Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en Computación

Proyecto Final

Reporte de Trabajo Colaborativo presentado por

Humberto de Jesus Peña Dueñas
Karla Rebeca Hernández Elizarrarás
Elizabeth Arroyo Moreno

Alumnos de Cuarto Semestre ICOM

Arquitectura de Computadoras

Mtro. Jorge Ernesto Lopez Arce Delgado

Guadalajara, Jal. Mayo de 2025

Introducción.

[FASE 1]

La arquitectura MIPS (Microprocessor without Interlocked Pipeline Stages), representa una de las implementaciones más influyentes adoptadas dentro del paradigma RISC (Reduced Instruction Set Computer). Su diseño ha sido reconocido por su simplicidad y facilidad estructural, su eficiencia operativa y su enfoque modular. Gracias a su organización y a la previsibilidad en el tiempo de ejecución de sus instrucciones,el MIPS es un modelo ideal para la enseñanza de conceptos clave como el control de flujo, la segmentación por etapas y el procesamiento paralelo.

El propósito principal de este proyecto final es desarrollar e implementar un datapath de 32 bits basado en la arquitectura MIPS, capaz de interpretar y ejecutar un subconjunto específico de 28 instrucciones pertenecientes a los tres formatos clásicos definidos por esta arquitectura: tipo R el que es registro, tipo I es el inmediato y tipo J que es de salto. Estas instrucciones han sido seleccionadas por su utilidad representativa en programas típicos de nivel básico-intermedio, dejando la validación desde operaciones aritméticas hasta mecanismos de control de flujo y acceso a memoria.

En particular, las instrucciones de tipo R en MIPS son fundamentales para realizar operaciones aritméticas y lógicas que involucran exclusivamente registros. Este formato de instrucción utiliza campos específicos para identificar los registros fuente y destino, así como la operación a ejecutar, sin incluir valores inmediatos. Las instrucciones tipo R, como add, sub, and y or, permiten manipular datos almacenados en los registros de manera directa, lo que contribuye a la eficiencia y rapidez en la ejecución. Además, estas instrucciones mantienen la filosofía RISC al ser simples y uniformes, facilitando su decodificación y ejecución en un ciclo de reloj. La implementación de estas instrucciones en el datapath requiere un manejo preciso de los registros y la unidad aritmético-lógica (ALU), asegurando que las operaciones se realicen correctamente y que los resultados se almacenen en los registros adecuados.

En conjunto, la integración de los formatos de instrucción tipo R, I y J en el datapath permitirá cubrir un amplio espectro de funcionalidades necesarias para la ejecución de programas representativos, consolidando así una base sólida para el estudio y aplicación de la arquitectura MIPS.

Tablas de Instrucciones

Instrucción	Tipo	Sintaxis
Add	R	Add $rd, $rs, $rt
Sub	R	Sub $rd, $rs, $rt
Mul	R	Mul $rd, $rs, $rt
Div	R	Div $rd, $rs, $rt
Or	R	Or $rd, $rs, $rt
And	R	And $rd, $rs, $rt
Addi	I	Addi $rt, $rs, immediate
Subi	I	Subi $rt, $rs, immediate
Ori	I	Ori $rt, $rs, immediate
Andi	I	Andi $rt, $rs, immediate

Instrucción	Tipo	Sintaxis
Lw	I	Lw $rt, $rs, immediate
Sw	I	Sw $rt, offset($rs)
slt	R	slt $rd, $rs, $rt
Slti	I	Slti $rt, $rs, $immediate
beq	I	beq $rs, $rt, offset
bne	I	bne $rs, $rt, offset
J	J	J target
nop	R	nop
bgtz	I	bgtz $rs, offset

[FASE 2]

En esta segunda fase del proyecto, se amplía el diseño del datapath previamente desarrollado para incorporar el soporte de instrucciones tipo I, utilizadas comúnmente para operaciones que involucran un registro y un valor inmediato, como addi, andi, ori, lw y sw. Las instrucciones tipo I son fundamentales para la ejecución de programas más complejos, ya que permiten la manipulación de datos inmediatos y el acceso a memoria. Esta etapa tiene como propósito modificar e integrar los componentes necesarios dentro del datapath, incluyendo el uso de señales de control específicas y la manipulación de direcciones de memoria, para garantizar un funcionamiento correcto de dichas instrucciones.

