



FACULTAD DE CIENCIAS DE LA COMPUTACIÓN Y DISEÑO DIGITAL

INTEGRANTES

HERRERA MORAN GILMAR JAVIER

MENDOZA PARRAGA ANDY JOHEL

ZAMORA AGUILAR RONALDO WILFRIDO

CURSO

2DO SOFTWARE “B”

GRUPO

F

MATERIA

ARQUITECTURA DE COMPUTADORAS

TEMA

COMPONENTES DEL COMPUTADOR

ÍNDICE

OBJETIVO.....	3
FUNDAMENTO TEÓRICO.....	3
1. Memoria.....	3
2. Memoria Caché, Interna y Externa.....	4
3. Entrada/Salida (E/S).....	5
3.1. Conceptos básicos de Entrada y Salida.....	6
3.2. Interacción entre CPU, memoria y dispositivos de E/S.....	6
3.3. Módulos de E/S.....	7
3.4. Métodos de Entrada/Salida.....	7
4. Interconexión con Buses.....	8
4.1. Elementos del diseño del bus.....	9
4.2. Tipos de buses.....	9
4.2.1. Bus de datos.....	9
4.2.2. Bus de direcciones.....	9
4.2.3. Bus de control.....	9
4.3. Rol de los buses en la transferencia de datos entre componentes.....	10
PROCEDIMIENTOS.....	10
CONCLUSIÓN.....	22
BIBLIOGRAFÍA.....	22
ANEXO.....	24

OBJETIVO

Conocer y analizar los principales componentes de un computador, incluyendo diferentes tipos de memoria, dispositivos de entrada/salida y la interconexión mediante buses, para comprender su función y comportamiento en el sistema computacional.

FUNDAMENTO TEÓRICO

1. Memoria

La memoria en las computadoras es el conjunto de recursos que almacenan instrucciones y datos para que la CPU y otros dispositivos los puedan procesar. Desde el punto de vista del diseño de sistemas, la memoria no trabaja por si sola ya que esta incluye niveles con distintas latencias, anchos de banda y persistencia (registros, caché L1/L2/L3, memoria principal DRAM y almacenamiento secundario). Entender los niveles y sus características es importante para optimizar software y hardware, porque muchas optimizaciones de rendimiento resultan de ajustar la localidad de datos a la jerarquía de memoria [1].

Los tipos de memoria usados en computadores se diferencian por capacidad, velocidad y persistencia. En la cima de todo están los registros que son los más rápidos y de menor capacidad, la caché que funciona en microsegundos o nanosegundos de latencia; en el medio está la memoria interna (RAM/DRAM) con mayor capacidad y latencia; al final está la memoria externa (SSD, HDD, almacenamiento en red) que ofrece mucha capacidad y persistencia, pero con latencias y anchos de banda peores. Además, aparecen tecnologías emergentes (NVM, PCM, ReRAM) y técnicas near-memory / processing-in-memory (PIM) que buscan reducir el coste de mover datos entre niveles [2].

La jerarquía de memoria organiza esos niveles por latencia y capacidad para mejorar la velocidad de acceso, los niveles superiores (caché) ejecutan datos rápidamente y filtran solicitudes hacia niveles más lentos solo cuando es necesario. Modelos analíticos y de colas han demostrado cómo el número de niveles y sus parámetros (latencias, tasas de aciertos/fallos, ancho de banda compartido) afectan la respuesta del sistema y las variaciones en el servicio, esto es especialmente crítico en arquitecturas multicore donde la memoria es compartida y los efectos de contención aumentan la variabilidad del tiempo de respuesta [3].

2. Memoria Caché, Interna y Externa

La memoria caché tiene funciones y mecanismos específicos: mantener copias de bloques de memoria más utilizados, reducir latencia efectiva y disminuir la presión sobre la RAM/almacenamiento. Las políticas de reemplazo (LRU, LFU, políticas híbridas) y las estrategias de coherencia en caches compartidas influyen fuertemente en la tasa de aciertos y en su rendimiento. Trabajos recientes proponen caches in-memory con privación dinámica que cambian los tiempos de ejecución para minimizar la tasa de fallos, mostrando mejoras reales en entornos de almacenamiento y sistemas distribuidos [4].

Al comparar memoria interna (RAM) y memoria externa (almacenamiento secundario) destacan diferencias funcionales: la RAM es volátil, optimizada para acceso aleatorio rápido y altas tasas de transferencia en acceso a datos activos; el almacenamiento externo es no volátil y está optimizado para capacidad y coste por cada byte. Sin embargo, nuevas arquitecturas de almacenamiento computacional y de memoria cercana estrechan la distancia entre ambos, moviendo parte del procesamiento hacia el almacenamiento o acercándose a la memoria para reducir movimiento de datos y mejorar latencias o consumo energético en aplicaciones intensivas en datos [5].

La importancia de la caché en el rendimiento se cuantifica con métricas como las tasas de acierto y tiempo de acceso efectivo (EAT). Modelos y simuladores muestran que pequeñas mejoras en tasa de aciertos o en la política de reemplazo pueden dar reducciones significativas del tiempo promedio de acceso y carga sobre la RAM y el almacenamiento, especialmente en cargas concurrentes (multicore, DNNs). Por eso, el co-diseño de software (localidad de datos, optimizaciones de acceso) y hardware (tamaños de línea, niveles de caché, ancho de banda) es una práctica recurrente para mejorar el rendimiento y la latencia [6].

Las tendencias emergentes modifican el panorama, ya sea en el procesamiento en memoria (PIM), computación en memoria y dispositivos con capacidades especiales integradas proponen reducir la transferencia de datos entre CPU y memoria, mejorar la eficiencia energética y aumentar el rendimiento para tareas específicas (p. ej. inferencia de redes neuronales u operaciones de reducción en grandes volúmenes de datos). Investigaciones en materiales y dispositivos (ferroeléctricos, memoria no volátil) y revisiones de arquitecturas PIM señalan tanto prometedoras ganancias de rendimiento como retos (programabilidad, coherencia, herramientas de verificación) [7].

Para investigación y medición práctica, los trabajos de conferencias sobre sistemas de memoria (actas MEMSYS) y artículos recientes sobre caches y dispositivos de almacenamiento computacional recomiendan usar simuladores detallados (modelado de canales, ranks, banks), benchmarks orientados a memoria y experimentos controlados con métricas de latencia, rendimiento y perfiles de localidad. Además, los estudios de revisión sistemática sobre computational storage recomiendan definir claramente el caso de uso y medir tanto el impacto en latencia como en consumo energético y coste por operación [8].

3. Entrada/Salida (E/S)

En la arquitectura de computadoras, la conexión entre el teclado/monitor (E/S), la CPU y la memoria es muy importante para el funcionamiento y uso de cualquier sistema. Esta interacción no solo depende de cómo pasa la información física y lógicamente, sino también de cómo se organizan y hacen mejor los procesos para que la información llegue de la forma más eficiente posible. Un caso ilustrativo de la importancia de un planteamiento adecuado lo encontramos en Structured Input-Output Tools for Modal Analysis of a Transitional Channel Flow, en el que se utilizan herramientas estructuradas de E/S para el análisis de flujos transicionales complejos. Es en este trabajo en el que disponer de un sistema de E/S que opera de manera adecuada nos permite capturar y detectar de manera precisa los puntos existentes y las estructuras coherentes, sacando todo el partido de la capacidad de cálculo sin tirar recursos por errores [9].

Por otro lado, Advances in Microprocessor Cache Architectures Over the Last 25 Years traza el desarrollo de las arquitecturas de caché en microprocesadores que han cambiado las reglas de cómo los datos son transferidos de la CPU a la memoria principal y los periféricos. La inteligente introducción de niveles de caché (L1, L2, L3) y técnicas avanzadas de particionamiento han conseguido reducir este retardo por un lado y, por otro, mejorar el rendimiento de las aplicaciones, especialmente en escenarios en que es necesario aumentar la capacidad de almacenamiento y al mismo tiempo ofrecer un rápido acceso a un gran volumen de datos [10].

