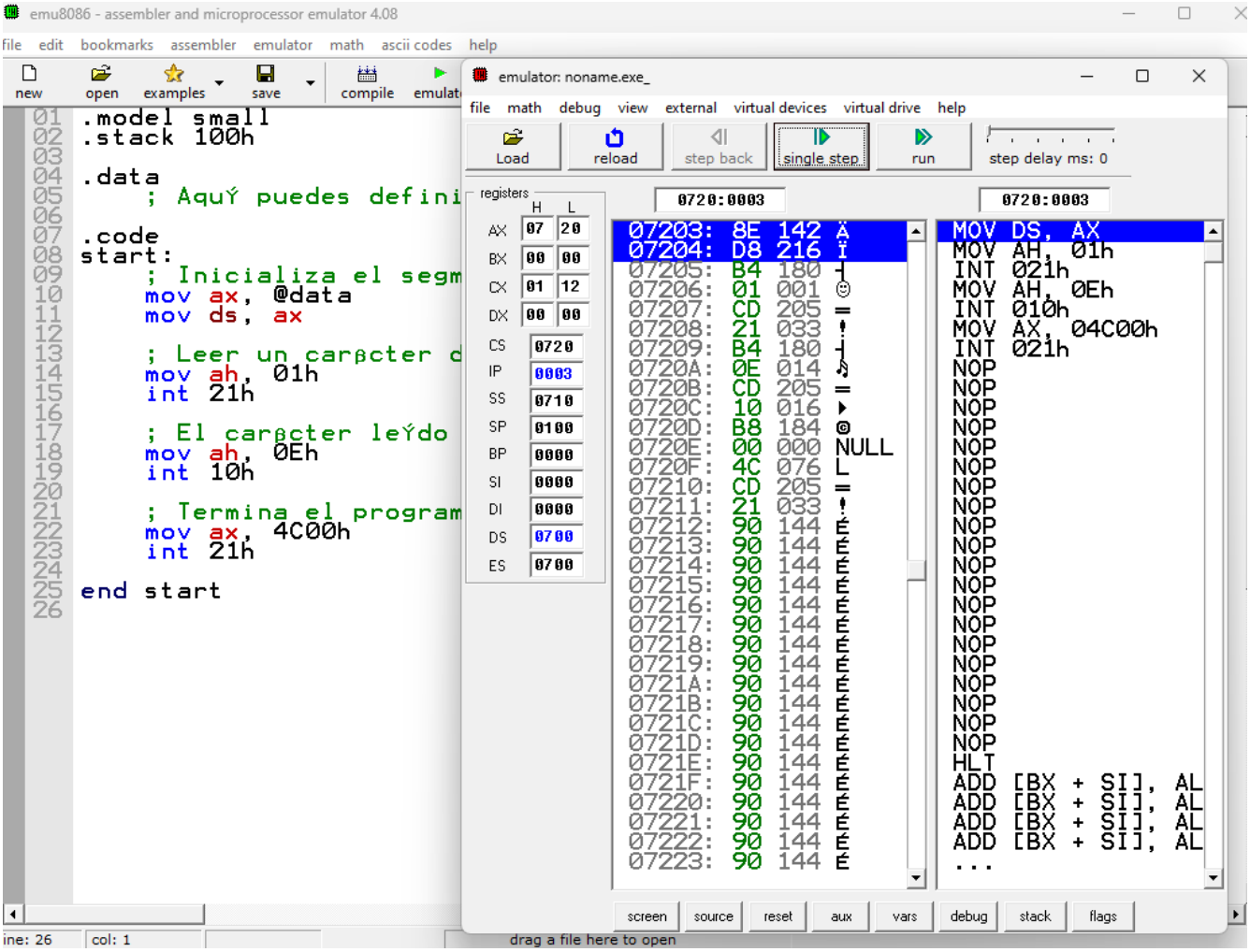
Interconexiones y Módulo de Entrada/Salida (E/S)

