Arquitectura de Computadoras D12
Mtro. J. Ernesto López Arce Delgado

Proyecto Final – Arquitectura de Computadoras

Betancourt Haro Lilia Sofia | 323003201
Plascencia Gutiérrez Irving Yosef | 219873498

Proyecto Final: Procesador MIPS Pipeline y Decodificador Python

Introducción

La arquitectura MIPS (Microprocessor without Interlocked Pipelined Stages) es un estándar académico e industrial fundamental para comprender los procesadores RISC (Reduced Instruction Set Computer). Su diseño se caracteriza por un conjunto de instrucciones simplificado y regular, lo que facilita la implementación de técnicas de paralelismo a nivel de instrucción, específicamente la segmentación o pipelining.

El presente proyecto aborda el diseño, implementación y simulación de un procesador MIPS32 de ciclo único segmentado en 5 etapas (Fetch, Decode, Execute, Memory, Writeback). Para validar el funcionamiento del hardware, se ha desarrollado un ecosistema completo que incluye no solo la descripción del hardware en lenguaje Verilog, sino también una herramienta de software ("toolchain") desarrollada en Python. Esta herramienta funciona como ensamblador y desensamblador, permitiendo la traducción de programas escritos en lenguaje ensamblador a código máquina legible por la memoria del procesador.

La integración de estas dos disciplinas (hardware y software) permite verificar el flujo completo de ejecución de un programa, desde su concepción lógica en ensamblador hasta su procesamiento físico simulado en compuertas lógicas.

Objetivo General

Diseñar, implementar y validar un procesador MIPS32 con arquitectura segmentada (Pipeline), integrando una solución de software para la codificación de instrucciones y demostrando su funcionalidad mediante la ejecución de un algoritmo de filtrado de datos.

Objetivos Particulares

1. **Implementar la arquitectura Pipeline:** Desarrollar en Verilog los módulos correspondientes a las cinco etapas clásicas (IF, ID, EX, MEM, WB) y los registros de segmentación intermedios.
2. **Gestionar Riesgos de Datos (Data Hazards):** Diseñar e integrar una Unidad de Adelantamiento (*Forwarding Unit*) para resolver dependencias de datos sin necesidad de detener el procesador (*stalling*) en la mayoría de los casos.
3. **Desarrollar una Toolchain de Software:** Crear una aplicación en Python con interfaz gráfica capaz de:
    - **Codificar:** Traducir código fuente .asm a archivos de memoria .mem/.bin.
    - **Decodificar:** Realizar ingeniería inversa de archivos binarios a código

ensamblador legible.

4. **Validar mediante Algoritmos:** Diseñar un programa en ensamblador ("Filtro de Paridad") que utilice instrucciones aritméticas, lógicas, de acceso a memoria y saltos condicionales para probar todas las capacidades del procesador.

Desarrollo

Arquitectura de Hardware (Verilog)

El núcleo del proyecto es el archivo `mips_pipeline.v`, que integra los siguientes componentes distribuidos en cinco etapas:

- **Instruction** Fetch (IF): Se instancia el Contador de Programa (PC) y la Memoria de Instrucciones. Se implementa un sumador para calcular PC + 4.
- **Instruction Decode (ID):** Incluye la Unidad de Control principal, el Banco de Registros (Register File) y la Unidad de Extensión de Signo. Aquí se decodifican los opcodes para generar las señales de control.
- **Execute (EX):** Contiene la ALU (Unidad Aritmético-Lógica) y la unidad de control de la ALU. Se implementa la Unidad de Adelantamiento (Forwarding Unit), crítica para este diseño, la cual compara los registros destino de las etapas MEM y WB con los registros fuente de la etapa EX actual, inyectando el dato correcto si se detecta un riesgo.
- **Memory Access (MEM):** Gestiona la Memoria de Datos y la lógica de los saltos condicionales (Branch), decidiendo si se toma el salto basado en la bandera Zero de la ALU.
- **Write Back (WB):** Selecciona entre el resultado de la ALU o el dato leído de memoria para escribirlo finalmente en el Banco de Registros.

Herramienta de Software (Python)

Se desarrolló un script en Python (Codificador.py) utilizando la librería Tkinter para la interfaz gráfica. Esta herramienta cierra la brecha entre el programador y el hardware:

- **Codificador:** Parsea archivos .asm, elimina comentarios y espacios, y traduce mnemónicos (como addi, beq) a su representación binaria de 32 bits según la especificación MIPS. Genera el archivo instrucciones.mem.
- **Decodificador:** Lee cadenas binarias o hexadecimales y reconstruye el código ensamblador original. Esto permitió verificar la integridad de los archivos binarios generados ("Round-trip verification").
- **Manejo de Archivos:** Soporta la carga directa de archivos .asm para facilitar el flujo de trabajo, evitando la copia manual de código.

Programa de Prueba: Filtro de Paridad

Para validar el procesador, se escribió el programa programa.asm. Este algoritmo recorre un

arreglo de 5 números en memoria y realiza una suma acumulativa solo si el número cumple dos condiciones simultáneas:

1. **Magnitud:** El número debe ser mayor estricto a 50 (slt + beq).
2. **Paridad:** El número debe ser impar (andi con 1).

El programa finaliza escribiendo el resultado acumulado en la dirección de memoria 100, lo que permite una fácil verificación en la simulación.

Conclusiones

El proyecto concluye con éxito al lograr una simulación funcional de un procesador MIPS32 segmentado.

1. **Integración Exitosa:** Se demostró la interoperabilidad entre el software desarrollado (Codificador Python) y el hardware simulado (ModelSim). El archivo de memoria generado por Python fue leído y ejecutado correctamente por el procesador Verilog.
2. **Resolución de Riesgos:** La implementación de la Unidad de Adelantamiento (*Forwarding*) fue crucial. Sin ella, las instrucciones aritméticas dependientes (como las usadas en el ciclo de suma del filtro) habrían operado con datos obsoletos, produciendo resultados incorrectos.
3. **Validación Lógica:** La simulación del programa "Filtro de Paridad" arrojó el resultado esperado (suma de 128 para los datos de prueba 55 y 73), validando el correcto funcionamiento de las instrucciones tipo R, tipo I (inmediatas y de memoria) y los saltos condicionales.
4. **Utilidad de la Herramienta:** La capacidad de decodificación de la herramienta Python resultó invaluable para la depuración, permitiendo identificar rápidamente errores de formato en los archivos de memoria durante la fase de desarrollo.

Referencias

Patterson, D. A., & Hennessy, J. L. (2013). *Computer* Organization *and Design: The Hardware/Software Interface* (5th ed.). Morgan Kaufmann. (Base teórica para la arquitectura MIPS y segmentación).

IEEE Standard for Verilog Hardware Description Language (IEEE Std 1364-2005). (Sintaxis y semántica para la implementación en Verilog).