En su conjunto, ambos métodos muestran un mismo principio: el trabajo de los movimientos de E/S necesita soluciones de hardware buenas, bases de datos mejoradas y estructuras para guardar informaciones fuertes. Cuando se trata de imitar un hecho físico, hay que programarlo para usar E/S oculta, E/S que para, acceso recto a memoria (DMA) o algo parecido, luego de pensar cómo se moverá la información por el sistema de manera eficaz. Al juntar lo teórico, el diseño y la practica, nosotros podemos trabajar en sistemas que sean capaces de procesar los

datos más rápido seguro y esperado. Es muy importante para crear tecnología nueva y resolver problemas más difíciles [9], [10].

3.1. Conceptos básicos de Entrada y Salida

En el ámbito de los computadores, la Entrada/Salida (E/S) es el conjunto de transacciones, formas y dispositivos que un sistema permite operar con los objetos de fuera, mezclando datos entre la CPU, la memoria y los dispositivos conectados. Este proceso es el fundamento para que los programas se puedan beneficiar de datos procedentes de sensores, soportes, redes u otros sistemas, para luego producir salidas a raíz del almacenamiento, la representación o la remisión de datos. La E/S es por ello el vínculo entre el funcionamiento interno de la máquina y las fuentes o destinos que ocupan información exterior [11].

En sistemas de alto rendimiento, como los que ADIOS2 gestiona, la entrada/salida es crucial por la enorme cantidad de datos de simulaciones científicas, análisis masivos y computación distribuida. Aquí, los conceptos básicos de E/S incluyen la comunicación con dispositivos, la optimización de velocidad, la baja latencia y la gestión de grandes flujos de información. En este entorno, el diseño de un sistema de E/S debe pensar en cómo reducir los cuellos de botella y usar al máximo el hardware disponible se pueden clasificar en varias categorías[9], [11]:

- **E/S programada**, donde la CPU controla directamente la transferencia de datos, ejecutando instrucciones específicas para comunicarse con el dispositivo.
- **E/S mediante interrupciones**, que permite a los dispositivos notificar a la CPU cuando están listos para enviar o recibir datos, liberando a la CPU de la espera activa.
- **Acceso Directo a Memoria (DMA)**, que habilita la transferencia de datos entre memoria y dispositivo sin intervención constante de la CPU, mejorando la eficiencia.

Con ello, la E/S deja de ser solo un canal de comunicación básico para convertirse en un componente estratégico que define el rendimiento global de un sistema [11].

3.2. Interacción entre CPU, memoria y dispositivos de E/S

Un ordenador debe permitir a los usuarios o a otros ordenadores encontrar una relación definida y determinada en el sentido establecido antes para poder llevar a cabo el tratamiento de la información que se quiere obtener. La forma en que interactúan el microprocesador, la memoria y la E/S es crucial para el adecuado tratamiento informático de la información. El microprocesador realiza las instrucciones y controla la información, pero la memoria sirve para almacenar temporalmente los datos y los programas necesarios para su funcionamiento. En

cuanto a los periféricos, estos son los elementos que permiten que el sistema intercambie información con el exterior comunicándose, obteniendo información (entrada) o enviándola (salida) [12].

En este proceso, la CPU asume la función de la dirección de todo, controlando con instrucciones precisas las peticiones de entrada y salida; es decir, controla cuándo y cómo se produce la transferencia de datos entre los dispositivos y la memoria. Para mejorar este proceso, se pueden utilizar técnicas como la E/S programada, las interrupciones y el acceso directo a la memoria (DMA), con el objetivo de reducir los tiempos de espera y aumentar el rendimiento del sistema; la intención es conseguir que los datos circulen sin atascos y que la lentitud de un dispositivo no acabe por deteriorar el rendimiento del sistema [12], [13].

3.3. Módulos de E/S

La tarea del controlador de E/S es controlar, dirigir y mantener todo el sistema de periféricos. Entre sus funciones se incluye traducir las señales energéticas o analógicas en información binaria y viceversa. También se encarga de hacer que el ritmo de los dispositivos periféricos, que suelen funcionar más despacio, sea compatible con el de la CPU, que es muy rápido. Esto lo consigue almacenando temporalmente los datos en una memoria llamada buffer cuando los dispositivos periféricos no están preparados para trabajar o bien leyendo los datos a un ritmo mayor cuando lo requiere el dispositivo periférico. El controlador de E/S también tiene que detectar errores y, si los hay, determinar con qué periférico se ha producido el error y qué tipo de error ha sido y, a continuación, informar a la CPU. Para ello, utiliza varios registros especiales que posee, y que la CPU puede examinar y programar, de forma similar a como se hacía con el conjunto de registros del procesador antes de que se iniciara la transferencia [14].

3.4. Métodos de Entrada/Salida

A la hora de diseñar ordenadores, las operaciones de E/S (entrada y salida) se pueden realizar por tres métodos: E/S programada (polling), E/S por interrupción, y la propia memoria (DMA). En la E/S programada o polling, la CPU controla el dispositivo directamente y espera activamente hasta que el dispositivo está preparado para enviar o recibir datos, consultando frecuentemente su registro de estado. Es un método fácil de implementar y sirve bien para dispositivos rápidos o procesos extremadamente cortos, pero es un sistema ineficiente para dispositivos lentos, donde el procesador estará inactivo durante largos períodos en espera. Un caso típico será leer la tecla presionada de una manera que estará comprobando continuamente si hay datos disponibles en el buffer del teclado [13], [14].

Mediante el uso de entrada/salida a través de interrupciones, el CPU no se encuentra sondeando el estado del dispositivo, sino que sigue con otras tareas; cuando el periférico se encuentra listo, éste envía una señal de interrupción que provocará que el CPU detenga provisionalmente la actividad, atienda la solicitud y tras ello vuelva a lo que estaba haciendo. Este procedimiento mejora el funcionamiento de los recursos del procesador; además, le va muy bien a los dispositivos lentos o a aquellas tareas que se producen de forma ocasional, aunque necesita de un cuidado mayor y puede llegar a complicarse en caso de que haya en el sistema un gran número de interrupciones. Un caso habitual es el de una tarjeta de red, que interrumpe al procesador al haber recibido un paquete de datos [10], [14].

Finalmente, la transferencia directa de datos a memoria, o DMA (Direct Memory Access), permite que un controlador específico se encargue de la transferencia de datos entre memoria y dispositivo sin que la CPU tenga que estar siempre ocupada con ese tipo de tarea. El procesador sólo tiene que especificar la dirección de memoria, la dirección de dispositivo y la cantidad de datos que se tienen que transferir en la correcta configuración del controlador DMA y después el DMA se encarga, por sí mismo, de enviar o recibir los datos y efectuar el correspondiente aviso a la CPU. Se trata de un método muy eficiente para mover muchos datos con el hecho soportado de que la CPU queda liberada para ocuparse o realizar otras tareas, aunque requiere hardware adicional y genera competencia por el uso de la memoria. Un ejemplo de este uso sería la transferencia de un archivo desde disco a la RAM [14].

4. Interconexión con Buses

Los buses son canales de comunicación esenciales en la arquitectura de computadoras, ya que estos permiten la transferencia de datos, direcciones y señales de control entre los diferentes componentes. Según Phan y Nguyen (2023), un bus conecta todos los nodos del sistema a través de un protocolo que gestiona las solicitudes y concesiones de información, siendo esta una de las formas de interconexión más sencillos y ampliamente utilizados, tanto en sistemas de un solo núcleo como en los multinúcleos [15].

