Guía Completa: Cómo Funciona Nuestro Procesador MIPS

Una explicación paso a paso para entender cada componente

Proyecto Final - Arquitectura de Computadoras
Universidad de Guadalajara - CUCEI

1. ¿Qué es un procesador y cómo funciona?

Analogía: El procesador como una fábrica

Imagina que el procesador es una fábrica de ensamblaje de autos. En lugar de construir un auto completo en una sola estación (lo cual tomaría mucho tiempo), la fábrica tiene 5 estaciones de trabajo donde cada una hace una parte del trabajo. Mientras un auto está siendo pintado, otro está siendo ensamblado, y otro está recibiendo el motor. Esto es exactamente lo que hace nuestro procesador con las instrucciones.

Un procesador ejecuta **instrucciones**, que son órdenes muy simples como "suma estos dos números" o "guarda este valor en memoria". Nuestro procesador MIPS puede ejecutar varias instrucciones "al mismo tiempo" gracias a una técnica llamada **Pipeline**.

💡 **Dato clave:** Cada instrucción en MIPS tiene exactamente 32 bits (32 ceros y unos). Estos bits le dicen al procesador qué operación hacer y con qué datos.

2. Las 5 Etapas del Pipeline

Nuestro procesador divide el trabajo en **5 etapas**. Cada instrucción pasa por todas las etapas, una por una:

¿Qué hace cada etapa?

IF - Instruction Fetch (Buscar Instrucción): El procesador va a la memoria de instrucciones y trae la siguiente instrucción a ejecutar. Es como ir a la estantería y tomar el siguiente manual de instrucciones.

ID - Instruction Decode (Decodificar): El procesador "lee" la instrucción y entiende qué tiene que hacer. Identifica qué operación es (suma, resta, etc.) y qué registros necesita.

EX - Execute (Ejecutar): Aquí se hace el trabajo matemático. La ALU (calculadora del procesador) suma, resta, compara, o hace la operación que se necesite.

MEM - Memory Access (Acceder a Memoria): Si la instrucción necesita leer o escribir datos en la memoria RAM, se hace aquí. No todas las instrucciones usan esta etapa.

WB - Write Back (Escribir Resultado): El resultado final se guarda en un registro del procesador para usarlo después.

3. Explicación de Cada Módulo (.v)

Cada archivo **.v** (Verilog) es como una "pieza" del procesador. Vamos a ver qué hace cada uno:

Módulos de la Etapa IF

📍 PC.v - Program Counter

- **¿Qué es?** Un registro que guarda la dirección de la siguiente instrucción a ejecutar.
- **Analogía:** Es como el dedo que usas para seguir la línea que estás leyendo en un libro.
- Normalmente avanza de 4 en 4 (cada instrucción ocupa 4 bytes)
- Puede "saltar" a otra dirección si hay un branch o jump

📚 MemoriaInstrucciones.v - Memoria ROM

- **¿Qué es?** Donde se guardan todas las instrucciones del programa. Solo lectura.
- **Analogía:** Es como un libro de recetas. Puedes leer las recetas, pero no puedes escribir nuevas mientras cocinas.
- Recibe una dirección del PC y devuelve la instrucción de 32 bits

Módulos de la Etapa ID

🎛️ ControlUnit.v - Unidad de Control

- **¿Qué es?** El "cerebro" que decide qué hacer con cada instrucción.
- **Analogía:** Es como un director de orquesta. No toca ningún instrumento, pero le dice a cada músico cuándo y cómo tocar.
- Lee el opcode (primeros 6 bits) y genera señales de control

📂 BancoRegistros.v - Register File

- **¿Qué es?** 32 registros, cada uno capaz de guardar un número de 32 bits.
- **Analogía:** Son como 32 cajones numerados donde puedes guardar y sacar cosas muy rápidamente.
- $zero ($0) siempre vale 0, no se puede cambiar
- Puede leer 2 registros y escribir 1 simultáneamente

↔ SignExtend.v - Extensor de Signo

- **¿Qué hace?** Convierte un número de 16 bits a 32 bits, preservando el signo.
- Ejemplo: 0000000000000101 (5) → 00000000000000000000000000000101

Módulos de la Etapa EX

⚡ ALU.v - Unidad Aritmético-Lógica

- **¿Qué es?** La "calculadora" del procesador.
- Operaciones: ADD (suma), SUB (resta), AND, OR, SLT (comparar)
- Genera señal Zero cuando el resultado es 0 (para branches)

Códigos de operación de la ALU:

Código	Operación	Descripción
0010	ADD	Suma: A + B
0110	SUB	Resta: A - B
0000	AND	Y lógico bit a bit
0001	OR	O lógico bit a bit
0111	SLT	¿A < B? (1 si sí, 0 si no)

Mux.v - Multiplexores

- **¿Qué es?** Un "selector" que elige entre varias entradas.
- **Analogía:** Es como un interruptor de vías de tren. Dependiendo de la posición, el tren va por un camino u otro.