En la arquitectura MIPS, las instrucciones de tipo I (inmediato) son un formato de 32 bits utilizado principalmente para instrucciones de transferencia de datos, salto condicional y operaciones con operandos inmediatos. Estas instrucciones permiten operar con un registro y un valor inmediato codificado dentro de la propia instrucción, facilitando la manipulación rápida de datos constantes y el acceso a memoria.

Características del formato tipo I en MIPS

- Campos de la instrucción:
    - op (6 bits): código de operación que identifica la instrucción.
    - rs (5 bits): registro fuente, que contiene un operando.
    - rt (5 bits): registro destino o segundo operando.
    - inmediato (16 bits): valor inmediato o desplazamiento, que puede ser un número positivo o negativo (con signo extendido).
- La longitud total de la instrucción es fija en 32 bits, manteniendo la uniformidad en la arquitectura

[FASE 3]

En la arquitectura de computadoras modernas, el pipeline es una técnica fundamental para mejorar el rendimiento de los procesadores. La idea principal es dividir la ejecución de una instrucción en varias etapas, permitiendo que múltiples instrucciones se procesen simultáneamente, cada una en una etapa diferente. Esto incrementa el throughput del procesador, es decir, la cantidad de instrucciones que se completan en un intervalo de tiempo dado.

Sin embargo, para que el pipeline funcione correctamente y sin errores, es necesario implementar mecanismos que controlen el flujo de datos entre etapas. Aquí es donde entran en juego los buffers de pipeline o registros intermedios, que almacenan temporalmente la información entre cada etapa del pipeline.

Este reporte aborda la investigación sobre el pipeline, su funcionamiento, y cómo se implementaron los buffers en el datapath diseñado para ejecutar instrucciones tipo R, I y J en un procesador MIPS de 32 bits, siguiendo las fases indicadas en el proyecto.

¿Qué es un Pipeline?

Un pipeline es una técnica de diseño en la que la ejecución de instrucciones se divide en varias etapas secuenciales, donde cada etapa realiza una parte específica del proceso. En un procesador MIPS típico, estas etapas son:

- IF (Instruction Fetch): Se obtiene la instrucción desde la memoria.
- ID (Instruction Decode): Se decodifica la instrucción y se leen los registros fuente.
- EX (Execute): Se realiza la operación aritmética o lógica.
- MEM (Memory Access): Se accede a memoria para cargar o almacenar datos.
- WB (Write Back): Se escribe el resultado en el registro destino.

Cada una de estas etapas puede operar simultáneamente en diferentes instrucciones, aumentando la eficiencia.

Ventajas del Pipeline

- Incremento del rendimiento: Al procesar varias instrucciones en paralelo, se reduce el tiempo promedio por instrucción.
- Mejor utilización del hardware: Las unidades funcionales no permanecen inactivas esperando a que una instrucción termine completamente.
- Escalabilidad: Permite agregar más etapas para mejorar aún más la frecuencia del reloj.

Desafíos del Pipeline

- Riesgos de datos (Data Hazards): Cuando una instrucción depende del resultado de otra que aún no ha terminado.
- Riesgos de control (Control Hazards): Problemas causados por instrucciones de salto o bifurcación.
- Riesgos estructurales (Structural Hazards): Cuando dos etapas requieren el mismo recurso simultáneamente.
-

Para mitigar estos problemas, se utilizan técnicas como forwarding, stalls, predicción de saltos y, crucialmente, buffers de pipeline.

Implementación de Buffers en el Pipeline

¿Qué son los Buffers de Pipeline?

Los buffers de pipeline son registros intermedios que almacenan temporalmente los datos y señales de control entre las etapas del pipeline. Su función principal es:

- Sincronizar el flujo de datos: Almacenan la información que una etapa ha procesado para que la siguiente etapa pueda acceder a ella en el ciclo siguiente.
- Aislar las etapas: Permiten que cada etapa trabaje independientemente, evitando interferencias directas.
- Mantener la integridad de los datos: Evitan que las señales cambien durante la ejecución de una etapa, garantizando que cada instrucción se procese con los datos correctos.

