Optimización de Multiplezores y Líneas de Control en una Implementación Básica de MIPS

Juan Daniel Ardila[1], Juan Pablo Avila[2], Kevin Castro[3], Jhon Almanzar[4]

Escuela de Ingeniería de Sistemas e Informática, Universidad Industrial de Santander, Bucaramanga

[1]Juan2211910@correo.uis.edu.co, [2]castrokevin312@gmail.com, [3]avilaquitianjuanpablo@gmail.com, [4]jhonalmanzarq@gmail.com

Resumen. *La arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages) es fundamental en el diseño de microprocesadores debido a su simplicidad y eficiencia. Este artículo se enfoca en la implementación detallada de multiplexores y líneas de control dentro de un procesador MIPS básico, resaltando su importancia en el rendimiento y la funcionalidad del sistema. Los principales hallazgos demuestran que una correcta configuración de estos componentes es crucial para optimizar el procesamiento de instrucciones y minimizar errores. Además, se presentan las mejores prácticas y los desafíos comunes en la implementación de MIPS, proporcionando una guía esencial para ingenieros y diseñadores de hardware.*

Abstract. *The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is fundamental in microprocessor design due to its simplicity and efficiency. This article focuses on the detailed implementation of multiplexers and control lines within a basic MIPS processor, highlighting their importance in system performance and functionality. The main findings show that proper configuration of these components is crucial for optimizing instruction processing and minimizing errors. Furthermore, best practices and common challenges in MIPS implementation are presented, providing essential guidance for engineers and hardware designers.*

I. INTRODUCCIÓN

La arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages) es una de las arquitecturas de conjunto de instrucciones (ISA) más influyentes y ampliamente estudiadas en la industria de los microprocesadores. Desarrollada en la década de 1980 por MIPS Computer Systems, esta arquitectura se caracteriza por su simplicidad, eficiencia y facilidad de implementación, lo que la ha convertido en una elección popular tanto en entornos académicos como en aplicaciones comerciales, desde sistemas embebidos hasta supercomputadoras [1].

La relevancia de la arquitectura MIPS en la industria de los microprocesadores radica en su diseño RISC (Reduced Instruction Set Computing), que permite un procesamiento más rápido y eficiente al utilizar un conjunto reducido de instrucciones simples y optimizadas. Este enfoque no solo facilita la implementación de los procesadores, sino que también mejora su rendimiento y reduce el consumo de energía, aspectos críticos en el diseño de sistemas modernos.

El propósito de este artículo es analizar en detalle la implementación de componentes clave en una implementación básica de la arquitectura MIPS, específicamente los multiplexores y las líneas de control. Estos elementos son fundamentales para el funcionamiento correcto y eficiente del procesador, ya que gestionan la selección de datos y la coordinación de las operaciones internas del mismo. A través de este análisis, se busca proporcionar una comprensión profunda de cómo estos componentes contribuyen a la operación general de la arquitectura MIPS y cómo pueden ser optimizados para mejorar el desempeño del sistema.

II. METODOLOGÍA

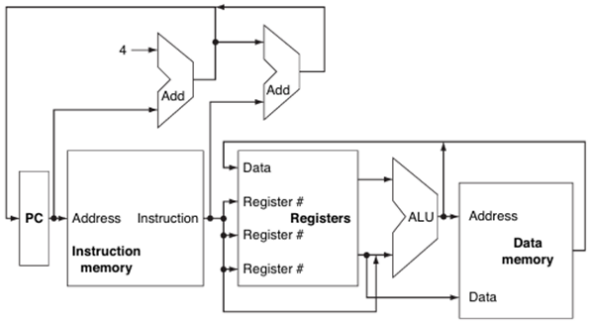
El diseño del datapath es fundamental en la arquitectura de computadoras para implementar eficientemente un conjunto de instrucciones específico, como el conjunto de instrucciones MIPS (Microprocessor without Interlocked Pipeline Stages). Cada instrucción MIPS requiere una combinación específica de componentes del datapath, que incluyen la unidad de memoria de instrucciones, el contador de programa (PC), el sumador (ALU)... Estos elementos trabajan en conjunto para permitir la lectura de instrucciones desde la memoria, la ejecución de operaciones aritméticas y lógicas, así como la transferencia de datos entre la memoria y los registros del procesador.