Comunicación entre la CPU y un Dispositivo E/S

A continuación tenemos un código que hace uso de la interrupción de BIOS para leer un carácter del teclado y mostrarlo en la pantalla.

```
.model small
.stack 100h

.data
.code
start:
    ; Inicializa el segmento de datos
    mov ax, @data
    mov ds, ax

    ; Leer un carácter del teclado
    mov ah, 01h ; Función 01h para leer un carácter
    int 21h ; Llamar a la interrupción 21h (teclado)

    ; El carácter leído está en AL, mostrarlo en la pantalla
    mov ah, 0Eh ; Función 0Eh para mostrar un carácter
    int 10h ; Llamar a la interrupción 10h (video)

    ; Termina el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```

La utilización de interrupciones permite que la CPU maneje estas operaciones de forma eficiente, liberando recursos para otras tareas mientras espera que el usuario interactúe. Esta forma de comunicación es fundamental en sistemas operativos y en la programación de bajo nivel.



Manejo de la Comunicación entre la CPU y los Dispositivos E/S

Aspecto	Descripción
Interrupciones	Permiten que la CPU pause su ejecución actual y responda a eventos externos, como la entrada desde el teclado o la finalización de una operación de E/S.
Instrucciones de E/S	Instrucciones específicas que permiten a la CPU enviar o recibir datos a través de dispositivos. Ejemplo: **`int 21h`** para operaciones de teclado y archivos.
Bus de Datos	Conjunto de líneas de comunicación que transportan datos, direcciones y señales de control entre la CPU y los dispositivos de E/S.
Controlador de E/S	Intermediario entre la CPU y el dispositivo que gestiona operaciones de lectura/escritura y genera interrupciones para informar a la CPU.

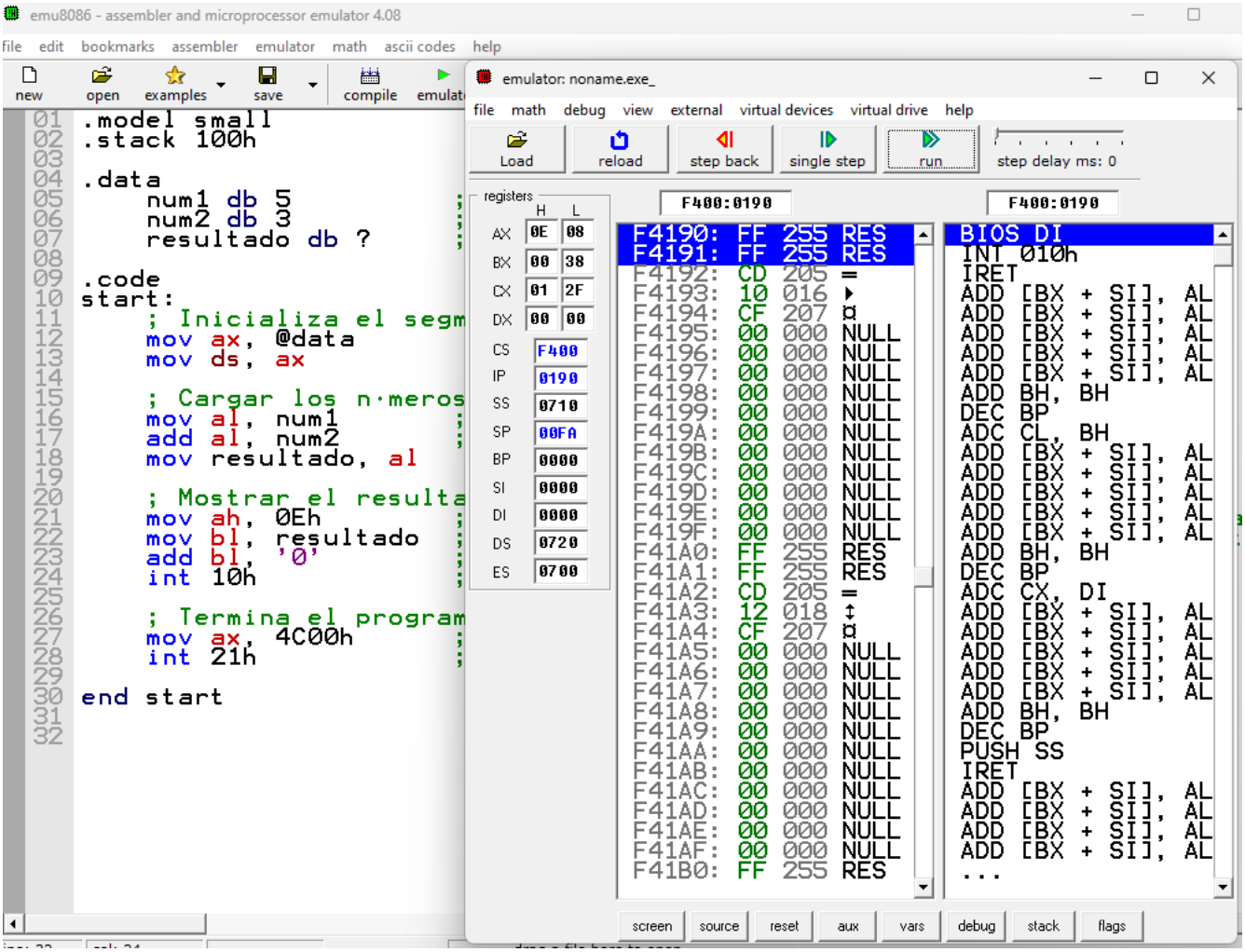
Importancia de las Interconexiones

Aspecto	Descripción
Eficiencia	Permiten una transferencia rápida de datos entre la CPU y los dispositivos, evitando cuellos de botella en la comunicación.
Modularidad	Facilitan la incorporación de múltiples dispositivos de E/S a través del mismo bus, permitiendo la expansión del sistema sin complicaciones.
Sincronización	Ayudan a coordinar las operaciones de E/S mediante señales de control, asegurando una correcta transmisión y recepción de datos.
Flexibilidad	Permiten que sistemas operativos y controladores gestionen dispositivos de manera eficiente y flexible, facilitando la comunicación y el control de una variedad de E/S.

Almacenamiento de Programas

Almacenamiento en Memoria Principal