Registros de Pipeline en MIPS
En un pipeline clásico de 5 etapas, se implementan cuatro registros de pipeline:

Registro de Pipeline	Función Principal
IF/ID	Almacena la instruccion obtenida y el valor de PC+4
ID/EX	Contiene datos decodificados, registros leidos y señales de control para la ejecución
EX/MEM	Almacena resultados de la ALU, datos para memoria y señales de control para el acceso a la memoria
MEM/WB	Contiene datos leidos de memoria o resultado de la ALU para escritura en el banco de registros.

Cada uno de estos buffers es esencial para mantener el orden y la sincronización del pipeline.

Las instrucciones tipo J (como `j` y `jal`) requieren saltos a direcciones absolutas.

- Modificaciones en buffers:

Se ajustaron los buffers para incluir el manejo de la dirección de salto calculada y señales de control para modificar el PC.
Por ejemplo, el buffer IF/ID debe almacenar correctamente la instrucción de salto para que la etapa de decodificación pueda generar la dirección de destino.

- Control de flujo:

Se implementaron mecanismos para manejar el cambio de flujo de instrucciones, evitando que instrucciones incorrectas ingresen al pipeline.

En esta fase se creó un algoritmo en ensamblador usando instrucciones de las tablas indicadas, que luego se decodificó a código binario para precargarlo en la memoria de instrucciones.

- Rol de los buffers:

Los buffers aseguran que las instrucciones precargadas se procesen de manera ordenada y sincronizada, almacenando temporalmente cada instrucción y sus datos asociados a medida que avanzan por el pipeline.

- Interacción con el script de decodificación:

El programa en Python que realiza la decodificación genera el código binario que alimenta la memoria de instrucciones, desde donde el pipeline comienza a operar, utilizando los buffers para mantener la integridad del flujo de instrucciones.

OBJETIVOS

[FASE 1]

Objetivo General.

Diseñar e implementar un datapath MIPS de 32 bits basado en arquitectura RISC, capaz de ejecutar un conjunto específico de 28 instrucciones (tipo R, I y J), simulando el flujo de datos y control en un entorno de pipeline de 5 etapas (IF, ID, EX, MEM, WB).

Objetivos específicos.

1. Analizar las instrucciones MIPS (R, I, J) para definir sus formatos binarios y su comportamiento en el datapath.
2. Diseñar un sistema modular que incluya:
   - Banco de registros.
   - Memorias (instrucciones y datos).
   - ALU, multiplexores y unidad de control.
   - Buffers para gestionar el pipeline.
3. Codificar un programa en ensamblador MIPS y traducirlo a código máquina.
4. Precargar el programa en memoria y simular su ejecución.
5. Verificar el funcionamiento del datapath, asegurando que cada etapa del pipeline opere correctamente.
6. Documentar el proceso desde el diseño teórico hasta la implementación práctica.
7. Comprender cómo un procesador ejecuta instrucciones a nivel hardware.
8. Aplicar conceptos de pipeline, hazards y control de flujo de datos.
9. Integrar conocimientos de lógica digital, programación en Verilog y arquitectura de computadoras.

[FASE 2]

Objetivo General.

El objetivo de esta segunda fase es modificar el datapath diseñado en la fase anterior para que sea capaz de ejecutar instrucciones tipo I, integrando señales de control adicionales que permitan distinguir entre instrucciones tipo R y tipo I, así como implementar el manejo de operandos inmediatos mediante la extensión de signo y su correcta conexión al ALU. Además, se busca añadir la lógica necesaria para soportar instrucciones de acceso a memoria como lw y sw, incluyendo las conexiones con la memoria de datos verificando su correcto funcionamiento a través de simulaciones. Finalmente, esta fase pretende consolidar la comprensión del funcionamiento del datapath completo, abarcando operaciones aritméticas, lógicas y de acceso a memoria utilizando valores inmediatos.

[FASE 3]

Objetivo General.