En este documento vamos a explorar cómo se integran estos componentes en el diseño del datapath para ejecutar instrucciones de distintos formatos en una arquitectura MIPS.

III. ELEMENTOS DEL DISEÑO DE CAMINO DE DATOS

Los elementos del camino de datos se dividen en dos categorías principales:

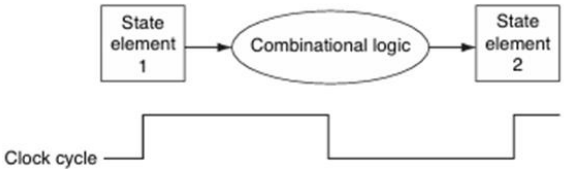
Los elementos que operan sobre valores de datos: Estos generan salidas exclusivamente a partir de las entradas actuales. Por ejemplo, la unidad (ALU) representada en la siguiente figura [2].



Los elementos que contienen estado: Son cruciales para el almacenamiento de información y la gestión de secuencias de datos en el computador. Estos elementos, como las memorias y los registros, retienen su estado incluso después de que se interrumpe la energía..

IV. METODOLOGÍA DE SINCRONIZACIÓN

En la arquitectura de MIPS, se emplea una metodología de sincronización basada en flancos de reloj, donde las transiciones de estado en los elementos del sistema solo ocurren en el momento exacto de un flanco de subida o bajada del reloj, con este enfoque garantizamos que las operaciones sean lo más consistente posibles, evitando problemas de interferencia y teniendo seguridad en los datos.



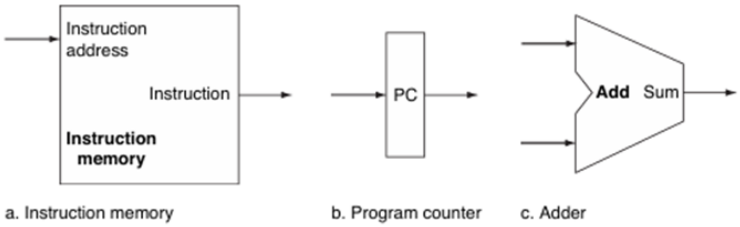
Acabamos de explorar un aspecto simple pero muy importante: entender la estructura y el funcionamiento, junto con la metodología de sincronización. Estos temas son los principios, bajos los cuales se construyen y optimizan los procesadores. A continuación, profundizaremos en el uso de elementos esenciales, componentes, instrucciones…

V. ALGUNOS ELEMENTOS ESCENCIALES

a. **Unidad de Memoria de Instrucciones**: La unidad de memoria de instrucciones almacena las instrucciones del programa y proporciona instrucciones basadas en una dirección.

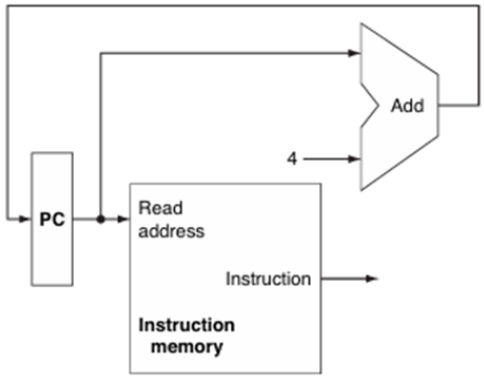
b. **Contador de Programa (PC):** El contador de programa mantiene la dirección de la instrucción actual en ejecución. Después de cada ciclo de reloj, se incrementa el valor del PC para apuntar a la siguiente instrucción en secuencia.

c. **Sumador (ALU):** Utilizaremos un sumador para incrementar el valor del PC y calcular la dirección de la siguiente instrucción. Este sumador, que es un componente combinacional, simplemente suma el valor actual del PC con un desplazamiento constante (4 en el caso de MIPS) para obtener la dirección siguiente.