Por otra parte, Rogowski, los describe como una “autopista electrónica” que ofrece una vía compartida para la transmisión de datos, lo que los hace esenciales para el diseño lógico y la organización física de los sistemas computacionales [16].

4.1. Elementos del diseño del bus

El diseño de buses en sistemas modernos se caracteriza por elementos clave que definen su rendimiento y eficiencia. El ancho del bus determina la cantidad de datos que se pueden transferir de manera simultánea, aumentando el ancho de banda a costa de más líneas físicas en el chip. La velocidad del bus, medida en MHz o GHz, define cuántas transferencias pueden realizarse por segundo. Los protocolos de comunicación como AMBA AXI, establecen reglas para la gestión de acceso compartido, sincronización y modos de transferencia [17].

Según Trevor (2024), estos principios se aplican especialmente en SoCs modernos, donde estos elementos se adaptan a arquitecturas multicore y reconfigurables, donde buses avanzados y redes-on-chip (NoC) permiten conectar CPUs, aceleradores y memorias de forma escalable y eficiente [17].

4.2. Tipos de buses

4.2.1. Bus de datos

El bus de datos es un canal de comunicación que transfiere información entre la CPU, la memoria y los periféricos, manejando comúnmente 32 o 64 bits por ciclo, lo cual determina su ancho de banda. En los inicios de la PC, existían buses de datos en paralelo que conectaban directamente la memoria y los dispositivos, lo que obligaba a todos a operar a la misma velocidad y limitaba el rendimiento. Para dar solución a ese problema, se añadió un controlador que separa la CPU y la memoria de los periféricos, lo que mejoró la velocidad y comunicación entre los componentes. Hoy en día existen dos tipos principales de buses de datos: seriales y paralelos, como ejemplo de los seriales tenemos el USB, y ATA y SCSI para los paralelos [18].

4.2.2. Bus de direcciones

Wang (2021), describe el bus de direcciones como un canal unidireccional que la CPU y otros dispositivos utilizan para indicar una ubicación específica en la memoria o en algún dispositivo. El ancho del bus de direcciones determina la cantidad máxima que se puede direccionar, por ejemplo, un bus de 32 bits puede direccionar hasta 4 Gb de memoria. Algunos sistemas optimizan esto enviando la dirección en dos partes a través de la mitad de las líneas [18].

4.2.3. Bus de control

El bus de control es un conjunto de señales que viajan en ambas direcciones, las cuales la CPU utiliza para enviar comandos y recibir respuestas de otros componentes. También maneja interrupciones y se encarga de coordinar y sincronizar todas las operaciones entre la CPU, la

memoria y los periféricos, asegurando que la información se transfiera en el momento y la forma correctos [18].

4.3. Rol de los buses en la transferencia de datos entre componentes

Los buses actúan como canales dedicados que interconectan los componentes de un sistema computacional, coordinando y optimizando el flujo de datos, con el objetivo de garantizar una comunicación eficiente. En arquitecturas especializadas como aceleradores DNN, estos soportan patrones críticos: uno a muchos para distribución de pesos, y de muchos a uno para recolección de resultados. Su diseño determina el rendimiento global, equilibrando ancho de banda, sincronización y escalabilidad, sobre todo en sistemas paralelos donde la transferencia de datos es esencial para el procesamiento acelerado [19].

PROCEDIMIENTOS

Estudio de Memoria:

- Realizar ejercicios prácticos para calcular tiempos de acceso y capacidad en diferentes niveles de memoria.
- Análisis del rendimiento de la memoria caché mediante ejemplos teóricos y simulaciones.

Operación de Entrada/Salida:

- Implementar ejemplos de E/S programada y E/S mediante interrupciones usando simuladores.
- Analizar el flujo de datos mediante DMA con un ejemplo práctico.

Diseño de Buses:

- Realizar diagramas que representen la estructura y función de un bus.
- Identificar y describir los diferentes tipos de buses y su impacto en la velocidad de transferencia.

PROCEDIMIENTOS

Estudio de Memoria:

- Realizar ejercicios prácticos para calcular tiempos de acceso y capacidad en diferentes niveles de memoria.

Tiempo de acceso.

Tiempo de acceso efectivo (EAT) con una caché L1

Una caché L1 tiene tiempo de acierto $t_{hit} = 1 \text{ ns}$ y tasa de aciertos $\text{hit-rate} = 95\%$. La memoria principal tarda $t_{mem} = 50 \text{ ns}$. Supón que la penalidad por fallo es igual al tiempo de la memoria principal. Calcula el EAT.

$$\text{miss-rate} = 1 - 0,95 = 0,05$$

$$\text{EAT} = \text{hit-rate} \times t_{hit} + \text{miss-rate} \times t_{miss}$$

$$\text{EAT} = 0,95 \times 1 \text{ ns} + 0,05 \times 50 \text{ ns}$$

$$\text{EAT} = 0,95 \text{ ns} + 2,5 \text{ ns}$$

$$\text{EAT} = 3,45 \text{ ns}$$

R.1) El tiempo de acceso efectivo (EAT) es de 3,45 ns.

AMAT en jerarquía de dos niveles

Durante una prueba de rendimiento, un procesador utiliza una jerarquía de memoria compuesta por: Caché L1 con un tiempo de acceso de 1 ns, tasa de aciertos del 91%, una Caché L2 con un tiempo de acceso de 8 ns con una tasa de acierto del 97%, y una memoria principal de acceso de 70 ns.

Determinar el Average Memory Access Time (AMAT) del sistema.

$$\text{AMAT} = h_1 t_1 + (1 - h_1) (h_2 t_2 + (1 - h_2) t_{mem})$$

$$\text{AMAT} = 0,91 \times 1 + (1 - 0,91) (0,97 \times 8 + (1 - 0,97) 70)$$

$$\text{AMAT} = 0,91 + (0,09) (7,76 + (0,03) 70)$$

$$\text{AMAT} = 0,91 + (0,09) (7,76 + 2,10)$$

$$\text{AMAT} = 0,91 + (0,09) (9,86)$$

$$\text{AMAT} = 0,91 + 0,8874$$

$$\text{AMAT} = 1,7974 \text{ ns}$$

R.1) El AMAT es de 1,7974 ns.

Cálculos de parámetros de caché

Se está configurando una memoria caché de 64 KB con bloques de 128 bytes en una arquitectura con direcciones de 32 bits. El mapeo es directo.

Calcular:

- Número de líneas de la caché
- Número de bits de offset
- Número de bits de índice
- Número de bits de etiqueta

$$64 \text{ KB} \times 1024 = 65536 \text{ bytes}$$

Número de líneas

$$\frac{65536}{128 \text{ B}} = 512 \text{ líneas}$$

Bits de offset

$$\log_2(128) = 7 \text{ bits}$$

Bits de índice

$$\log_2(512) = 9 \text{ bits}$$

Bits de etiqueta

$$32 - (9 + 7) = 32 - 16 = 16 \text{ bits.}$$

Capacidad de memoria

Capacidad total sumando niveles

Niveles con tamaño: L1 = 16 KB, L2 = 64 KB, L3 = 128 KB, memoria principal = 8 GB. Suma la capacidad total y exprésala en MiB y en GiB. (usando potencias de 2: 1 KiB = 1024 bytes).

$$1 \text{ MiB} = 1024 \text{ KB}$$

$$L1 = 16 \text{ KB} = \frac{16}{1024 \text{ MiB}} = 0,015625 \text{ MiB}$$

$$L2 = 64 \text{ KB} = \frac{64}{1024 \text{ MiB}} = 0,0625 \text{ MiB}$$

$$L3 = 128 \text{ KB} = \frac{128}{1024 \text{ MiB}} = 0,125 \text{ MiB}$$

$$\text{Memoria} = 8 \text{ GB} = 8 \times 1024 \text{ MiB} = 8192 \text{ MiB}$$

$$8192 + 0,125 + 0,0625 + 0,015625 = 8192,203125 \text{ MiB}$$

$$\frac{8192,203125}{1024} = 8,000198364 \text{ GiB}$$

- Análisis del rendimiento de la memoria caché mediante ejemplos teóricos y simulaciones.