Agregar los módulos necesarios para completar el datapath de MIPS

de 32 bits para que sea capaz de ejecutar las instrucciones tipo J. Vea

el archivo Fase3.PDF para más detalles.

Crear un algoritmo en código ensamblador, utilizando solo las

instrucciones de la tabla 1 y tabla 2.

o Dicho código debe ser “decodificado” a código binario/maquina, el

cual será pre-cargado a la memoria de instrucciones para que el

datapath lo ejecute.

o Crear un programa/script que realice la “decodificación” de su código

ensamblador (se recomienda Python).

o Vea el archivo Fase4.PDF para más detalles.

Desarrollo.

[FASE 1]

Este proyecto se fundamenta en la arquitectura MIPS de 32 bits, siguiendo el modelo clásico de cinco etapas de pipeline:

- IF (Instruction Fetch): Obtención de la instrucción desde la memoria.
- ID (Instruction Decode): Decodificación y lectura de registros.
- EX (Execute): Ejecución de operaciones aritméticas o lógicas.
- MEM (Memory): Acceso a la memoria de datos para cargar o almacenar valores.
- WB (Write Back): Escritura de resultados en el banco de registros.

Para garantizar la sincronización entre etapas, se implementaron buffers intermedios (IF/ID, ID/EX, EX/MEM, MEM/WB), que evitan conflictos y aseguran un flujo de datos ordenado.

Módulos Principales.

1. PC (Program Counter):

- Responsable de mantener y actualizar la dirección de la siguiente instrucción.
- Incrementa su valor en 4 bytes por ciclo, a menos que se active un salto (Branch o Jump).

2. Memoria de Instrucciones:
   - Almacena el programa en código máquina, precargado mediante un script en Python que traduce instrucciones ensamblador a binario.

3. Banco de Registros:
   - Permite la lectura simultánea de dos registros y la escritura en uno.
   - Utiliza registros específicos para almacenar resultados temporales, punteros y datos intermedios.

4. ALU y ALU Control:
   - Soporta operaciones aritméticas (Add, Sub), lógicas (And, Or) y comparaciones (Slt).
   - La unidad ALU Control interpreta las señales de la Unidad de Control y el campo funct para determinar la operación exacta.

5. Unidad de Control:
   - Genera señales críticas para cada tipo de instrucción (R, I, J), como habilitación de escritura en registros, selección de operaciones ALU y acceso a memoria.

6. Memoria de Datos:
   - Almacena valores requeridos por el programa y soporta operaciones de carga (Lw) y almacenamiento (Sw).

Pruebas y Verificación.

Con el objetivo de asegurar el correcto funcionamiento del datapath, se llevaron a cabo diversas simulaciones de manera progresiva, abarcando cada una de las etapas del flujo de datos. El proceso comenzó con la verificación de la lectura precisa de las instrucciones almacenadas en la memoria, asegurando que se recuperaran correctamente para su posterior ejecución. A continuación, se evaluó el comportamiento de los registros, comprobando que fueran actualizados de forma adecuada con los valores correspondientes durante cada ciclo de instrucción.

Posteriormente, se analizó el flujo de datos entre los diferentes módulos del sistema, prestando especial atención al tránsito de información a través de los buffers entre las distintas etapas del pipeline. Esto permitió verificar que los datos se transfirieran correctamente sin pérdidas ni errores de sincronización.

Asimismo, se monitoreó cuidadosamente el funcionamiento del pipeline al ejecutar instrucciones con dependencias entre sí, con el fin de detectar la presencia de posibles conflictos, tanto de datos como de control, y comprobar si el diseño era capaz de gestionarlos correctamente.

Para todo este proceso de verificación, se utilizó el entorno de simulación ModelSim, el cual permitió observar las señales internas. Se realizaron pruebas detalladas paso a paso, analizando cada transición y resultado, con el propósito de confirmar que el datapath cumpliera con el comportamiento esperado en todas las situaciones evaluadas.

[FASE 2]

En esta segunda fase del proyecto, se implementó en Verilog un procesador MIPS de 32 bits con arquitectura segmentada (pipeline) de cinco etapas: Fetch (IF), Decode (ID), Execute (EX), Memory (MEM) y Write Back (WB). El diseño modular y estructurado facilita la comprensión del flujo de datos e instrucciones dentro del procesador.