VI. Diseño del Datapath para la Actualización del PC:

El siguiente diagrama permita la recuperación de instrucciones desde la memoria y la actualización del PC para apuntar a la siguiente instrucción:



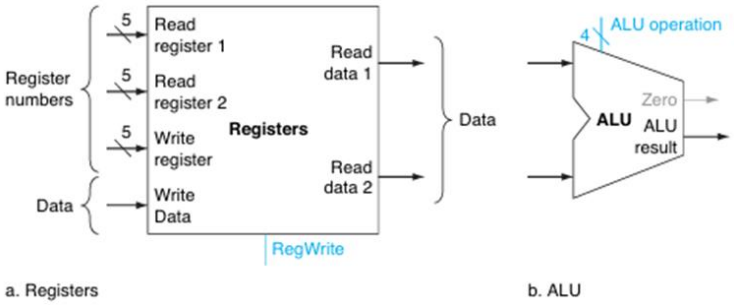
Su funcionamiento es el siguiente: El PC proporciona la dirección a la unidad de memoria de instrucciones de ahí la unidad de memoria de instrucciones lee la instrucción ubicada en la dirección proporcionada por el PC y luego con el uso del Add hace un incremento de 4 bytes lo que permite al PC apuntar a la siguiente dirección de la instrucción.

VII. Instrucciones de Formato R (Arithmetic-Logical)

Las instrucciones de formato R incluyen operaciones aritméticas y lógicas que operan en registros como lo son add, sub, and, or… Para implementar estas instrucciones, necesitamos:

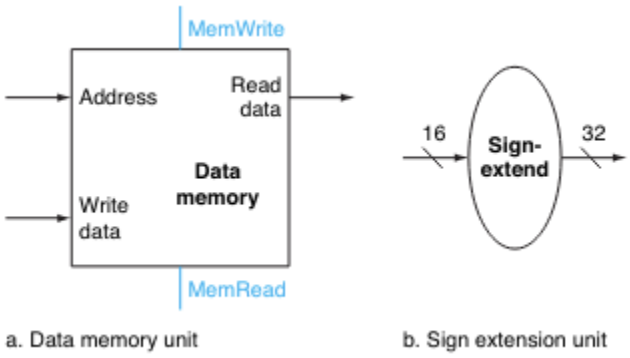
a. **Archivo de Registros**: Un conjunto de registros donde cada registro puede ser leído o escrito especificando su número. También se puede leer datos desde la memoria (lw) o escribir datos en la memoria (sw).

b. **ALU:** Una ALU que realiza operaciones aritméticas y lógicas en los valores leídos desde los registros y produce resultados que se pueden escribir de nuevo en los registros. También se puede calcular la dirección de memoria basada en el registro base y el offset especificado en la instrucción.



En este diagrama se puede observar parte de su funcionamiento: Se especifican dos registros de entrada para la ALU desde el archivo de registros, la ALU realiza la operación especificada por la instrucción (por ejemplo, suma, resta, AND, OR...) y este resultado se escribe de vuelta en el registro especificado en la instrucción.

Para implementar la instrucción ***beq*** (branch if equal) en un procesador, se necesitan dos unidades adicionales: la memoria de datos y la unidad de extensión de signo, como se muestra en la Figura 4.8.



VIII. MEMORIA DE DATOS

FUNCIONES: LECTURA Y ESCRITURA DE DATOS.

ENTRADAS Y SALIDAS:

- Dirección
- Datos a escribir
- Control de lectura (MemRead)
- Control de escritura (MemWrite)
- Datos leídos

Extensión de Signo

- Función: Convierte un valor de 16 bits a un valor de 32 bits con signo.
- Entradas y salidas:
- Entrada: 16 bits
- Salida: 32 bits

Implementación de beq

1. Operandos: Dos registros a comparar y un desplazamiento de 16 bits.
2. Cálculo de la dirección de destino:

    - Se suma el campo de desplazamiento, extendido a 32 bits, al valor del PC.
    - El valor base para este cálculo es la dirección de la instrucción siguiente (PC + 4).

