Diseño e Implementación de un Procesador RISC-V con Pipeline

Randy Steve Fernández Aguilar
Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica
Email: 2526@estudiantec.cr

Cambronero Solano Verónica
Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica
Email: cambv@estudiantec.cr

Fallas Quirós Ana Cristina
Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica
Email: 2021117825@estudiantec.cr

May 30, 2024

1 Resumen

Este documento presenta la solución del proyecto 01 del curso de Diseño de Sistemas Digitales. Se detallan las etapas del pipeline del procesador RISC-V, incluyendo Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) y Write-back (WB), así como los módulos específicos que soportan cada etapa. También se describen las unidades de control de riesgos que gestionan las dependencias de datos y control. Se presentan los módulos testbench utilizados para garantizar el funcionamiento correcto de cada sección del procesador y se incluye una breve descripción del programa implementado para demostrar el funcionamiento del procesador.

2 Repositorio

Todos los archivos de los módulos diseñados en este proyecto se encuentran disponibles en el siguiente repositorio **Pipeline**, el cual cuenta con una guía corta del contenido de cada carpeta.

3 Diseño del procesador

3.1 Especificaciones generales del Procesador

3.1.1 Conjunto de Instrucciones Implementadas

El diseño del procesador se basa en la arquitectura RISCV, cuyas instrucciones se caracterizan por tener una longitud de 32 bits. Este sistema es capaz de realizar una variedad de operaciones esenciales, como la suma (add), resta (sub), así como operaciones lógicas AND (and) y OR (or). Además, ofrece funcionalidades avanzadas, como el almacenar palabra (sw), carga palabra (lw) y la ejecución de saltos condicionales (beq). Cabe resaltar que se cuenta con un ancho de Palabra de 64 bits.

3.1.2 Etapas del Pipeline

El pipeline del procesador tiene cinco etapas: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), y Write-back (WB).

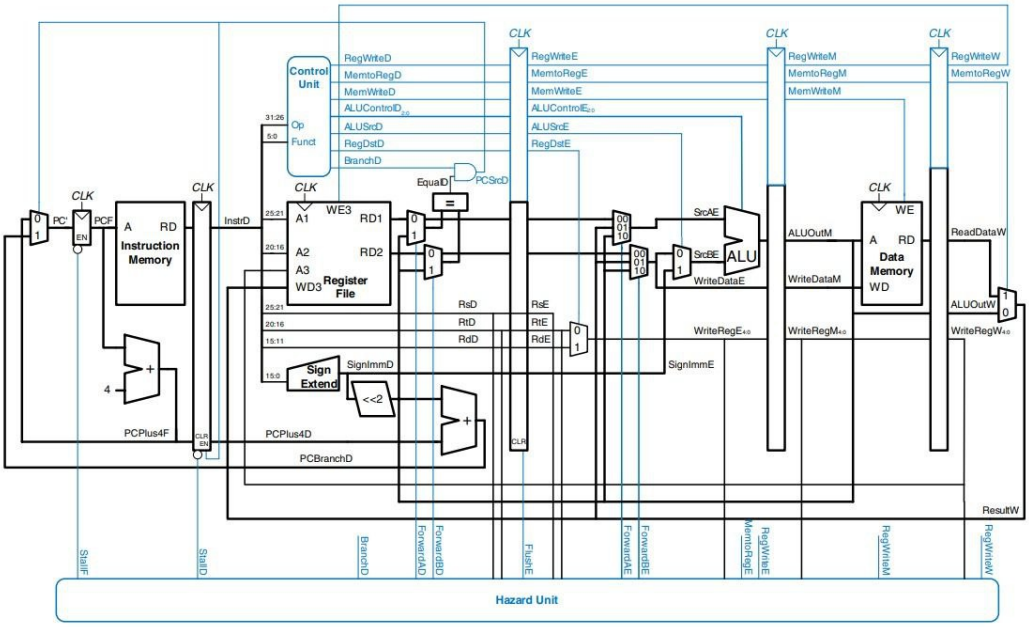
3.1.3 Unidades Funcionales

- **Unidad de Control**: Controla la ejecución de las instrucciones.
- **Banco de Registros**:Contiene 32 registros de 64 bits.
- **ALU (Unidad Lógica y Aritmética)**: Ejecuta operaciones aritméticas y lógicas según las instrucciones decodificadas.
- **Memoria de Datos**: Almacena datos para las operaciones de carga y almacenamiento.
- **Memoria de Instrucciones**: Almacena las instrucciones del programa.
- **Generador de Inmediatos**:Extiende los inmediatos de las instrucciones a 64 bits para su uso en las operaciones.
- **Unidades de Control de Riesgos**: Manejan forwarding y stalling para evitar peligros en el pipeline.