Módulos Implementados.

A continuación, se detallan los módulos principales desarrollados para la conformación del datapath segmentado:

Program_counter.v: Este módulo mantiene la dirección de la siguiente instrucción a ejecutar. Se actualiza en cada ciclo de reloj y puede ser modificado por instrucciones de salto o ramificación.

add_4_module.v: Suma 4 al valor actual del contador de programa (PC), asegurando el avance secuencial a la siguiente instrucción.

mem_instrucciones.v: Memoria ROM donde se cargan las instrucciones codificadas en binario desde prueba_binario.txt.

Unidad_control.v: Genera las señales de control necesarias en función del tipo de instrucción (R, I o J), habilitando o deshabilitando módulos y rutas de datos dentro del procesador.

banco_registros.v: Implementa 32 registros de propósito general. Permite dos lecturas simultáneas y una escritura por ciclo de reloj.

ALU.v: Realiza operaciones aritmético-lógicas como suma, resta, multiplicación, división, operaciones lógicas y comparación.

ALU_control.v: Decodifica el campo de función de la instrucción y, junto con las señales de control generales, determina la operación específica a ejecutar en la ALU.

multiplexor.v: Se utilizan en múltiples etapas del datapath para seleccionar entre diferentes rutas de datos, según las señales de control.

memoria_datos.v: Memoria RAM utilizada para realizar operaciones de carga (lw) y almacenamiento (sw).

Datapath_v1.v: Es el módulo principal donde se interconectan todos los componentes anteriores. Define el flujo de señales e información desde la etapa IF hasta WB, controlando el paso de datos y señales de control entre etapas del pipeline.

Archivos de Entrada y Prueba.

datos.txt y prueba_binario.txt: Archivos auxiliares en formato binario que contienen datos e instrucciones de prueba para simular la ejecución de programas MIPS en el procesador implementado. Estos archivos se cargan en las memorias correspondientes durante la simulación.

Compatibilidad de Instrucciones.

El procesador está diseñado para ejecutar un subconjunto de 28 instrucciones del conjunto MIPS, incluyendo:

- Tipo R: add, sub, mul, div, and, or, slt, nop
- Tipo I: addi, subi, ori, andi, lw, sw, slti, beq, bne, bgtz
- Tipo J: j

Estas instrucciones fueron codificadas y cargadas en la memoria de instrucciones para verificar el correcto funcionamiento del procesador en cada una de las cinco etapas del pipeline.

[FASE 3]

En la tercera y cuarta fase del proyecto, se completó la funcionalidad del datapath MIPS de 32 bits mediante la incorporación de instrucciones tipo J (Jump), la integración de buffers intermedios entre etapas y la ejecución de un algoritmo real programado en ensamblador: la búsqueda binaria.

Instrucciones tipo J

Para dar soporte a instrucciones tipo J, se implementó la decodificación del opcode j (000010) en la Unidad de Control (UnidadControl.v), agregando una nueva señal llamada Jump. Esta señal permite seleccionar si el PC (Program Counter) debe ser sobrescrito con una dirección generada a partir del campo inmediato de 26 bits.

Se añadió la lógica correspondiente para calcular la dirección de salto, usando la siguiente fórmula:

jump_address = {PC[31:28], instr[25:0], 2'b00};

Esta dirección se envía como entrada al PC solo si la señal Jump está activa. Para permitir esta sobrescritura, se modificó el módulo PC.v para aceptar una nueva entrada de control. Esta integración permite ejecutar instrucciones j de forma condicional dentro del flujo de control del procesador.

Buffers de pipeline
Se implementaron cuatro buffers intermedios:

* IF_ID.v: transporta la instrucción y el PC+4 de la etapa IF a ID.
* ID_EX.v: lleva datos de registros, el inmediato y señales de control a la etapa EX.
* EX_MEM.v: pasa el resultado de la ALU y señales a la etapa de memoria.
* MEM_WB.v: conecta los datos leídos y los resultados de la ALU a la etapa de escritura en registros.