Estas unidades y detalles son esenciales para realizar operaciones de carga, almacenamiento y ramas en el procesador.

IX. Creación de un Single Datapath

Para combinar los componentes del Datapath necesarios para las diferentes clases de instrucciones en un solo Datapath y agregar el control necesario, se deben considerar varios factores. El Datapath más simple intentará ejecutar todas las instrucciones en un solo ciclo de reloj. Esto significa que ningún recurso del Datapath puede usarse más de una vez por instrucción, por lo que cualquier elemento necesario más de una vez debe duplicarse. Por lo tanto, se necesita una memoria separada para instrucciones y otra para datos. Aunque algunas unidades funcionales deberán duplicarse, muchos elementos pueden ser compartidos por diferentes flujos de instrucciones.

Elementos del Datapath

1. Memoria de Instrucciones:
    - Almacena las instrucciones del programa.
    - Necesaria para la fase de búsqueda de instrucciones.
2. Memoria de Datos:
    - Utilizada para operaciones de carga y almacenamiento de datos.
    - Separa la memoria de instrucciones de la de datos para evitar conflictos.
3. Unidades Funcionales:
    - ALU: Realiza operaciones aritméticas y lógicas.
    - Sumador: Calcula direcciones de ramas y PC + 4.
    - Extensor de Signo: Convierte valores de 16 bits a 32 bits con signo.
4. Multiplexores:
    - Permiten seleccionar entre múltiples entradas para un único elemento de salida.
    - Utilizados para compartir elementos del Datapath entre diferentes flujos de instrucciones.

Compartición de Recursos

Para compartir un elemento del Datapath entre dos clases de instrucciones diferentes, se pueden usar multiplexores y señales de control para seleccionar entre múltiples entradas. Esto permite que el mismo elemento del Datapath se use en diferentes etapas de ejecución de instrucciones sin necesidad de duplicarlo.

Control

El control se encarga de generar las señales necesarias para seleccionar y activar los diferentes componentes del Datapath en cada ciclo de reloj. Esto incluye:

- Señales de control para multiplexores: Seleccionan las entradas adecuadas según la instrucción actual.
- Señales de lectura/escritura para memorias: Determinan cuándo se deben leer o escribir datos.
- Señales de operación para la ALU: Indican qué operación debe realizar la ALU (suma, resta, comparación, etc.).

Integración de Componentes

Al combinar todos estos componentes en un único Datapath, se deben seguir los siguientes pasos:

1. Conexión de Memorias: Asegurar que las memorias de instrucciones y datos estén correctamente conectadas y accesibles.
2. Interconexión de Unidades Funcionales: Conectar la ALU, sumadores y extensores de signo a través de multiplexores según sea necesario.
3. Implementación de Control: Diseñar un módulo de control que genere las señales adecuadas para cada ciclo de reloj basado en la instrucción actual.

Esquema del Datapath

1. Fase de Búsqueda de Instrucción: PC -> Memoria de Instrucciones -> Instrucción.
2. Fase de Decodificación y Lectura de Registros: Instrucción -> Archivo de Registros.
3. Fase de Ejecución:
    - ALU: Operaciones aritméticas/lógicas.
    - Sumador: Cálculo de direcciones de ramas.
    - Extensor de Signo: Extensión de desplazamientos.
4. Fase de Memoria:
    - Memoria de Datos: Operaciones de carga/almacenamiento.
5. Fase de Escritura de Resultados: Resultados de la ALU/Memoria -> Archivo de Registros.

Este enfoque asegura que el Datapath pueda manejar eficientemente todas las clases de instrucciones, ejecutándolas en un solo ciclo de reloj mediante la correcta distribución y control de recursos.

Esquema de Implementación Sencilla

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Esta implementación simple del subconjunto MIPS cubre las instrucciones de carga de palabra (lw), almacenamiento de palabra (sw), rama si igual (beq), y las instrucciones aritmético-lógicas (add, sub, AND, OR, y set on less than). Posteriormente, se mejorará el diseño para incluir la instrucción de salto (j).