3.1.4 Diagrama del Bloque del Procesador

Para el diseño del procesador del presente proyecto se utilizó un diagrama de bloques típico del procesador RISC-V, el cual se encuentra en la Figura 1, y se tomó del capítulo 7 del libro [2].

Figure 1: Diagrama de la Microarquitectura Pipeline



4 Descripción del Pipeline

4.0.1 Descripción detallada de las etapas del pipeline

Instruction Fetch (IF): Esta etapa se encarga de obtener la instrucción actual desde la memoria de instrucciones y actualizar el program counter para la siguiente instrucción en el pipeline.

Módulos utilizados en esta etapa:

- **MUX**
- **PC**
- **Instruction Memory**
- **Adder PC + 4**
- **Registro IF_ID**

Instruction Decode (ID): Esta etapa se encarga de decodificar la instrucción obtenida en la etapa de Instruction Fetch (IF) y de preparar los datos necesarios para la etapa de ejecución (EX). Durante esta etapa, se determinan las

señales de control y se leen los datos de los registros fuente necesarios para la instrucción actual.

Módulos utilizados en esta etapa:

- **Registro IF_ID**
- **Control Unit**
- **Register Bank**
- **Generador de inmediatos**
- **Comparador**
- **Shift Left**
- **Adder**
- **Registro ID_EX**

Execute (EX) Esta etapa se encarga de realizar las operaciones aritméticas y lógicas, así como de calcular las direcciones de memoria para las instrucciones de carga y almacenamiento. Aquí, se toman los datos preparados en la etapa ID y se ejecutan las operaciones especificadas por la instrucción.

Módulos utilizados en esta etapa:

- **Registro ID_EX**
- **ALU**
- **Muxes para Forwarding**
- **Multiplexor para ALUSrc**
- **Registro EX_MEM**

Memory (MEM) En esta etapa, se realizan operaciones de lectura y escritura en la memoria de datos, en caso de instrucciones de carga y almacenamiento respectivamente. También se manejan las señales de control relacionadas con el acceso a la memoria.

Módulos utilizados en esta etapa:

- **Registro EX_MEM**
- **Data Memory**
- **Registro MEM_WB**

Write Back (WB) En esta etapa, se escriben los resultados finales en el banco de registros. Los datos que se escriben pueden provenir de la ejecución de operaciones aritméticas y lógicas, de lecturas de memoria o de operaciones de carga con signo extendido.

Módulos utilizados en esta etapa:

- **Registro MEM_WB**
- **Register Bank**
- **Multiplexor para MemtoReg**

4.0.2 Diseño de los Módulos de los Registros del Pipeline

- **Registro IF_ID**

  El módulo diseñado se encarga de almacenar la instrucción y el valor del contador de programa (PC) obtenidos en la etapa de Instruction Fetch (IF) para ser utilizados en la etapa de Instruction Decode (ID). Este módulo utiliza un registro de pipeline para retener estos valores hasta el siguiente ciclo de reloj. El módulo implementa un registro de pipeline que se actualiza en cada flanco positivo del reloj. Si la señal de reinicio (rst) o la señal de vaciado (flush) están activadas, el registro se resetea a 0. Si la señal de control (`PCSrcD_Control`) está desactivada, el registro se actualiza con la nueva instrucción y el nuevo valor del PC. De lo contrario, mantiene sus valores actuales. La salida del módulo (instruction_out y out_pc) refleja los valores almacenados en el registro.