```
.model small
.stack 100h

.data
    num1 db 5 ; Primer número a sumar
    num2 db 3 ; Segundo número a sumar
    resultado db ? ; Variable para almacenar el resultado

.code
start:
    ; Inicializa el segmento de datos
    mov ax, @data
    mov ds, ax

    ; Cargar los números en registros
    mov al, num1 ; Mover num1 a AL
    add al, num2 ; Sumar num2 a AL
    mov resultado, al ; Almacenar el resultado

    ; Mostrar el resultado en pantalla
    mov ah, 0Eh ; Función para mostrar un carácter en la pantalla
    mov bl, resultado ; Mover el resultado a BL para convertir a carácter
    add bl, '0' ; Convertir el número a carácter ASCII
    int 10h ; Llamar a la interrupción de video para mostrar el carácter

    ; Termina el programa
    mov ax, 4C00h ; Código de salida
    int 21h ; Interrupción de DOS para salir

end start
```



Análisis del Modelo de Von Neuman Basado en el Código Implementado

En el código, tanto los números (**num1** y **num**2) como las instrucciones del programa se almacenan en la misma sección de memoria. Esto simplifica el acceso a los datos requeridos para las operaciones, ya que la CPU puede leer tanto las instrucciones como los datos de la misma memoria. Al compilar y ejecutar el código en EMU8086, se puede observar cómo se accede a los números en la sección de datos directamente durante la ejecución de las instrucciones.

El programa puede ser fácilmente modificado para cambiar los números a sumar. Por ejemplo, si se quiere cambiar **num1** a 10, simplemente se actualiza la línea **num1 db 5** a **num1 db 10**. Realizar cambios y ver inmediatamente el efecto en el resultado refuerza la flexibilidad que ofrece el modelo de von Neumann.

El ciclo de instrucción (fetch-decode-execute) es eficiente porque las instrucciones de suma y el almacenamiento del resultado se ejecutan en un flujo continuo. La CPU carga

primero el número de la memoria, luego lo suma y finalmente almacena el resultado, todo sin interrupciones innecesarias. Al observar el flujo de ejecución en el simulador, se puede notar cómo cada paso del ciclo de instrucción se realiza de manera secuencial y clara.

La secuencia de instrucciones es clara y directa. Las operaciones aritméticas se llevan a cabo utilizando registros, lo que hace que el proceso sea más rápido y eficiente. Al ejecutar el programa, la simplicidad del flujo de instrucciones permite una comprensión fácil de cómo los registros manejan la suma.

El uso de registros (como **AL** para almacenar temporalmente el resultado de la suma) demuestra la eficiencia en la utilización de recursos. Esto evita accesos innecesarios a la memoria principal, acelerando la ejecución. Al ver cómo se mueve el resultado entre registros y memoria, se puede apreciar cómo el modelo de von Neumann optimiza la ejecución del programa.

El modelo de von Neumann proporciona numerosas ventajas en la ejecución de programas almacenados en la memoria principal. En el contexto del código utilizado, se demuestra cómo el almacenamiento unificado de datos e instrucciones facilita el acceso y la manipulación, la flexibilidad permite modificaciones sencillas y el ciclo de instrucción eficiente asegura una ejecución rápida y comprensible. La experiencia en el simulador EMU8086 refuerza estos conceptos, mostrando claramente cómo funcionan juntos para lograr un procesamiento efectivo en la arquitectura moderna de computadoras.

CONCLUSIÓN