Control de la ALU

La ALU de MIPS puede realizar seis combinaciones de entradas de control, como se muestra en la tabla proporcionada. Dependiendo de la clase de instrucción, la ALU necesitará realizar una de las siguientes funciones:

- Instrucciones de carga y almacenamiento (lw y sw): La ALU calcula la dirección de memoria sumando.
- Instrucciones R-type: La ALU realiza una de las cinco acciones (AND, OR, subtract, add, o set on less than) según el campo funct de 6 bits en la instrucción.
- Instrucción de rama (beq): La ALU realiza una resta para verificar la igualdad.

Generación de Señales de Control de la ALU

Para generar la entrada de control de 4 bits de la ALU, se utiliza una pequeña unidad de control que recibe como entradas el campo de función de la instrucción y un campo de control de 2 bits llamado ALUOp. ALUOp indica la operación a realizar:

- 00: Suma (para cargas y almacenes).
- 01: Resta (para beq).
- 10: Determinada por el campo funct.

El campo funct se utiliza para determinar la operación específica en las instrucciones R-type. La salida de la unidad de control de la ALU es una señal de 4 bits que controla directamente la ALU, generando una de las combinaciones mostradas anteriormente.

Integración en el Datapath

1. Memoria de Instrucciones y Datos: Se separan para evitar conflictos de acceso.
2. Archivo de Registros: Lee los registros necesarios para las operaciones.
3. Unidad de Extensión de Signo: Extiende desplazamientos de 16 bits a 32 bits.
4. ALU y Sumador: Realizan operaciones aritméticas y lógicas, y calculan direcciones de memoria.
5. Multiplexores: Permiten la selección de entradas adecuadas para cada operación.
6. Control: Genera las señales necesarias para coordinar todas las unidades del Datapath.

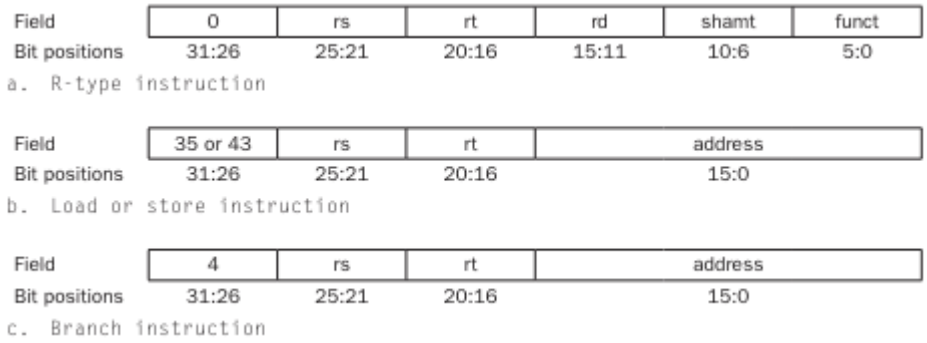
Implementación de Control para Instrucciones

- lw y sw:
  - Utilizan la ALU para calcular la dirección de memoria.
  - ALUOp = 00 (suma).
- Instrucciones R-type:
  - Utilizan el campo funct para determinar la operación específica.
  - ALUOp = 10.
- beq:
  - Utiliza la ALU para restar y verificar igualdad.
  - ALUOp = 01.

Este esquema asegura que el Datapath pueda manejar eficientemente las instrucciones del subconjunto MIPS, ejecutándolas en un solo ciclo de reloj mediante la correcta distribución y control de recursos.

X. Diseño de la Unidad de Control Principal

Para diseñar la unidad de control principal, primero identificamos los campos de una instrucción y las líneas de control necesarias para el Datapath construido. A continuación, revisamos los formatos de las tres clases de instrucciones: R-type, de carga y almacenamiento, y de rama.



a) Formatos de Instrucción (Figura 4.14)

1. **R-type (Formato R)**:
   - **Campos**:
     - opcode: 6 bits, siempre 0 para instrucciones R-type.
     - rs: 5 bits, primer registro fuente.
     - rt: 5 bits, segundo registro fuente.
     - rd: 5 bits, registro destino.