- **Registro ID_EX**

  Este módulo se encarga de almacenar los valores decodificados y las señales de control obtenidas en la etapa de Instruction Decode (ID) para su uso en la etapa de Execute (EX). Este módulo utiliza un registro de pipeline para retener estos valores hasta el siguiente ciclo de reloj. El registro de pipeline se actualiza en cada flanco positivo del reloj (`posedge clk`). Si la señal de reinicio (`rst`) está activada, el registro se restablece a sus valores predeterminados. Si la señal de control AluSrc_in, MemtoReg_in, RegWrite_in, MemRead_in, MemWrite_in, Aluop_in, rs1Data_in, rs2Data_in, rs_in, rt_in, rd_in o immediate_in cambia, el registro se actualiza con los nuevos valores. De lo contrario, mantiene sus valores actuales. La salida del módulo (AluSrc_out, MemtoReg_out, RegWrite_out, MemRead_out, MemWrite_out, Aluop_out, rs1Data_out, rs2Data_out,rs_out, rt_out, rd_out, immediate_out) refleja los valores almacenados en el registro.

- **Registro EX_MEM**

  El módulo EX_MEM se encarga de almacenar los resultados de la etapa de ejecución (EX) y las señales de control asociadas para su uso en la etapa

de memoria (MEM). También proporciona los datos y señales de control necesarios para la siguiente etapa del pipeline. El registro de pipeline se actualiza en cada flanco positivo del reloj (posedge clk). Cuando la señal de reinicio (reset) está activada, todas las señales se restablecen a sus valores predeterminados. De lo contrario, las señales se actualizan con los valores de entrada. Las señales de salida (RegWrite_Out, MemtoReg_Out, MemWrite_Out, MemRead_out, AluOut, DataOut, Rd_out) reflejan los valores almacenados en el registro de pipeline. Este módulo proporciona una interfaz entre la etapa de ejecución (EX) y la etapa de memoria (MEM), asegurando que los datos y las señales de control se transfieran correctamente entre estas etapas del pipeline.

- **Registro MEM_WB**

  Para el diseño de este módulo se creó una lógica que se encarga de almacenar los resultados de la etapa de memoria (MEM) y las señales de control asociadas para su uso en la etapa de write back (WB). También proporciona los datos y señales de control necesarios para la siguiente etapa del pipeline. El registro de pipeline se actualiza en cada flanco positivo del reloj (posedge clk) o en caso de un flanco positivo de la señal de reinicio (reset). Cuando la señal de reinicio está activada, todas las señales se restablecen a sus valores predeterminados. De lo contrario, las señales se actualizan con los valores de entrada. Las señales de salida (RegWrite_Out, MemtoReg_Out, AluOut, DataOut, Rd_out) reflejan los valores almacenados en el registro de pipeline. El módulo MEM_WB establece una conexión esencial entre la etapa de memoria (MEM) y la etapa de write back (WB) del pipeline.

4.0.3 Diseño de los módulos de Control de Riesgos

Los riesgos en la ejecución son situaciones que pueden provocar que el procesador no avance de manera eficiente. Estos riesgos pueden manifestarse como conflictos en el acceso a los datos o en la secuencia de ejecución de las instrucciones, lo que puede resultar en errores o en la ralentización del rendimiento del sistema. Los dos tipos principales de riesgos son los riesgos de datos y los riesgos de control. Para gestionar estos riesgos en este proyecto se utilizan unidades especializadas, como las unidades de avance (forwarding) y las unidades de detección de riesgos (hazard detection units), que garantizan una ejecución fluida y eficiente del programa.

- **Riesgo de Datos**:
  - `Unidad de Forwarding`: Para abordar los riesgos de dependencias de datos, se implementa una Unidad de Forwarding que detecta y resuelve conflictos de datos entre etapas del pipeline. Esta unidad monitorea las señales de escritura de registros en las etapas EX/MEM y MEM/WB, y reenvía los datos necesarios directamente desde estas etapas a las etapas anteriores para evitar esperas innecesarias.

El módulo forwardunit determina si los registros necesarios para la ejecución de una instrucción están siendo modificados en las etapas posteriores del pipeline. Si es así, activa las señales de reenvío (forwardA y forwardB) correspondientes para permitir que la etapa actual acceda a los datos actualizados sin tener que esperar a que se escriban en los registros del banco de registros.