Ejemplo teórico

Un procesador realiza acceso a memoria principal para ejecutar instrucciones y manipular datos. La arquitectura incluye un nivel de caché L1 con un tiempo de acceso de 1ns, mientras que la memoria principal tiene un tiempo de acceso de 50ns.

La tasa de aciertos (hit rate) de la caché es del 92%.

El 8% restante de las referencias se resuelve en la memoria principal (miss rate).

- Calcular el tiempo de acceso medio AMAT.
- Analizar el impacto en el rendimiento si el hit rate aumenta a 96%.

AMAT

Con 92% $AMAT = T_{caché} + (miss_rate \times T_{memoria})$

$$AMAT = 1ns + (0,08 \times 50ns)$$

$$AMAT = 1ns + 4ns$$

$$AMAT = 5ns$$

R. // En promedio, cada acceso a memoria tarda 5ns.

Con el 96% de aciertos

$$AMAT = 1ns + (0,04 \times 50ns)$$

$$AMAT = 1ns + 2ns$$

$$AMAT = 3ns$$

Con la mejora del hit rate al 96%, el tiempo se reduce a 3ns.

Simulación en C# de comportamiento de una memoria caché

Código

```
Program.cs* X
[+] SimulacionCache_GrupoF Program Main()
1 using System;
2 using System.Collections.Generic;
3
4 ~referencias
5 class Program
6 {
7     ~referencias
8     static void Main()
9     {
10         // Caché
11         int[] cache = new int[4]; // 4 bloques de caché
12         for (int i = 0; i < cache.Length; i++)
13         {
14             cache[i] = -1;
15         }
16
17         int hits = 0;
18         int misses = 0;
19
20         // Simulamos accesos a memoria
21         int[] accesos = { 0, 4, 8, 12, 0, 4, 16, 20, 0, 4 };
22
23         foreach (int direccion in accesos)
24         {
25             int bloque = direccion / 4; // Tamaño de bloque = 4 bytes
26             int indice = bloque % cache.Length;
27
28             Console.WriteLine($"Acceso a dirección {direccion} (bloque {bloque})... ");
29
30             if (cache[indice] == bloque)
31             {
32                 hits++;
33                 Console.WriteLine("HIT!");
34             }
35             else
36             {
37                 misses++;
38                 cache[indice] = bloque;
39                 Console.WriteLine("MISS - bloque cargado en caché");
40             }
41         }
42
43         // Resultados
44         Console.WriteLine("\nResumen:");
45         Console.WriteLine($"Total accesos: {accesos.Length}");
46         Console.WriteLine($"Hits: {hits}");
47         Console.WriteLine($"Misses: {misses}");
48         Console.WriteLine($"Tasa de hits: {(float)hits / accesos.Length:P0}%");
49     }
50 }
```

Ejecución

```
C:\Windows\system32\cmd.exe X + v
Acceso a dirección 0 (bloque 0)... MISS - bloque cargado en caché
Acceso a dirección 4 (bloque 1)... MISS - bloque cargado en caché
Acceso a dirección 8 (bloque 2)... MISS - bloque cargado en caché
Acceso a dirección 12 (bloque 3)... MISS - bloque cargado en caché
Acceso a dirección 0 (bloque 0)... HIT!
Acceso a dirección 4 (bloque 1)... HIT!
Acceso a dirección 16 (bloque 4)... MISS - bloque cargado en caché
Acceso a dirección 20 (bloque 5)... MISS - bloque cargado en caché
Acceso a dirección 0 (bloque 0)... MISS - bloque cargado en caché
Acceso a dirección 4 (bloque 1)... MISS - bloque cargado en caché

Resumen:
Total accesos: 10
Hits: 2
Misses: 8
Tasa de hits: 20 %
Presione una tecla para continuar . . . |
```

Explicación

Este programa simula cómo una CPU accede a datos desde la memoria RAM usando una caché pequeña.

- Se crea una caché de 4 bloques (como una "cajita" rápida donde la CPU guarda datos usados recientemente).
- La CPU intenta leer 10 direcciones de memoria (como 0, 4, 8, 12, 0, 4, 16, 20, 0, 4).
- En cada acceso:
 - Si el dato está en caché (HIT): La CPU lo lee al instante.
 - Si no está (MISS): La CPU lo trae de la RAM (más lento) y lo guarda en caché para después.

1. Inicialización:

La caché empieza vacía (todos los bloques valen -1).

2. Acceso a memoria:

La CPU quiere leer la dirección 0:

- MISS (la caché está vacía).
- Guarda el dato en la caché.

Luego lee 4:

- MISS (nuevo dato).
- Lo guarda.

Cuando repite 0:

- HIT (ya estaba en caché).

3. Resultados:

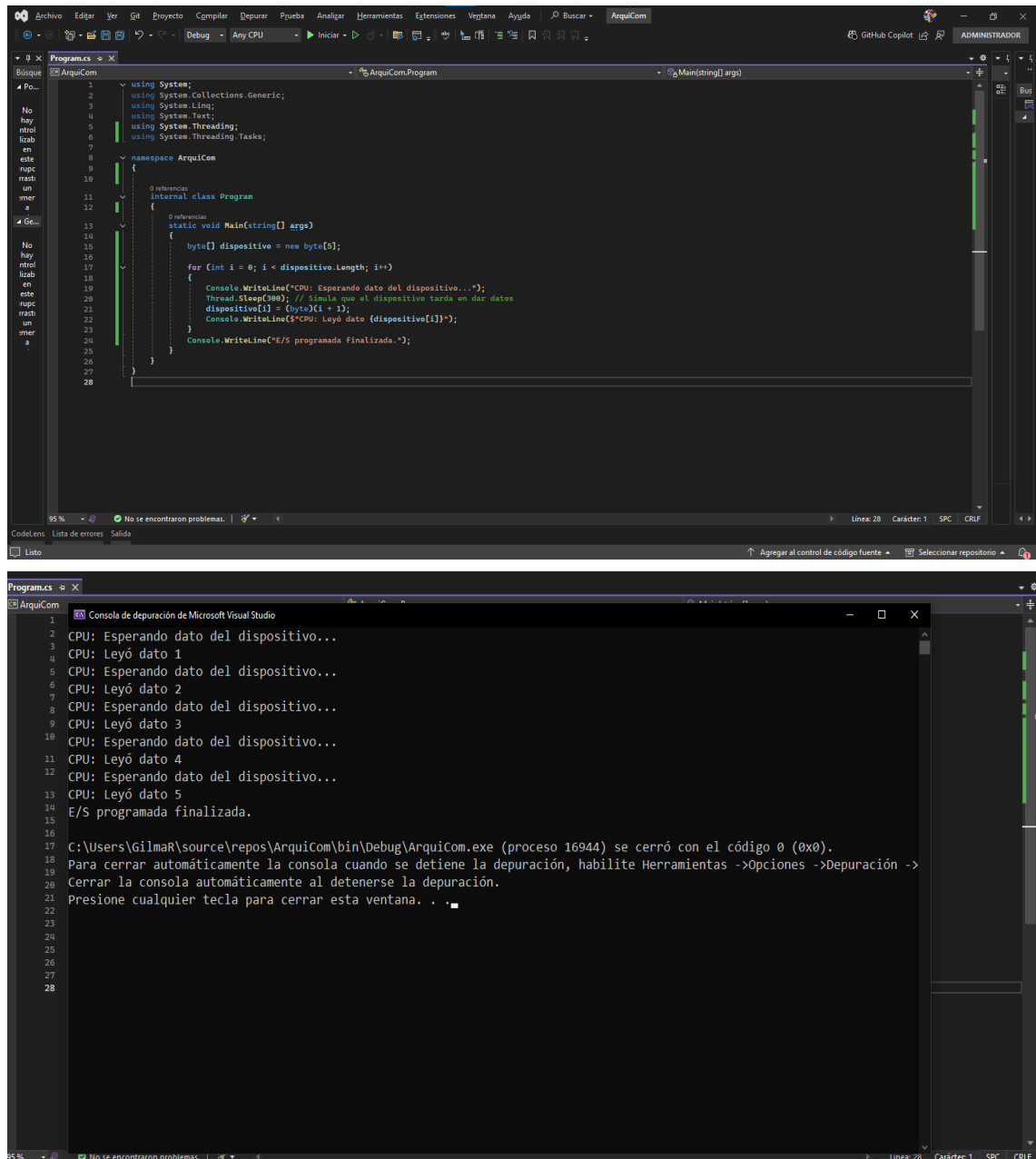
Muestra cuántos accesos fueron rápidos (HIT) y cuántos lentos (MISS).