        - shamt: 5 bits, usado solo para instrucciones de desplazamiento (se ignora aquí).
        - funct: 6 bits, función específica para la ALU.
    - **Instrucciones Implementadas**: add, sub, AND, OR, slt.
2. **Load/Store (Carga y Almacenamiento)**:
    - **Campos**:
        - opcode: 6 bits, 35 para lw, 43 para sw.
        - rs: 5 bits, registro base.
        - rt: 5 bits, registro destino (lw) o fuente (sw).
        - address: 16 bits, desplazamiento para la dirección de memoria.
3. **Branch (Rama)**:
    - **Campos**:
        - opcode: 6 bits, 4 para beq.
        - rs: 5 bits, primer registro fuente.
        - rt: 5 bits, segundo registro fuente.
        - address: 16 bits, desplazamiento para calcular la dirección de destino de la rama.
        -

XI. LINEAS DE CONTROL PARA EL DATAPATH

1)

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Para controlar el Datapath, necesitamos generar las siguientes señales de control:

1. **RegDst**: Selecciona el destino del registro (rt o rd).
2. **ALUSrc**: Selecciona la segunda entrada de la ALU (rt o el desplazamiento sign-extendido).
3. **MemtoReg**: Selecciona la entrada al registro destino (resultado de la ALU o dato leído de memoria).
4. **RegWrite**: Habilita la escritura en el registro destino.
5. **MemRead**: Habilita la lectura de memoria.
6. **MemWrite**: Habilita la escritura de memoria.
7. **Branch**: Indica si la instrucción es una rama.
8. **ALUOp**: Controla la operación de la ALU (00: add, 01: sub, 10: función de funct).

XII. FINALIZACIÓN DEL CONTROL

Para completar la implementación de control, debemos definir la función de control usando una tabla de verdad basada en el campo de opcode de 6 bits, Op [5:0]. Esto permitirá especificar las salidas de las líneas de control necesarias para cada instrucción.

Para agregar la instrucción jump, extendemos el Datapath y el control de la siguiente manera:

1. **Formato de la Instrucción Jump**:
    - **Campos**:
        - opcode: 6 bits (000010).
        - address: 26 bits.
2. Datapath **para Jump**:
    - La instrucción jump reemplaza los 28 bits inferiores del PC con los 26 bits de la instrucción desplazados a la izquierda por 2 bits.
3. **Control para Jump**:
    - Se añade una nueva línea de control, Jump, que selecciona la dirección de salto.
    - La lógica de control para jump se activa con el opcode 000010.

a) Implementación del Control en Hardware

La función de control se puede implementar directamente en hardware usando puertas lógicas. Se muestra una estructura básica de cómo se puede implementar:

1. **Decodificador de Opcode**:
    - Un decodificador toma el opcode y genera señales de habilitación para cada tipo de instrucción.
    - Ejemplo: 000000 habilita las señales para instrucciones R-type, 100011 para lw, etc.
2. **Generación de Señales de Control**:
    - Cada salida del decodificador se usa para establecer las líneas de control adecuadas según la tabla de verdad.
3. **Lógica de Control para Jump**:
    - Para jump, se establece Jump = 1 y se desactivan otras señales de control.

XIII. CONCLUSIONES

Mejora en el Rendimiento del Procesador: La optimización de los multiplexores y líneas de control puede influir significativamente en el rendimiento del procesador MIPS. Al reducir la latencia asociada con los multiplexores, se acortan los ciclos de reloj, permitiendo que el procesador ejecute más instrucciones por segundo. Un diseño eficiente de los multiplexores reduce el número de niveles de estos elementos, minimizando la demora en la selección y transmisión de datos. Esto es crucial en aplicaciones que requieren procesamiento rápido de grandes volúmenes de datos, como en gráficos, inteligencia artificial y simulaciones científicas. Además, la optimización de las líneas de control asegura que las señales de comando se propaguen con mayor velocidad y precisión, mejorando la coordinación entre las diferentes unidades funcionales del procesador, lo que resulta en una ejecución más rápida y fluida de las instrucciones.