  - `Unidad de Hazard`: Además, se utiliza una Unidad de Hazard para detectar situaciones de riesgo durante la ejecución de instrucciones de carga (lw). Estas instrucciones pueden causar conflictos de datos si se intenta acceder a los registros antes de que los datos cargados estén disponibles. La Unidad de Hazard controla el flujo del programa y, si se detecta un riesgo, detiene temporalmente el avance del pipeline mediante la activación de la señal SignalPC, lo que impide que el contador de programa se actualice y permite que la instrucción actual se repita hasta que los datos necesarios estén listos para su uso. Estas medidas garantizan una ejecución suave y eficiente de las instrucciones, minimizando los tiempos de espera y optimizando el rendimiento del procesador RISC-V.

- **Riesgos de Control**: La unidad de riesgos de control en el procesador RISC se encarga de detectar y manejar situaciones donde se requiere tomar decisiones basadas en el resultado de operaciones de comparación, típicamente utilizadas en instrucciones de salto condicional (beq). Esta unidad utiliza un comparador que compara los valores de dos registros relevantes y genera una señal de salida que indica si los valores son iguales o no. El módulo utilizado para esta función es el comparador. Este módulo toma dos entradas, dato_rs1 y dato_rs2, que representan los valores de los registros que se están comparando. Luego, determina si los valores son iguales y establece la señal de salida resultado en alto si lo son, o en bajo si no lo son. La salida del comparador se utiliza en conjunto con la unidad de control branch, que es el selector del multiplexor en la etapa de Instruction Fetch (IF). Esta unidad toma la decisión de si se debe tomar un desvío en el flujo de ejecución del programa basado en el resultado de la comparación. Si el resultado es positivo (es decir, los valores de los registros son iguales), se activa la señal de control correspondiente para dirigir el flujo hacia el objetivo del salto condicional.

5 Descripción de módulos clave

5.1 Unidad de Control

La Unidad de Control (UC) es responsable de generar las señales de control necesarias para dirigir las operaciones del procesador. Se basa en el opcode y otras partes de la instrucción para determinar las señales de control adecuadas para cada etapa del pipeline. Las señales de control producidas incluyen `RegWrite`, `ALUSrc`, `MemWrite`, `MemtoReg`, `PCSrc`, `ImmType`, y `MemRead`.

El módulo `Unit Control` implementado en el trabajo está diseñado para determinar las señales de control en base al `OpCode` recibido. A continuación, se detallan algunas de las configuraciones:

- **Tipo R (OpCode = 7'b0110011)**: Este tipo de instrucción se utiliza para operaciones aritméticas y lógicas entre registros.
  - `AluR = 3'b000: outcome = 10'b001000_10_11` (ADD y SUB)
  - `AluR = 3'b111: outcome = 10'b001000_00_11` (AND)
  - `AluR = 3'b110: outcome = 10'b001000_01_11` (OR)
- **LOAD (OpCode = 7'b0000011)**: Esta instrucción carga un valor desde la memoria al registro.
  - `outcome = 10'b111100_10_00`
- **STORE (OpCode = 7'b0100011)**: Esta instrucción almacena un valor desde el registro a la memoria.
  - `outcome = 10'b110110_10_01`
- **BRANCH (OpCode = 7'b1100011)**: Esta instrucción realiza un salto condicional.
  - `outcome = 10'b000001_10_10`

5.2 ALU

La ALU realiza operaciones aritméticas y lógicas en los operandos de entrada. Las operaciones específicas que puede realizar son determinadas por las señales de control de la ALU (`ALUControl`).

El módulo `Alu` opera según la señal de selección, que determina cuál de las operaciones (AND, OR, suma) se debe realizar. El submódulo `SumaC2` se encarga de las operaciones de suma, proporcionando tanto el resultado de la suma como el *carry-out*.

Un bloque de lógica combinacional dentro del módulo determina la operación a realizar en base a la señal de selección. El resultado de esta operación se asigna a la salida del módulo. La salida del *carry-out* de la suma se conecta directamente al *carry-out* generado por el submódulo `SumaC2`.

5.3 Memoria

La memoria en el procesador se divide en memoria de instrucciones y memoria de datos. La memoria de instrucciones almacena las instrucciones que el procesador debe ejecutar, mientras que la memoria de datos almacena los datos que las instrucciones manipulan. Estas memorias son accedidas mediante direcciones específicas generadas durante la ejecución de las instrucciones.