Módulos de la Etapa MEM

MemoriaDatos.v - Memoria RAM

- **¿Qué es?** La memoria donde se guardan los datos del programa (variables, arreglos).
- **Analogía:** Es como un almacén grande. Puedes ir a buscar cosas (LW) o dejar cosas (SW).

Buffers del Pipeline

IF_ID_Buffer.v, ID_EX_Buffer.v, EX_MEM_Buffer.v, MEM_WB_Buffer.v

- **¿Qué son?** Registros de almacenamiento temporal entre cada etapa.
- **Analogía:** Bandejas en una cinta transportadora donde se coloca el producto semi-terminado.
- Permiten que cada etapa trabaje en una instrucción diferente al mismo tiempo

4. Ejemplo: Ejecutando ADD $t2, $t0, $t1

Vamos a ver paso a paso qué pasa cuando el procesador ejecuta la instrucción **ADD $t2, $t0, $t1** (que significa: $t2 = $t0 + $t1).

Instrucción: ADD $t2, $t0, $t1

Binario: 000000 01000 01001 01010 00000 100000
opcode rs rt rd shamt funct

🔴 **Ciclo 1 - IF:** El PC apunta a la dirección. La Memoria de Instrucciones devuelve la instrucción. El Sumador calcula PC+4.

🟠 **Ciclo 2 - ID:** La Unidad de Control lee opcode=000000 y reconoce: "¡Es R-Type!". El Banco de Registros lee $t0=10 y $t1=20.

🟢 **Ciclo 3 - EX:** La ALU recibe A=10, B=20 y la operación ADD. Calcula: 10 + 20 = 30.

🔵 **Ciclo 4 - MEM:** ADD no usa memoria. El resultado (30) simplemente pasa al siguiente buffer.

🟣 **Ciclo 5 - WB:** El Banco de Registros escribe 30 en $t2. ¡Instrucción completa!

✅ **Resultado:** Después de 5 ciclos, el registro $t2 contiene el valor 30.

5. Ejemplo: Ejecutando LW $t3, 0($t0)

Ahora veamos una instrucción de memoria: **LW $t3, 0($t0)** (cargar en $t3 el valor que está en memoria[direccion $t0 + 0]).

Instrucción: LW $t3, 0($t0)

Binario: 100011 01000 01011 0000000000000000
opcode rs rt offset (16 bits)

🔴 **Ciclo 1 - IF:** Se busca la instrucción LW de la memoria.

🟠 **Ciclo 2 - ID:** Opcode=100011 → Control reconoce: "¡Es LW!". Se activa MemRead=1 y MemtoReg=1.

🟢 **Ciclo 3 - EX:** La ALU calcula la dirección: $t0 + offset = 0 + 0 = 0 (dirección de memoria).

🔵 **Ciclo 4 - MEM:** La Memoria de Datos recibe dirección=0. Como MemRead=1, devuelve memoria[0] = 15.

🟣 **Ciclo 5 - WB:** Como MemtoReg=1, se selecciona el dato de memoria (15) y se escribe en $t3.

✅ **Resultado:** $t3 ahora contiene 15, el valor que estaba en memoria[0].

6. Ejemplo: Ejecutando J loop

Veamos la instrucción de salto incondicional: **J loop** (saltar a la etiqueta "loop").

Instrucción: J loop

Binario: 000010 00000000000000000000000101
opcode dirección (26 bits)

🟠 **En el ciclo ID:** Opcode=000010 → Control reconoce: "¡Es JUMP!". La señal Jump=1 activa el MUX del PC.

Cálculo de dirección: PC_nuevo = {PC[31:28], dirección, 00}. Los 4 bits altos del PC se concatenan con los 26 bits de la instrucción y 2 ceros.

⚠️ **Importante:** Cuando hay un JUMP, la instrucción que ya estaba en IF debe descartarse (flush) porque no debe ejecutarse.

✅ **Resultado:** El PC ahora apunta a la dirección de "loop", y la siguiente instrucción será la que está ahí.

7. Resumen Final

¡Lo logramos! Nuestro procesador MIPS Pipeline ejecuta instrucciones en 5 etapas, permitiendo que varias instrucciones se procesen simultáneamente.

Cada módulo Verilog (.v) representa un componente físico del procesador:

- **PC.v** - Sabe dónde estamos en el programa
- **MemoriaInstrucciones.v** - Guarda el programa
- **ControlUnit.v** - Decide qué hacer con cada instrucción
- **BancoRegistros.v** - Almacenamiento rápido de datos
- **ALU.v** - Hace los cálculos matemáticos
- **MemoriaDatos.v** - Almacenamiento de datos del programa
- **Buffers** - Permiten el paralelismo del pipeline

Proyecto Final - Arquitectura de Computadoras
Universidad de Guadalajara - CUCEI

Basado en Patterson & Hennessy y el Manual MIPS32 (MD00086)