Operación de Entrada/Salida:

Implementar ejemplos de E/S programada y E/S mediante interrupciones usando simuladores. Analizar el flujo de datos mediante DMA con un ejemplo práctico.

1. Ejemplo de E/S Programada (Polling)

En este método, la CPU consulta repetidamente el estado del dispositivo hasta que haya datos listos. Esto consume tiempo de CPU.



The image displays two screenshots from the Visual Studio IDE. The top screenshot shows the source code for a C# program named 'Arquicom'. The code is as follows:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading;
6 using System.Threading.Tasks;
7
8 namespace Arquicom
9 {
10     0 referencias
11     internal class Program
12     {
13         0 referencias
14         static void Main(string[] args)
15         {
16             byte[] dispositivo = new byte[5];
17
18             for (int i = 0; i < dispositivo.Length; i++)
19             {
20                 Console.WriteLine("CPU: Esperando dato del dispositivo...");
21                 Thread.Sleep(300); // Simula que el dispositivo tarda en dar datos
22                 dispositivo[i] = (byte)(i + 1);
23                 Console.WriteLine($"CPU: Leyó dato {dispositivo[i]}");
24             }
25             Console.WriteLine("E/S programada finalizada.");
26         }
27     }
28 }
```

The bottom screenshot shows the 'Console de depuración de Microsoft Visual Studio' window. It displays the output of the program during a debug session:

```
1 CPU: Esperando dato del dispositivo...
2 CPU: Leyó dato 1
3 CPU: Esperando dato del dispositivo...
4 CPU: Leyó dato 2
5 CPU: Esperando dato del dispositivo...
6 CPU: Leyó dato 3
7 CPU: Esperando dato del dispositivo...
8 CPU: Leyó dato 4
9 CPU: Esperando dato del dispositivo...
10 CPU: Leyó dato 5
11 E/S programada finalizada.
12
13 C:\Users\GilmaR\source\repos\Arquicom\bin\Debug\Arquicom.exe (proceso 16944) se cerró con el código 0 (0x0).
14 Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas -> Opciones -> Depuración ->
15 Cerrar la consola automáticamente al detenerse la depuración.
16 Presione cualquier tecla para cerrar esta ventana. . .
```

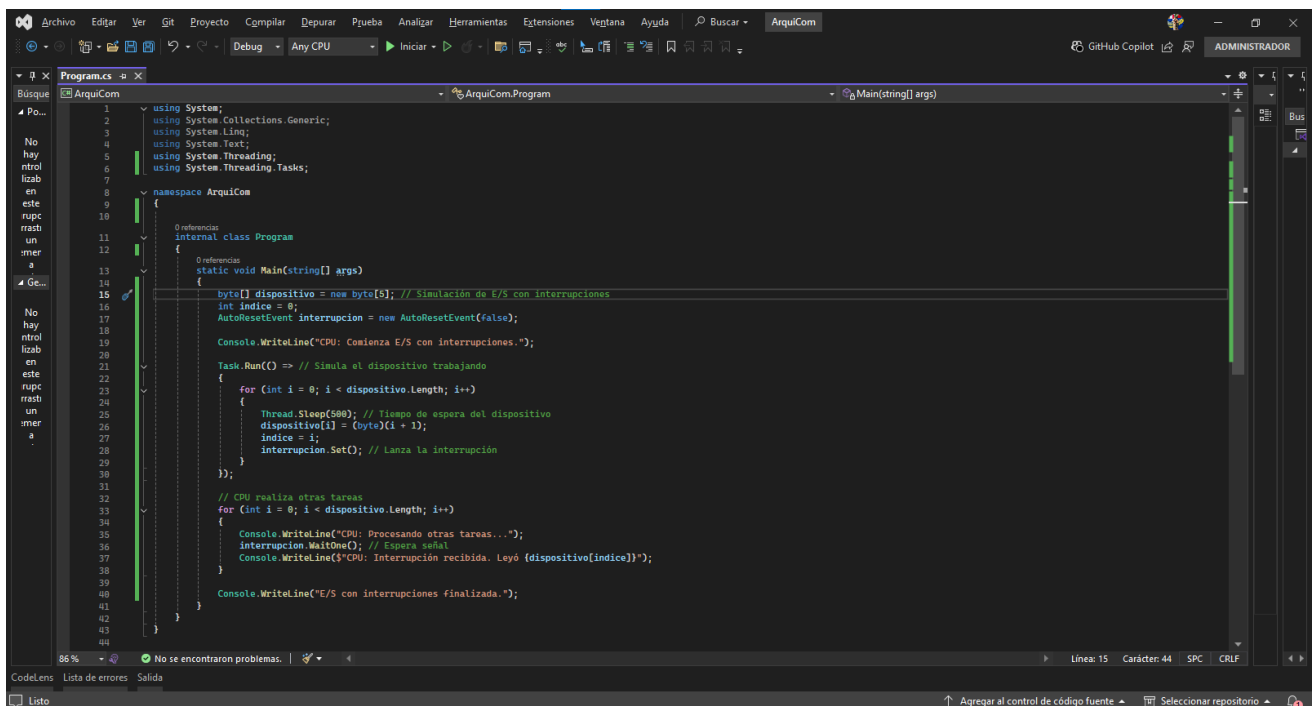
Explicación:

Este programa simula cómo la CPU lee datos de un dispositivo usando **E/S programada** (polling). Se crea un arreglo de 5 bytes que representa los datos que la CPU debe recibir. En un

ciclo, la CPU espera activamente (con una pausa de 300 ms para simular la demora del dispositivo), luego lee un dato (que se asigna como $i + 1$) y lo muestra en pantalla. Este proceso se repite hasta leer los 5 datos. Al final, se indica que la operación de E/S programada terminó. En resumen: el programa muestra cómo la CPU espera y obtiene datos secuencialmente de un dispositivo sin hacer otra cosa mientras espera.

2. Ejemplo de E/S mediante Interrupciones

Aquí la CPU puede realizar otras tareas mientras espera que el dispositivo “avise” que tiene datos listos.



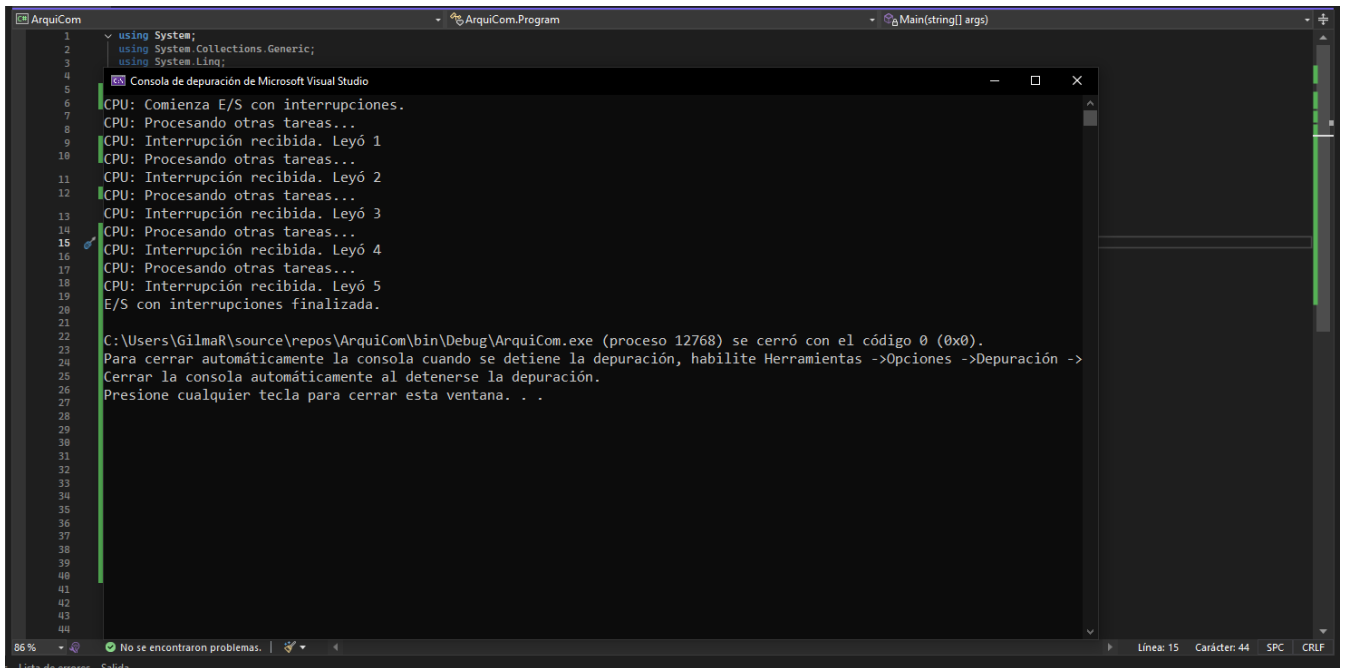
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading;
6 using System.Threading.Tasks;
7
8 namespace Arquicom
9 {
10     internal class Program
11     {
12         static void Main(string[] args)
13         {
14             byte[] dispositivo = new byte[5]; // Simulación de E/S con interrupciones
15             int indice = 0;
16             AutoResetEvent interrupcion = new AutoResetEvent(false);
17             Console.WriteLine("CPU: Comienza E/S con interrupciones.");
18
19             Task.Run(() => // Simula el dispositivo trabajando
20             {
21                 for (int i = 0; i < dispositivo.Length; i++)
22                 {
23                     Thread.Sleep(500); // Tiempo de espera del dispositivo
24                     dispositivo[i] = (byte)(i + 1);
25                     indice = i;
26                     interrupcion.Set(); // Lanza la interrupción
27                 }
28             });
29
30             // CPU realiza otras tareas
31             for (int i = 0; i < dispositivo.Length; i++)
32             {
33                 Console.WriteLine("CPU: Procesando otras tareas...");
34                 interrupcion.WaitOne(); // Espera señal
35                 Console.WriteLine($"CPU: Interrupción recibida. Leyó {dispositivo[indice]}");
36             }
37
38             Console.WriteLine("E/S con interrupciones finalizada.");
39         }
40     }
41 }
```