- **DataMemory**: El módulo `DataMemory` utiliza un arreglo de memoria (`MEMO`) implementado como un arreglo bidimensional de 8 bits, lo que resulta en una memoria de 32 registros de 64 bits. La dirección de memoria se utiliza como un índice para acceder a los datos en el arreglo `MEMO`. El comportamiento del módulo está definido por una asignación condicional en la que los datos de salida (`dataout`) se determinan en función de las señales de lectura (`r`) y escritura (`w`). Durante una lectura (`r` activo), se accede al contenido de la memoria y se envían los datos correspondientes a la salida. Durante una escritura (`w` activo), los datos de entrada (`datain`) se escriben en la dirección de memoria especificada.

- **InstructionMemory**: El módulo `InstructionMemory` utiliza un arreglo de memoria (`Memory`) para almacenar las instrucciones. Este arreglo está dimensionado por el parámetro `size` y se inicializa con las instrucciones deseadas en la sección `initial` del código. Para proporcionar la instrucción solicitada, el módulo utiliza la dirección de memoria (`adr`) para acceder a las ubicaciones correspondientes en el arreglo `Memory`. Luego, las partes relevantes de la instrucción se concatenan para formar la instrucción de salida de 32 bits.

5.4 RegisterBank

El banco de registros es un pequeño conjunto de registros rápidos dentro del procesador que almacena datos temporales. Los registros son utilizados para almacenar operandos de las instrucciones y resultados intermedios. El módulo `RegisterBank` utiliza un arreglo de memoria (`Bank`) para almacenar los valores de los registros. Cada registro se representa como un elemento en el arreglo de memoria. Cuando la señal de escritura (`regwrite`) está activa y se proporciona una dirección de registro válida (`register3`), el nuevo valor de entrada (`datain`) se escribe en el registro correspondiente en el arreglo de memoria.

La lectura de los valores de los registros especificados por `register1` y `register2` se realiza de manera síncrona en el flanco de subida del reloj (`clk`). Los valores de los registros solicitados se proporcionan como salidas (`dataout1` y `dataout2`) y se utilizan en las operaciones del procesador.

5.5 Multiplexores y Otros Componentes Importantes

Los multiplexores (`mux`) se utilizan para seleccionar entre varias señales de entrada y producir una sola señal de salida. Otros componentes importantes incluyen unidades de desplazamiento, generadores de inmediatos, unidades de comparación y unidades de avance de datos para manejar peligros de datos en el pipeline.

6 Verificación de funcionamiento de los bloques

Con el fin de verificar el correcto funcionamiento del Procesador Pipeline, se desarrollaron archivos Testbench específicos para cada uno de los módulos. En este proceso, se han empleado comandos tales como $dumpfile("")$ y $dumpvars()$ para generar archivos .vcd, los cuales permiten la visualización de los datos en GTK Wave. Esta herramienta posibilita el análisis del comportamiento de las señales y su interacción a lo largo del procesador, facilitando así la identificación de posibles fallos o áreas de mejora.

6.1 Prueba de bloques reutilizados

6.1.1 Prueba Módulo Adder

El módulo `Adder_tb` es un testbench diseñado para verificar el funcionamiento del módulo adder, que implementa un sumador. Se definen señales de entrada (a y b) y una salida (out). Se inicializan las entradas con valores específicos y se espera un tiempo para la estabilización antes de finalizar la simulación. La salida (out) se monitoriza para verificar la suma correcta de los valores de entrada.

6.1.2 Prueba Módulo Alu

El módulo `Alu_tb` es un testbench diseñado para verificar el funcionamiento del módulo Alu, que representa una Unidad Lógica Aritmética. Este testbench simula diversas condiciones de entrada y monitorea las salidas correspondientes. Comienza definiendo parámetros y señales de entrada/salida. Luego, instancia el módulo Alu y genera estímulos iniciales y cambios en las entradas. Se monitorizan las salidas del módulo Alu en función del flanco de subida del reloj. Este proceso proporciona información sobre el comportamiento del módulo Alu bajo diferentes condiciones de entrada.

6.1.3 Prueba Módulo Clock

El módulo `Clock_tb` es un testbench para simular un reloj digital. Utiliza el módulo Clock para generar la señal de reloj CLK. Comienza configurando la escala de tiempo y definiendo la instancia del módulo Clock. En el bloque initial, inicializa la simulación, establece la señal CLK en bajo, espera 10 unidades de tiempo y luego alterna el valor de CLK cada 100 unidades de tiempo durante 20 iteraciones. Finalmente, finaliza la simulación.