Estos buffers permiten simular un entorno de pipeline de 5 etapas ordenado y eficiente, previniendo interferencias entre instrucciones y asegurando la integridad del flujo de datos.

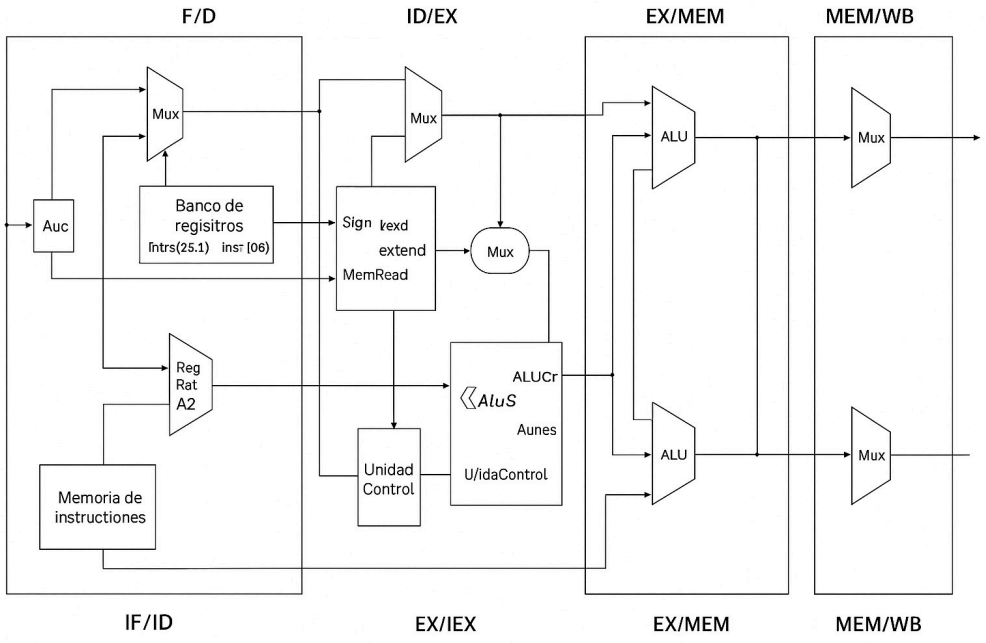
Algoritmo de Búsqueda Binaria

Como parte de la validación final, se implementó el algoritmo de búsqueda binaria en lenguaje ensamblador MIPS, utilizando exclusivamente las instrucciones compatibles con el procesador desarrollado. Este algoritmo compara un valor objetivo con los elementos de un arreglo ordenado cargado en la memoria de datos, actualizando los índices low y high para reducir el espacio de búsqueda.

El código fue estructurado de manera eficiente usando instrucciones como addi, slt, beq, bne, mul, srl, lw y j, y los registros $s0-$s5 y $t0-$t7 para controlar el flujo. El archivo busqueda_binaria.asm fue convertido a código máquina binario utilizando un script de Python (deco_asm_binario.py), generando un archivo binario_prueba.txt para su precarga en la memoria de instrucciones.

Adicionalmente, se construyó un archivo datos.txt con una secuencia de números ordenados representados en binario de 32 bits, simulando un entorno real de prueba. Todo el sistema fue probado en simulación (ModelSim), confirmando el correcto funcionamiento de la instrucción j, el manejo del pipeline, la escritura de registros y el acceso a memoria.

Esta etapa cerró con éxito el ciclo de diseño e implementación de un procesador MIPS capaz de ejecutar instrucciones reales y simular un algoritmo funcional desde la memoria.



Conclusión general.

[FASE 1]

La realización de este proyecto facilitó una comprensión mucho más detallada del funcionamiento interno de un procesador MIPS, particularmente en lo que respecta al procesamiento de instrucciones dentro de un datapath segmentado. A través de la implementación de los diferentes módulos en Verilog, así como la conversión manual de un programa escrito en lenguaje ensamblador a su forma binaria en código máquina, se logró fortalecer de manera significativa tanto los conocimientos teóricos como las habilidades prácticas en el área de arquitectura de computadoras.