Reducción del Consumo de Energía: La eficiencia energética es fundamental en el diseño de procesadores, especialmente para dispositivos portátiles y embebidos. Optimizar los multiplexores y las líneas de control contribuye a una significativa reducción en el consumo de energía. Menos transiciones lógicas significan menos cambios de estado en los circuitos, lo que directamente reduce la energía consumida. Además, la optimización puede implicar el uso de técnicas como la reducción de voltaje y la implementación de modos de ahorro de energía cuando partes del procesador no están en uso. La disminución de la carga capacitiva en las rutas de datos y control también juega un papel crucial, ya que menos capacitancia significa menor energía requerida para cargar y descargar las líneas. Esto no solo extiende la vida útil de las baterías en dispositivos móviles, sino que también reduce la generación de calor, mejorando la eficiencia térmica y prolongando la vida del hardware.

Simplificación del Diseño del Circuito: Un diseño optimizado resulta en una arquitectura más sencilla y manejable. La simplificación del diseño del circuito reduce la complejidad de la implementación y facilita el proceso de verificación y prueba. Un diseño más simple implica menos componentes, lo cual reduce el espacio necesario en el chip y puede disminuir los costos de producción. Además, un circuito menos complejo es más fácil de depurar y menos propenso a errores, lo que mejora la confiabilidad del sistema. Esta simplificación también permite una mayor facilidad de modificación y actualización del diseño, lo cual es vital en un entorno tecnológico que avanza rápidamente. La reducción de la complejidad también puede mejorar el rendimiento al minimizar la latencia y el número de etapas en las rutas críticas del procesador.

Aumento de la Escalabilidad: Un diseño escalable es crucial para la adaptabilidad a diversas aplicaciones y tamaños de procesador. La optimización de los multiplexores y líneas de control en una implementación básica de MIPS permite crear una arquitectura modular que puede ser fácilmente ajustada para diferentes necesidades. Por ejemplo, un diseño básico puede ser utilizado en un microcontrolador simple para aplicaciones de IoT, mientras que una versión más avanzada del mismo diseño puede ser utilizada en procesadores de alto rendimiento para servidores. Esta escalabilidad permite a los fabricantes reutilizar gran parte del diseño básico, ahorrando tiempo y costos de desarrollo. Además, una arquitectura escalable facilita la integración de nuevas tecnologías y mejoras sin necesidad de rediseñar todo el sistema desde cero, lo que es esencial para mantenerse competitivo en el mercado.

Mejora en la Confiabilidad y Estabilidad del Sistema: La confiabilidad y estabilidad son cruciales para cualquier sistema de procesamiento, especialmente en aplicaciones críticas como automotriz, aeroespacial y médica. Un diseño optimizado con menos componentes y rutas de control más directas reduce significativamente las posibilidades de fallos. Menos puntos de fallo significan menos posibilidades de errores y un sistema más robusto. La optimización también puede incluir la implementación de técnicas de redundancia y corrección de errores, mejorando aún más la confiabilidad. Además, un diseño más simple y directo reduce la susceptibilidad a interferencias electromagnéticas y variaciones en el entorno operativo, como cambios de temperatura y voltaje. Esto asegura que el procesador funcione de manera consistente y fiable, incluso en condiciones adversas, lo cual es vital para aplicaciones donde la seguridad y la precisión son esenciales.

Referencias

- [1] Bastida, F. (s. f.). Trabajo 1 - MIPS. Universidad de Valladolid. Recuperado de https://www.infor.uva.es/~bastida/OC/TRABAJO1_MIPS.pdf
- [2] (s. f.). DataPath2. Universidad Tecnológica de la Mixteca. https://www.utm.mx/~merg/AC/pdfs/DataPath2.pdf
- Patterson, D. A., & Hennessy, J. L. (2009). Computer organization and design: The hardware/software interface (4ª ed. revisada). Morgan Kaufmann.