6.1.4 Prueba Módulo Control Unit

El módulo `Control_Unit_tb` es un testbench para probar la unidad de control `Control_Unit`. Se definen constantes y señales de entrada/salida. Se generan estímulos cambiando el OpCode para probar diferentes combinaciones de operaciones. En el bloque always, se monitorizan las salidas del módulo `Control_Unit`

y se muestran en la consola los valores de OpCode, Branch, MemRead, entre otras, en función del cambio en OpCode. Esto proporciona una forma de verificar el comportamiento del módulo bajo diferentes condiciones de entrada.

6.1.5 Prueba Módulo DataMemory

El módulo `Data_Memory_Testbench` se encarga de verificar el correcto funcionamiento del módulo DataMemory, que representa una memoria de datos. Este testbench establece señales de entrada como la dirección (adr), los datos de entrada (datain), y señales de control de escritura y lectura (w, r), además de definir una señal de salida (dataout). Durante la simulación, se generan estímulos mediante cambios en estas señales, mientras que un ciclo de reloj se simula con un período definido. Las salidas del módulo DataMemory son monitoreadas en cada ciclo de reloj para verificar su comportamiento bajo diversas condiciones de entrada.

6.1.6 Prueba Módulo Immediato

El módulo `immgen_tb` es un testbench diseñado para verificar el funcionamiento del módulo immgen, que genera inmediatos para el procesador. Se definen señales de entrada (instr y ImmSrc) y una salida (Out). Se simulan diferentes casos de prueba, como tipos de inmediatos (Tipo I, Tipo S, Tipo B) y un caso por defecto. Se monitoriza la salida (Out) para verificar la generación correcta de inmediatos.

6.1.7 Prueba Módulo InstructionMemory

El módulo `InstructionMemory_Testbench` es un testbench diseñado para evaluar el correcto funcionamiento del módulo InstructionMemory, el cual simula una memoria de instrucciones. Se establecen parámetros como el periodo del reloj `CLOCK_PERIOD` y el tamaño de la memoria `(MEM_SIZE)`. Durante la simulación, se generan estímulos aleatorios para realizar lecturas de instrucciones, donde se seleccionan direcciones de memoria de manera aleatoria y se espera un ciclo de reloj entre cada lectura. Además, se simula el comportamiento del reloj. Este testbench permite verificar cómo responde el módulo InstructionMemory ante distintas lecturas de instrucciones.

6.1.8 Prueba Módulo Multiplexor

El módulo `Multiplexor_tb` es un testbench para evaluar el módulo Multiplexor, que actúa como un multiplexor. Se definen parámetros como el ancho de los datos y el tiempo total de simulación. Durante la simulación, se generan estímulos cambiando las señales de entrada y la señal de control. Se monitorizan las salidas del multiplexor y se muestra su comportamiento en la consola. Este testbench permite verificar cómo el multiplexor selecciona entre las entradas proporcionadas bajo diferentes condiciones de control. Una vez completada la simulación, esta termina.

6.1.9 Prueba Módulo PC

El módulo `PC_Testbench` es un testbench para evaluar el módulo 'pc', que representa un contador de programa. Durante la simulación, se inicializan señales como el reloj ('clk'), la señal de reset ('rst'), y el valor inicial del contador de programa ('oldpc'). Después de un periodo de reloj, se desactiva la señal de reset y comienza la operación normal del contador de programa. Se generan ciclos de reloj mediante un bucle 'repeat', donde el contador de programa se incrementa en 4 en cada ciclo. Además, se simula el comportamiento del reloj alternando su valor en cada medio periodo. Este testbench permite verificar cómo el contador de programa responde al avance del reloj y al incremento de su valor en cada ciclo, lo que representa el progreso del programa. Una vez completados los ciclos de reloj especificados, la simulación concluye.

6.1.10 Prueba Módulo ShiftUnit

El testbench `ShiftUnit_TB` evalúa el módulo ShiftUnit, el cual realiza desplazamientos de bits. Se inicializa la simulación asignando un valor hexadecimal a la señal de entrada `input_data`. Se muestra el valor inicial y luego el resultado de la salida `output_data` después del desplazamiento. Se verifica si la salida es igual al resultado esperado (el desplazamiento de `input_data` a la izquierda por un bit). Si la prueba es exitosa, se muestra un mensaje indicándolo; de lo contrario, se muestra un mensaje de prueba fallida. La simulación termina después de estas verificaciones.