Asimismo, la etapa de simulación y verificación del sistema completo permitió observar cómo interactúan los distintos componentes entre sí, haciendo evidente la importancia de la sincronización precisa entre etapas del pipeline. Este proceso ayudó a visualizar el recorrido de los datos a lo largo del datapath y resaltó el papel fundamental del control en la ejecución ordenada de las instrucciones. La experiencia contribuyó a consolidar conceptos clave como el manejo de riesgos de datos, el flujo de control y la importancia de una buena organización modular.

Conclusiones personales.

Humberto

Desde mi perspectiva, este proyecto representó un reto considerable, pero también fue una experiencia muy valiosa en términos académicos y formativos. Me permitió integrar y aplicar conocimientos previos de forma práctica. Aunque enfrentamos dificultades en la integración de módulos y en la identificación de errores, cada obstáculo superado fortaleció mi capacidad de análisis y resolución de problemas. Ver el sistema funcionar correctamente después de varias pruebas fue muy gratificante. El tema me pareció especialmente interesante porque me brindó una visión clara del diseño y funcionamiento de los procesadores, algo esencial para cualquier estudiante de ingeniería en computación.

Rebeca

Este proyecto fue un desafío importante que al mismo tiempo me brindó una experiencia de gran valor académico y personal. Tuve la oportunidad de poner en práctica diversos conocimientos adquiridos en cursos anteriores, enfrentando dificultades como la correcta integración de los módulos y la detección de errores en el programa. Cada uno de estos retos fortaleció mis habilidades para analizar problemas y buscar soluciones efectivas. Fue muy satisfactorio comprobar que, tras múltiples ajustes, el sistema operaba de manera correcta. Considero que trabajar

con la arquitectura MIPS es fundamental para entender el diseño de procesadores y acercar conceptos teóricos a aplicaciones reales.

Elizabeth

Desde mi punto de vista, esta primera fase del proyecto fue un reto significativo que se convirtió en una experiencia enriquecedora tanto académica como formativamente. Pude aplicar de forma práctica los conocimientos aprendidos en materias previas. Aunque hubo momentos complicados, como integrar correctamente los módulos y detectar errores durante la ejecución, cada dificultad superada fortaleció mi capacidad de análisis y resolución de problemas. La satisfacción de ver el sistema funcionando correctamente después de varias pruebas fue muy grande. El estudio de la arquitectura MIPS me pareció especialmente útil, pues permite entender cómo operan los procesadores desde un enfoque de diseño claro y modular, fundamental para nuestra formación como ingenieros en computación.

[FASE 2]

Conclusión general.

La incorporación de las instrucciones tipo I en el datapath MIPS de 32 bits representó un avance significativo en el diseño del procesador, al ampliar su capacidad para ejecutar operaciones que combinan registros con valores inmediatos y acceder a memoria de manera eficiente. Estas instrucciones son esenciales en la arquitectura MIPS, ya que permiten implementar programas más complejos y funcionales, abarcando desde operaciones aritméticas básicas hasta transferencias de datos.

Durante el desarrollo de esta fase, se identificó la importancia de respetar la alineación de direcciones de memoria (en múltiplos de 4 bytes), lo cual contribuyó a simplificar el diseño del hardware y prevenir errores de acceso. Además, la correcta implementación de la extensión de signo para operandos inmediatos, así como su integración con la ALU y la memoria de datos, fue clave para garantizar la funcionalidad del sistema.

Las simulaciones realizadas permitieron verificar el correcto funcionamiento del datapath modificado, facilitando la detección y corrección de errores antes de una implementación física. Este proceso no solo validó la arquitectura propuesta, sino que también reforzó la comprensión del procesamiento de instrucciones tipo I dentro de un entorno segmentado. En conjunto, esta fase consolidó la estructura del procesador, completando su soporte para los tres formatos de instrucciones del conjunto MIPS.

Conclusiones personales.

Humberto
La incorporación de las instrucciones tipo I en el datapath MIPS amplió notablemente las capacidades del procesador, permitiendo manejar no solo registros sino también valores inmediatos y accesos a memoria de forma más eficiente. Esto hizo que el procesador fuera más versátil y capaz de ejecutar programas más complejos, lo que es fundamental para aplicaciones modernas que requieren rapidez y precisión en el procesamiento.