Explicación:

Este programa simula la comunicación entre un dispositivo y la CPU usando interrupciones en vez de E/S programada. Aquí la CPU no está constantemente esperando, sino que realiza otras tareas hasta que el dispositivo le avisa que hay datos listos (interrupción).

1. Se crea un arreglo dispositivo de 5 bytes para almacenar datos que el dispositivo "envía".
2. Se usa un objeto AutoResetEvent llamado interrupcion, que sirve para que la CPU espere una señal
3. Se inicia una tarea en paralelo (Task.Run) que simula el trabajo del dispositivo:
 - Cada 500 ms, el dispositivo "genera" un dato (valores 1, 2, 3, 4 y 5).
 - Cuando el dato está listo, actualiza la posición actual índice y llama a interrupcion.Set() para notificar a la CPU (simula la interrupción).
4. Mientras tanto, en el hilo principal (CPU), se ejecuta un ciclo donde:

- La CPU dice que está haciendo otras tareas (simulación).
 - Luego llama a `interrupcion.WaitOne()`, que hace que espere hasta recibir la señal de interrupción.
 - Cuando recibe la señal, lee el dato actual del arreglo y lo muestra.
5. Al final, se imprime que la operación de E/S con interrupciones ha terminado.



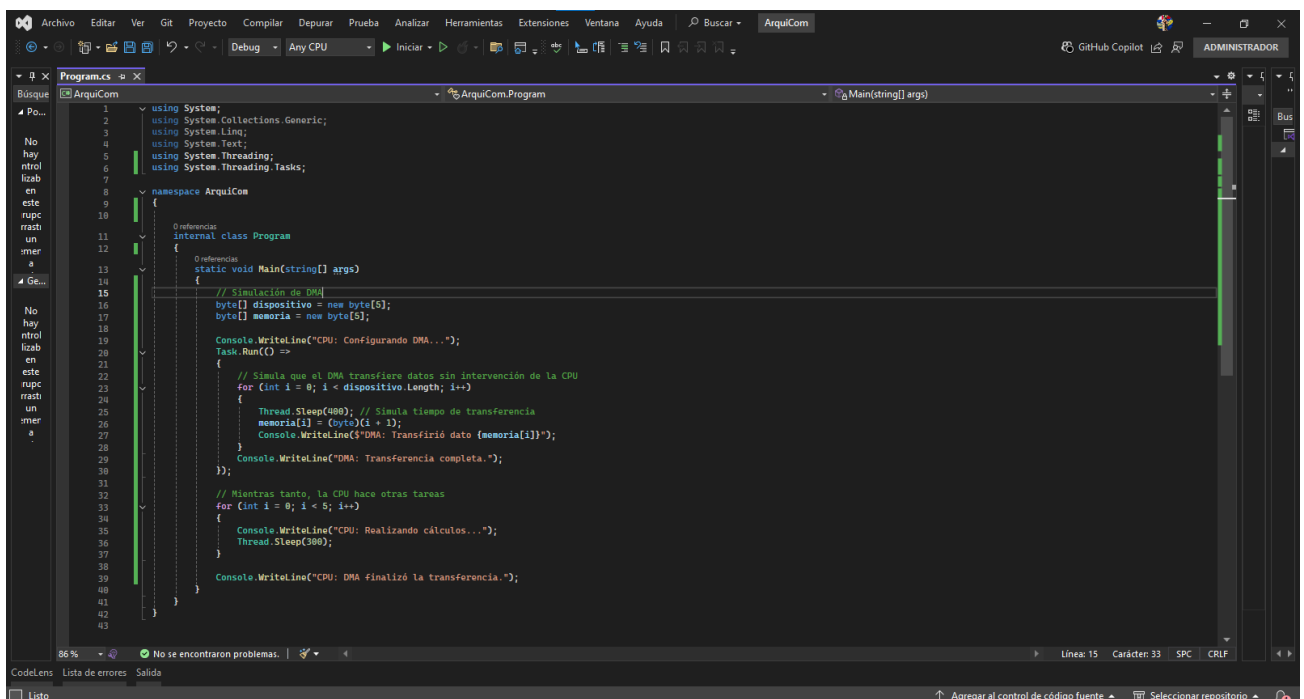
```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  Console de depuración de Microsoft Visual Studio
6  CPU: Comienza E/S con interrupciones.
7  CPU: Procesando otras tareas...
8  CPU: Interrupción recibida. Leyó 1
9  CPU: Procesando otras tareas...
10 CPU: Interrupción recibida. Leyó 2
11 CPU: Procesando otras tareas...
12 CPU: Interrupción recibida. Leyó 3
13 CPU: Procesando otras tareas...
14 CPU: Interrupción recibida. Leyó 4
15 CPU: Procesando otras tareas...
16 CPU: Interrupción recibida. Leyó 5
17 CPU: Procesando otras tareas...
18 CPU: Interrupción recibida. Leyó 5
19 E/S con interrupciones finalizada.
20
21 C:\Users\Gilmar\source\repos\ArquiCom\bin\Debug\ArquiCom.exe (proceso 12768) se cerró con el código 0 (0x0).
22 Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas -> Opciones -> Depuración ->
23 Cerrar la consola automáticamente al detenerse la depuración.
24 Presione cualquier tecla para cerrar esta ventana. . .
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