6.1.11 Prueba Módulo SumaC2

El testbench `SumaC2_Testbench` evalúa el módulo SumaC2, el cual suma dos números en complemento a 2. Se definen señales de entrada para los operandos y el acarreo, así como señales de salida para el resultado y el acarreo de salida. Se generan estímulos cambiando las entradas para probar diferentes casos de suma. La simulación registra los cambios en las señales de entrada y muestra las salidas correspondientes en la consola. Esto permite verificar el comportamiento del módulo bajo diversas condiciones de entrada. Una vez completada la simulación, esta termina.

6.2 Pruebas de módulos clave

`Unidades de Riesgos`

6.2.1 Prueba Módulo Hazard

El testbench `hazard_U_tb` se utiliza para verificar el comportamiento del módulo Hazard_U, que es un componente diseñado para detectar riesgos de datos en el procesador. Este testbench establece diferentes señales de entrada, como la dirección del registro de destino (R_d), la señal de lectura de memoria (MemRead), y la instrucción actual (Instruction). Luego, simula un escenario donde

una instrucción requiere leer datos de un registro (referenciado por R_d) que aún no ha sido escrito debido a la presencia de una instrucción de lectura de memoria pendiente. La salida SignalPC indica si se detectó un riesgo de datos, lo que podría requerir la inserción de burbujas en el pipeline del procesador para evitar problemas de dependencia de datos. La simulación termina después de verificar el resultado de la detección de riesgos.

6.2.2 Prueba Módulo Forward Unit

El testbench `forwardunit_tb` evalúa el funcionamiento del módulo forwardunit, que es una unidad de reenvío (forwarding unit) diseñada para el procesador. El testbench configura varias señales de entrada, como los registros de entrada (Registro1 y Registro2), las señales de control de escritura (ex_regwrite y wb_regwrite), y las señales de identificación de registros (Rd_execute y Rd_writeback). Luego, simula diferentes escenarios de reenvío, donde los datos pueden ser reenviados desde etapas anteriores del pipeline hacia etapas posteriores para evitar conflictos de dependencia de datos. Cada prueba se ejecuta durante un tiempo específico y se monitorean las señales de salida forwardA y forwardB, que indican si se realizó un reenvío para los operandos de la ALU. Finalmente, la simulación termina después de un cierto tiempo.

```
Registros
```

6.2.3 Prueba Módulo Registro IF/ID

El testbench `IF_ID_tb` evalúa el módulo IF_ID, parte del procesador pipeline. En el bloque initial, se inicializan las variables necesarias para la simulación. Se establece el reloj y se desactiva el reset después de un breve periodo. Luego se inyectan datos de entrada como una instrucción y un contador de programa (PC).Después de otro ciclo de reloj, muestra las señales de salida. Finalmente, la simulación termina.

6.2.4 Prueba Módulo Registro ID/EX

El testbench `ID_EX_tb` verifica el módulo ID_EX del procesador pipeline. En el bloque initial, se inicializan las variables necesarias para la simulación. Se establece el reloj y se desactiva el reset después de un breve periodo. Luego se asignan valores a las señales de entrada. Después de un ciclo de reloj, se muestran los valores de las señales de salida. Finalmente, la simulación termina con finish.

6.2.5 Prueba Módulo Registro EX/MEM

El testbench `EX_MEM_tb` verifica el funcionamiento del módulo EX_MEM, que forma parte del procesador pipeline. Este testbench configura las señales de entrada y simula un ciclo de reloj. Luego, se muestran los valores de las señales de salida (RegWrite_Out, MemtoReg_Out, MemWrite_Out, AluOut, DataOut,

Rd_out). y verifica si coinciden con los valores esperados. Si la verificación es exitosa, la simulación termina; de lo contrario, se muestran mensajes de error.

6.2.6 Prueba Módulo Registro MEM/WB

El testbench `MEM_WB_tb` verifica el módulo MEM_WB del procesador pipeline. En el bloque initial, se inicializan las variables necesarias para la simulación. Se establece el reloj y se desactiva el reset después de un breve periodo. Luego se asignan valores a las señales de entrada. Después de un ciclo de reloj, se muestran los valores de las señales de salida. Finalmente, la simulación termina con finish.