Rebeca

Durante el diseño, aprendí que respetar la alineación de memoria y aplicar correctamente la extensión de signo son detalles técnicos que pueden parecer simples, pero son esenciales para evitar errores y facilitar el trabajo del hardware. Estos aspectos ayudan a que el procesador funcione de manera estable y eficiente, lo que demuestra que un buen diseño debe cuidar tanto la funcionalidad como la precisión en los detalles.

Elizabeth
Las simulaciones fueron una herramienta clave para validar el diseño del datapath con instrucciones tipo I antes de implementarlo físicamente. Gracias a ellas, fue posible detectar y corregir errores a tiempo, lo que ahorra recursos y tiempo en el desarrollo. Además, la simulación ayuda a entender mejor cómo interactúan las diferentes partes del procesador, fortaleciendo el aprendizaje y mejorando el diseño final.

[FASE 3]

Conclusión general.

La implementación del pipeline en el datapath MIPS, junto con el uso adecuado de buffers entre las etapas, es fundamental para lograr un procesamiento eficiente y ordenado de las instrucciones. Los buffers aseguran la sincronización y la integridad de los datos a medida que las instrucciones avanzan por las diferentes fases del pipeline, permitiendo manejar distintos tipos de instrucciones (R, I y J) sin pérdida de información ni conflictos. Este enfoque modular y escalable facilita la extensión del procesador y mejora significativamente su rendimiento, demostrando la importancia de un diseño cuidadoso y progresivo en la arquitectura de computadoras.

Humberto
Este proyecto me permitió reafirmar la importancia pedagógica de enseñar pipeline y buffers con un enfoque práctico y progresivo. La división en fases facilita la comprensión de cómo se construye un procesador desde lo más básico hasta un sistema completo capaz de ejecutar instrucciones complejas. Personalmente, valoro que la implementación de buffers no solo mejora el rendimiento, sino que también es clave para la estabilidad y la correcta ejecución del código, aspectos que suelo enfatizar con mis estudiantes para que comprendan el diseño integral de un procesador.

Rebeca

Durante el desarrollo de este proyecto, comprendí que el pipeline no solo es una cuestión de acelerar la ejecución, sino también de manejar cuidadosamente la comunicación entre etapas. Los buffers me parecieron una solución elegante para evitar que las instrucciones se mezclaran o perdieran datos, lo que me hizo valorar la importancia del diseño detallado en hardware. Además, ver cómo se adaptan los buffers a diferentes tipos de instrucciones me ayudó a entender la flexibilidad necesaria en un procesador real para soportar una variedad de operaciones.

Elizabeth

Desde una perspectiva más técnica, me llamó la atención cómo los buffers actúan como puntos de control que mantienen el flujo de datos estable y sincronizado en un entorno tan dinámico como el pipeline. La implementación fase por fase refleja un buen enfoque para manejar la complejidad creciente al agregar instrucciones tipo I y J. Considero que este método es esencial para evitar riesgos y garantizar que el datapath funcione correctamente, especialmente cuando se trabaja con instrucciones que modifican el flujo de control, como los saltos.

Bibliografías

Abd-El-Barr, M., & El-Rewini, H. (2004). *Fundamentals of computer organization and architecture* (Vol. 38). John Wiley & Sons.

Stalling, W. (2005). Organización y Arquitectura de Computadores (7a ed.) Pearson-Prentice-Hall.

Patterson, D. A. (2007). Computer organization and design: The hardware/software interface. Morgan Kaufmann.

Hwang K. (s.f.). Arquitectura de Computadores y Procesamiento Paralelo. Mc Graw-Hill.

Joyanes Aguilar, L., & Zahonero Martínez, I. (2008). Estructura de datos: algoritmos, abstracción y objetos. McGraw-Hill.

Weiss, M. A. (2012). Estructuras de datos en Java (4ª ed.). Pearson Educación.

Joyanes Aguilar, L., Castillo Sanz, A., Sánchez García, L., & Zahonero Martínez, I. (2009). C. Algoritmos, programación y estructuras de datos. Serie Schaum. McGraw-Hill.