```

3. Ejemplo de DMA (Acceso Directo a Memoria)

En DMA, la CPU solo inicia la transferencia y un controlador especializado mueve los datos directamente a la memoria.



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading;
6  using System.Threading.Tasks;
7
8  namespace ArquiCom
9  {
10     0 referencias
11     internal class Program
12     {
13         0 referencias
14         static void Main(string[] args)
15         {
16             // Simulación de DMA
17             byte[] dispositivo = new byte[5];
18             byte[] memoria = new byte[5];
19
20             Console.WriteLine("CPU: Configurando DMA...");
21             Task.Run(() =>
22             {
23                 // Simula que el DMA transfiere datos sin intervención de la CPU
24                 for (int i = 0; i < dispositivo.Length; i++)
25                 {
26                     Thread.Sleep(400); // Simula tiempo de transferencia
27                     memoria[i] = (byte)(i + 1);
28                     Console.WriteLine($"DMA: Transfirió dato {memoria[i]}");
29                 }
30                 Console.WriteLine("DMA: Transferencia completa.");
31             });
32
33             // Mientras tanto, la CPU hace otras tareas
34             for (int i = 0; i < 5; i++)
35             {
36                 Console.WriteLine("CPU: Realizando cálculos...");
37                 Thread.Sleep(300);
38             }
39
40             Console.WriteLine("CPU: DMA finalizó la transferencia.");
41         }
42     }
43 }

```

```
Program.cs x
ArquiCom
using System;
using System.Collections.Generic;
using Console de depuración de Microsoft Visual Studio
CPU: Configurando DMA...
CPU: Realizando cálculos...
CPU: Realizando cálculos...
CPU: Realizando cálculos...
DMA: Transfirió dato 1
CPU: Realizando cálculos...
DMA: Transfirió dato 2
CPU: Realizando cálculos...
CPU: Realizando cálculos...
DMA: Transfirió dato 3
CPU: DMA finalizó la transferencia.

C:\Users\GilmaR\source\repos\ArquiCom\bin\Debug\ArquiCom.exe (proceso 6536) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Explicación:

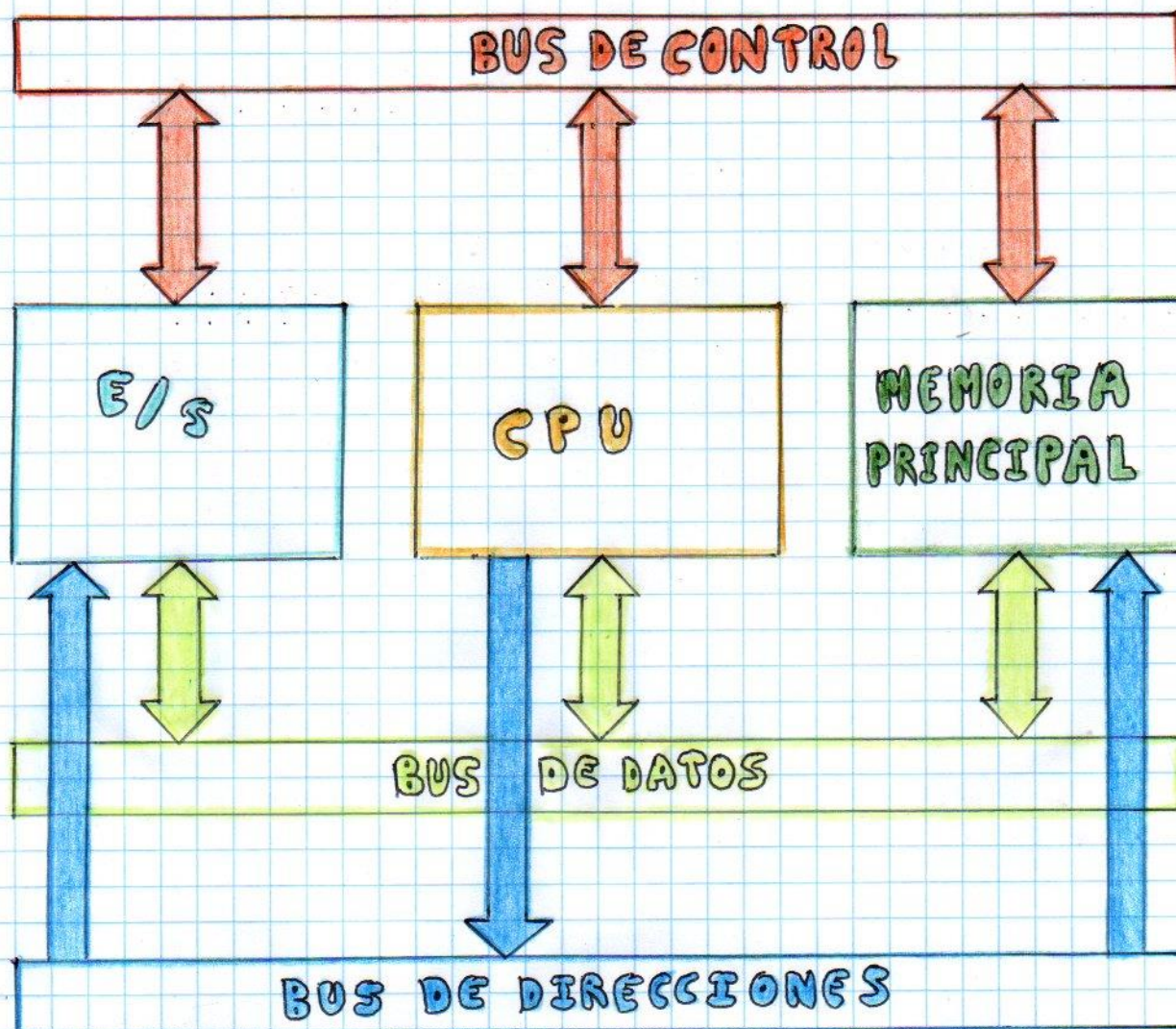
Este programa simula cómo un controlador DMA transfiere datos directamente desde un dispositivo a la memoria sin intervención activa de la CPU. Mientras el DMA transfiere, la CPU puede hacer otras tareas sin bloquearse.

- 1) Se crean dos arreglos de bytes:
 - dispositivo (simula los datos que vienen del dispositivo),
 - memoria (simula la memoria principal donde se copiarán los datos).
- 2) La CPU "configura" el DMA (simulado con un mensaje).
- 3) Se lanza una tarea en paralelo (Task.Run) que representa al controlador DMA:
 - Recorre los 5 datos,
 - Simula un tiempo de transferencia de 400 ms por dato,
 - Copia cada dato al arreglo memoria (aquí simplemente asigna valores de 1 a 5),
 - Muestra en consola qué dato transfirió.
- 4) Mientras el DMA está transfiriendo, en el hilo principal la CPU realiza otras tareas, simuladas con un ciclo que imprime "Realizando cálculos..." y espera 300 ms en cada iteración.
- 5) Al terminar la tarea DMA y las tareas de la CPU, se muestra un mensaje que la transferencia ha finalizado.

