Fundamento Teórico

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 [1].

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 [2].

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 [3].

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 [3].

Tipos de buses

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 [4].

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 [4].

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 [4].

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 [5].

Procedimientos

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 $t_{hit} = 1\,ns$ y tasa de aciertos hit_rate = 95%. La memoria principal tarda $t_{mem} = 50\,ns$. Supón que la penalidad por fallo es igual al tiempo de la memoria principal. Calcula el EAT.

$$miss\_rate = 1 - 0,95 = 0,05$$

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

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

$$EAT = 0,95\,ns + 2,5\,ns$$

$$EAT = 3,45\,ns$$

R.// 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 1ns, tasa de aciertos del 91%, una Caché L2 con un tiempo de acceso de 8ns 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.

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

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

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

$$AMAT = 0,91 + (0,09)(7,76 + 2,10)$$

$$AMAT = 0,91 + (0,09)(9,86)$$

$$AMAT = 0,91 + 0,8874$$

$$AMAT = 1,7974\,ns$$

R.// 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 Kb x 1024 = 65536 bytes

Número de líneas

$\frac{65536}{128} = 512$ líneas

Bits de offset

$\log_2(128) = 7$ bits

Bits de índice

$\log_2(512) = 9$ bits

Bits de etiqueta

$32 - (9+7) = 32 - 16 = 16$ 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 expresalo en MiB y en GiB. (usando potencias de 2: 1 KiB = 1024 bytes).

1 MiB = 1024 KB

$L_1 = 16\ KB = \frac{16}{1024\ MiB} = 0,015625\ MiB$

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

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

Memoria = 8 GB = 8 x 1024 MiB = 8192 MiB

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

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

- 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 hitrate 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 mejoja del hit_rate al 96% el tiempo se reduce a 3ns.

Simulación en C# de comportamiento de una memoria caché

Código

Program.cs*

SimulacionCache_GrupoF | Program | Main()

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Caché
        int[] cache = new int[4]; // 4 bloques de caché
        for (int i = 0; i < cache.Length; i++)
        {
            cache[i] = -1;
        }

        int hits = 0;
        int misses = 0;

        // Simulamos accesos a memoria
        int[] accesos = { 0, 4, 8, 12, 0, 4, 16, 20, 0, 4 };

        foreach (int direccion in accesos)
        {
            int bloque = direccion / 4; // Tamaño de bloque = 4 bytes
            int indice = bloque % cache.Length;

            Console.Write($"Acceso a dirección {direccion} (bloque {bloque})... ");

            if (cache[indice] == bloque)
            {
                hits++;
                Console.WriteLine("HIT!");
            }
            else
            {
                misses++;
                cache[indice] = bloque;
                Console.WriteLine("MISS - bloque cargado en caché");
            }
        }

        // Resultados
        Console.WriteLine("\nResumen:");
        Console.WriteLine($"Total accesos: {accesos.Length}");
        Console.WriteLine($"Hits: {hits}");
        Console.WriteLine($"Misses: {misses}");
        Console.WriteLine($"Tasa de hits: {(float)hits / accesos.Length:P0}");
    }
}
```

Ejecución

C:\Windows\system32\cmd.e:

```
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).

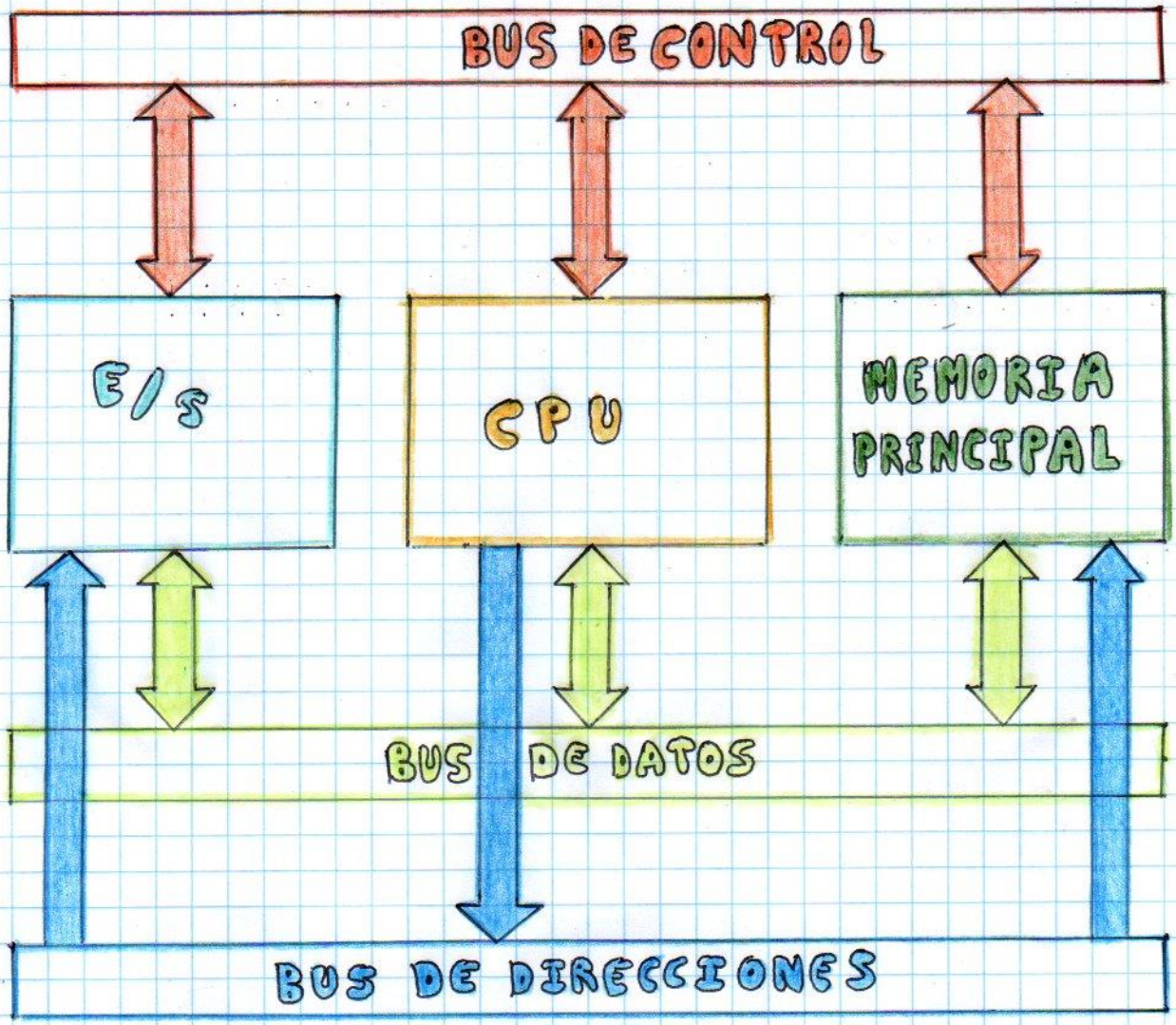
3. Resultados:

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

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.

Bibliografía

[1] 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.

[2] S. J. Rogowski, "Bus," in *Encyclopedia of Computer Science*, GBR: John Wiley and Sons Ltd., 2003, pp. 165–167. doi: 10.5555/1074100.1074184.

[3] 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.

[4] S. P. Wang, *Computer Architecture and Organization*, 1st ed. Singapore: Springer Singapore, 2021. doi: 10.1007/978-981-16-5662-0.

[5] 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.