6.3 Prueba del Módulo Top del Procesador

El procesador contiene un InstructionMemory con un set de instrucciones las cuales fueron ensambladas utilizando RISC-V Assembly and Routine Simulator (RARS), las cuales fueron las siguientes:

```
add x3,x2,x4
sub x5,x3,x8
lw x9, 100(x5)
loop:  sw x7,(x9)
lw x7,20(x1)
add x3,x4,x5
sub x5, x8, x3
beq x1,x5,loop
```

Se realizó un Top que a su vez es el Testbench que se utiliza para probar el Procesador, esto se realizó de esta manera por facilidad.

En la prueba se utilizaron:

Declaraciones y configuraciones iniciales:

- `default_nettype none`: Esta directiva establece que todos los identificadores no explícitamente declarados son considerados como no definidos. Esto ayuda a detectar errores de tipografía en el código.
- `timescale 1ns/10ps`: Establece la escala de tiempo para la simulación.
- `include`: Se incluyen todos los archivos necesarios para el diseño del procesador. Esto asegura que todas las definiciones de módulos y señales estén disponibles para la simulación.

Declaraciones de módulos y cables:

- Se declaran todas las señales de entrada y salida necesarias para conectar los diferentes módulos del procesador. Esto incluye señales para el reloj (`clk`), señales de control (`RegWrite`, `MemtoReg`, etc.), y señales de datos (como `instruction_fetch`, `output_alu_ex`, etc.).

Configuración del reloj y del reset:

- Se utiliza un módulo `Clock` para generar el pulso de reloj necesario para sincronizar las operaciones del procesador. El reset se inicializa y se desactiva después de un cierto tiempo.

El procesador sigue un ciclo de ejecución que incluye varias etapas clave: En la Etapa del Instruction Fetch, se recupera la siguiente instrucción de la memoria utilizando el contador de programa y se envía al siguiente registro del pipeline para su procesamiento. Luego, en la Etapa del Instruction Decode, la instrucción se decodifica para extraer información como los registros de origen y destino, y se generan las señales de control correspondientes. La lógica del branch y parte de la lógica del PC manejan las instrucciones de branch y determinan la dirección de la próxima instrucción a ejecutar. En la Etapa del execute, se ejecuta la operación especificada por la instrucción, utilizando multiplexores para seleccionar los datos de origen adecuados para la ALU. En la Etapa de MEM, se accede a la memoria de datos si es necesario y los datos leídos se envían al siguiente registro del pipeline para su procesamiento. Los registros de pipeline intermedios se utilizan para almacenar temporalmente resultados durante las diferentes etapas del procesamiento, facilitando el control de flujo y la detección de riesgos. Finalmente, las Unidades de control de hazards, como forwardunit y Hazard_U, detectan y manejan riesgos de datos y control en el pipeline, asegurando una ejecución correcta de las instrucciones mediante el reenvío de datos desde etapas anteriores si es necesario.

Inicialización de la simulación y finalización:

- Se inicializa la simulación y se establece un punto de finalización después de un cierto número de ciclos de reloj. Esto asegura que la simulación se ejecute durante el tiempo suficiente para observar el comportamiento del procesador en diferentes condiciones de entrada.

7 Conclusiones

- El diseño e implementación del procesador RISC-V con pipeline permitió adquirir un conocimiento profundo sobre las etapas clave de la arquitectura de un procesador moderno y la importancia de la coordinación eficiente entre sus unidades funcionales.
- Las unidades de forwarding y hazard detection fueron cruciales para evitar conflictos y retrasos, asegurando una ejecución fluida y optimizada del pipeline, lo que mejoró significativamente el rendimiento del procesador.
- La utilización de módulos testbench fue esencial para verificar el funcionamiento de cada submódulo, permitiendo identificar y corregir errores tempranamente, asegurando la fiabilidad y el desempeño correcto del procesador RISC-V.

References

[1] Patterson, D., *Computer Organization and Design*, Morgan Kaufmann Publishers, 2018.

[2] David Money Harris y Sarah L. Harris, *Digital Design and Computer Architecture, RISC-V Edition*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2022. ISBN: 978-0-12-820064-3.