PROCEDIMIENTOS

Diseño de Buses:

- Realizar diagramas que representen la estructura y función de un bus.



Este diagrama muestra la organización de un sistema computacional basado en tres tipos de buses: de datos, de direcciones y de control.

Bus de direcciones: transporta las direcciones de memoria o de entrada/salida que la CPU necesita leer o escribir. Va desde la CPU hacia la memoria y los módulos E/S.

Bus de datos: Permite la transferencia de información entre la CPU, la memoria principal y los dispositivos de E/S. El flujo puede ser bidireccional.

Bus de control: Transmite señales de control y sincronización, como lectura, escritura, interrupciones y reloj, coordinando el funcionamiento entre los componentes.

- Identificar y describir los diferentes tipos de buses y su impacto en la velocidad de transferencia.

- Bus de datos

Es un canal de comunicación bidireccional que transporta la información real (datos) entre la CPU, la memoria y los periféricos. Su ancho (32 o 64 bits) determina cuántos bits se transfieren por ciclo.

Impacto en la velocidad de transferencia.

Un bus de datos más ancho permite mover más información por ciclo, aumentando el ancho de banda. En los buses paralelos antiguos, todos los dispositivos funcionaban a la misma velocidad, lo que podía ser un cuello de botella. La introducción de controladores y la transición a buses seriales de alta velocidad mejoró considerablemente el rendimiento.

- Bus de direcciones

Es un canal unidireccional utilizado para indicar la posición específica de la memoria o dispositivo al que se accederá. El número de líneas (bits) que lo componen determina el espacio máximo direccionable.

Impacto en la velocidad de transferencia.

Un bus de direcciones más amplio permite acceder directamente a una mayor cantidad de memoria sin necesidad de técnicas de paginación o multiplexado, reduciendo la latencia en accesos. Sin embargo, no influye directamente en la tasa de transferencia, sino en la capacidad y rapidez de acceso a datos específicos.

- Bus de control

Un bus de control es un conjunto de señales bidireccionales que coordinan las operaciones entre CPU, memoria y periféricos, gestiona comandos, respuestas e interrupciones.

Impacto en la velocidad de transferencia.

Su eficiencia influye en la sincronización del sistema. Una coordinación más precisa reduce tiempos de espera y colisiones en el acceso, optimizando el rendimiento global incluso si el bus de datos y de direcciones son rápidos.

CONCLUSIÓN

A través de esta investigación bibliográfica, se logró comprender la función e importancia de los principales componentes de un computador: la memoria con su jerarquía y el papel clave de la caché, los dispositivos de entrada/salida y sus métodos de comunicación con la CPU y los buses como eje de interconexión entre componentes. Los ejercicios prácticos, como las simulaciones de caché y E/S, demostraron cómo estos elementos trabajan en conjunto para optimizar el rendimiento del sistema.

BIBLIOGRAFÍA

- [1] B. Jacob, *The Memory System*. Cham: Springer International Publishing, 2009. doi: 10.1007/978-3-031-01724-7.
- [2] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, and J. S. Vetter, “A survey on processing-in-memory techniques: Advances and challenges,” *Memories - Materials, Devices, Circuits and Systems*, vol. 4, p. 100022, Jul. 2023, doi: 10.1016/j.memori.2022.100022.
- [3] A. M. Mohamed, N. Mubark, and S. Zaghloul, “Performance aware shared memory hierarchy model for multicore processors,” *Sci Rep*, vol. 13, no. 1, p. 7313, May 2023, doi: 10.1038/s41598-023-34297-3.
- [4] K. Shakiba and M. Stumm, “PaperCache: In-Memory Caching with Dynamic Eviction Policies,” in *Proceedings of the 17th ACM Workshop on Hot Topics in Storage and File Systems*, New York, NY, USA: ACM, Jul. 2025, pp. 107–113. doi: 10.1145/3736548.3737836.
- [5] S. A. Shirke, N. Jayakumar, and S. Patil, “Design and performance analysis of modern computational storage devices: A systematic review,” *Expert Syst Appl*, vol. 250, p. 123570, Sep. 2024, doi: 10.1016/j.eswa.2024.123570.
- [6] “Near-ideal in-memory sensing and computing devices using ferroelectrics,” *Nat Mater*, vol. 22, no. 12, pp. 1447–1448, Dec. 2023, doi: 10.1038/s41563-023-01692-0.
- [7] D. Fakhry, M. Abdelsalam, M. W. El-Kharashi, and M. Safar, “A review on computational storage devices and near memory computing for high performance applications,” *Memories - Materials, Devices, Circuits and Systems*, vol. 4, p. 100051, Jul. 2023, doi: 10.1016/j.memori.2023.100051.

- [8] “MEMSYS: Memory Systems,” in *The International Symposium on Memory Systems*, New York, NY, USA: ACM, 2021.
- [9] “Structured Input-Output Tools for Modal Analysis of a Transitional Channel Flow,” Jan. 23, 2023. doi: 10.2514/6.2023-1805.vid.
- [10] R. Iyer *et al.*, “Advances in Microprocessor Cache Architectures Over the Last 25 Years,” *IEEE Micro*, vol. 41, no. 6, pp. 78–88, Nov. 2021, doi: 10.1109/MM.2021.3114903.
- [11] W. F. Godoy *et al.*, “ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management,” *SoftwareX*, vol. 12, p. 100561, Jul. 2020, doi: 10.1016/j.softx.2020.100561.
- [12] L. Robaldo, C. Bartolini, M. Palmirani, A. Rossi, M. Martoni, and G. Lenzini, “Formalizing GDPR Provisions in Reified I/O Logic: The DAPRECO Knowledge Base,” *J Logic Lang Inf*, vol. 29, no. 4, pp. 401–449, Dec. 2020, doi: 10.1007/s10849-019-09309-z.
- [13] D. Comer, *Essentials of Computer Architecture*. Boca Raton: Chapman and Hall/CRC, 2024. doi: 10.1201/9781003410140.
- [14] D. P. López Carrillo, Á. M. Acuña Félix, R. F. Tipan Tisalema, G. I. Vanegas Zabala, and D. F. Yumisa León, *Arquitectura de computadoras*. Centro de Investigación y Desarrollo Ecuador, 2025. doi: 10.33996/cide.ecuador.AC2679376.
- [15] C. V. Phan and T. D. Nguyen, *Context-Aware Systems and Applications*, vol. 475. Cham: Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-28816-6.
- [16] S. J. Rogowski, “Bus,” in *Encyclopedia of Computer Science*, GBR: John Wiley and Sons Ltd., 2003, pp. 165–167. doi: 10.5555/1074100.1074184.
- [17] T. E. Carlson, “Bus and Memory Architectures,” in *Handbook of Computer Architecture*, Singapore: Springer Nature Singapore, 2024, pp. 201–212. doi: 10.1007/978-981-97-9314-3_68.
- [18] S. P. Wang, *Computer Architecture and Organization*, 1st ed. Singapore: Springer Singapore, 2021. doi: 10.1007/978-981-16-5662-0.

- [19] B. Tiwari, M. Yang, X. Wang, and Y. Jiang, “Data Streaming and Traffic Gathering in Mesh-based NoC for Deep Neural Network Acceleration,” *CoRR*, vol. abs/2108.02569, 2021, doi: 10.48550/arXiv.2108.02569.

ANEXO

Link del GitHub: <https://github.com/AndyMendoza0308/ARQCOMP-GRUPOF---COMPONENTES-DEL-COMPUTADOR.git>