Los ejercicios realizados en el simulador EMU8086 han proporcionado una visión práctica y profunda de los principios del modelo de Von Neumann en acción. A través de estos experimentos, se ha podido observar el papel central que desempeña la Unidad de Control en el ciclo de instrucción. Esta unidad no solo orquesta el flujo de las instrucciones, sino que también garantiza que cada paso del proceso de ejecución se realice de manera eficiente y en el orden correcto. La importancia de la UC es crucial, ya que su función de decodificación y ejecución permite que la CPU interprete y procese las instrucciones sin ambigüedad.

En adición, se ha analizado el uso de registros en la CPU, los cuales son fundamentales para la ejecución rápida de operaciones. Los registros permiten que la CPU almacene temporalmente datos e instrucciones, minimizando la necesidad de acceder constantemente a la memoria principal, lo que a su vez mejora el rendimiento general del sistema. Las operaciones aritméticas y lógicas realizadas con la ALU (Unidad Aritmético-Lógica) se benefician considerablemente de esta arquitectura, ya que los registros permiten realizar cálculos sin demoras significativas.

Los experimentos también han resaltado la distinción entre la memoria RAM y ROM en términos de velocidad y accesibilidad. La RAM, siendo volátil y de acceso rápido, permite que la CPU lea y escriba datos de manera eficiente durante la ejecución de un programa, mientras que la ROM, siendo no volátil, proporciona almacenamiento permanente para instrucciones y datos que no requieren cambios. Esta diferencia es fundamental en la operatividad del sistema, afectando directamente la velocidad de ejecución y el rendimiento del mismo.

Finalmente, se ha evaluado cómo los patrones de transferencia de datos entre la CPU y la memoria impactan el rendimiento general del sistema. La comprensión de estos patrones es esencial para el diseño eficiente de sistemas, ya que optimizar la comunicación y minimizar los cuellos de botella puede mejorar significativamente la capacidad de procesamiento. A lo largo de esta investigación, ha quedado claro que el modelo de Von Neumann no solo fundamenta la arquitectura de las computadoras modernas, sino que su relevancia perdura en la manera en que se abordan y resuelven los desafíos en el diseño y la implementación de sistemas informático

BIBLIOGRAFÍA

@aviviano, @alexbuckgit, @mhopkins-msft (2024) **¿Qué es un controlador?** Artículo digital. Recuperado en septiembre de 2024 de https://learn.microsoft.com/es-es/windows-hardware/drivers/gettingstarted/what-is-a-driver-

Carralero, K., Cedeño, A., Espí, R., Marrero, I., Moreno, A., Parra, Y. (2011). **Pasarela de datos natural sobre estándar TETRA.** Recuperado en octubre de 2024 de https://www.researchgate.net/publication/295842997_Pasarela_de_datos_natural_sobre_estandar_TETRA

Facultad de Informática (S/F) **Estructura de Computadores**, Madrid – España. Universidad Complutense de Madrid.

Larraza, E. (2013) **The Computer Input/Output Subsystem Education in an undergraduate introductory course: a Multiperspective Study.** Bilbao – España. Servicio Editorial de la Universidad del País Vasco.

Mano M. (1985), **Lógica digital y diseño de computadores**, Juárez - México, Editorial Prentice Hall Inc.

Santamaría E. (1993), **Electrónica digital y microprocesadore***s,* España, Editado por Universidad Pontificia Comillas.

Stallings, W. (2006) **Organización y Arquitectura de Computadores.** Madrid – España. Pearson Prentice Hall.

Valvano J. W (2004), **Introducción a los sistemas de microcomputadores Empotrados**, México, Editorial Cengage Learning.