Proyecto Final programa en código ensamblador y código maquina

Introducción

Objetivo del documento:

Este documento tiene como objetivo proporcionar una detallada información sobre cada uno de los módulos, mencionando sus funcionalidades y el como se integran entre sí para formar este un sistema funcional, así llevando a cabo un mejor uso del código. Se pretende dar una guía sencilla y comprensible para el usuario con objetivo que así conozca lo necesario para poder comprender y ejecutarlo sin preocupaciones.

Alcance

Este documento pretende cubrir la descripción de todos los módulos, sus funciones y su integración en un sistema completo.
Se proporciona información más detallada sobre cada módulo, su funcionamiento y señales relevante, así como instrucciones paso a paso para su integración. Este enfoque ayuda a los usuarios a comprender y aplicar estos conceptos presentados de manera efectiva.
Garantiza que los usuarios adquieran un conocimiento de cada componente del sistema, así como sus interacciones entre sí, esto ayudando a generar un conocimiento útil y funcional para otras necesidades.

Público objetivo

Esta dirigido a cualquier persona interesada en este lenguaje llamado verilog o que este curioso por este gran mundo de procesadores, memorias, MIPS, etc.
Un público especifico son estudiantes de ingeniería y/o profesionales que les gusten temas de este tipo.

Requisitos previos

Conocimiento básico en arquitectura de computadora, familiaridad con el lenguaje Verilog para facilitar su comprensión.
ModelSim para poder hacer simulaciones.

Descripción de los módulos

Módulo UnidadControl

Es el que se encarga de generar las señales de control necesarias para un correcto funcionamiento. Principalmente en este código está el Opcode que toma como entrada y es de 6 bits y produce varias señales de control que dicen cómo deben operar los diferentes componentes del datapath.

El módulo UnidadControl evalúa el Opcode de una instrucción para determinar qué señales de control activar. Estas señales de control dirigen el flujo de datos y las operaciones dentro del procesador. Las señales de control generadas incluyen:

- RegDst: Dice si el registro de destino viene del campo “rt” o “rd”.
- Branch: Abre la posibilidad de usar instrucciones de bifurcación (como beq).
- MemRead: Es el que activa la lectura de memoria de datos.
- MemToReg: Decide si los datos a escribir en el registro vienen de la memoria de datos o de la ALU.
- ALUOp: Determina la operación que debe realizar la ALU.
- MemToWrite: Es el que controla si se debe escribir en la memoria de datos.
- ALUSrc: Selecciona si la segunda entrada de la ALU es un registro o un valor inmediato.
- RegWrite: Habilita la escritura en un registro.
- Jump: Activa las instrucciones de salto.

Una mayor explicación de cada función del módulo es:

Ejemplo para la instrucción tipo I: “lw”, en este cado el “Opcode” es 6’b100011 y las señales de control son las siguientes:

- RegDst = 1’b0: El registro de destino se selecciona del campo “rt”.
- Branch = 1’b0: Indica que no es una instrucción de bifurcación.
- MemRead = 1’b1: Nos habilita la lectura de memoria de datos.
- MemToReg = 1’b1: Los datos que se escribirán en el registro provienen de la memoria.
- ALUOp = 2’b00: La ALU realizará una suma.
- MemToWrite = 1’b0: No se escribirá en la memoria.
- ALUSrc = 1’b1: La segunda entrada de la ALU es un valor inmediato.
- RegWrite = 1’b1: Nos habilita la escritura en un registro.
- Jump = 1’b0: Indica que no es una instrucción de salto.

El módulo UnidadControl desempeña un papel vital en la generación de señales de control basadas en el Opcode de las instrucciones, asegurando que el procesador ejecute correctamente las operaciones correspondientes a cada instrucción.

Módulo: SE (Sign-Extend)

Este módulo se encarga de hacer la extensión de signo de un valor de 16 bits a 32 bits. Nos sirve mucho para las instrucciones tipo I ya que muchas de estas instrucciones utilizan valores inmediatos de 16 bits y necesitan ser extendidos a 32 bits para poder ser utilizados en muchas cosas.

Toma un valor de 16 bits como entrada y como salida una de 32 bits.
La extensión del signo se realiza tomando el bit más significativo del valor de 16 bits (“unextend[15]”) y se replica 16 veces para llevar los 16 bits más significativos de la salida de 32 bits y los 16 bits originales (“unextend”) se colocan en los 16 bits menos significativos de la salida.

Esto se realiza con la instrucción de concatenación:

{{16{unextend[15]}}, unextend}.

Un ejemplo de uso aquí es que algunas instrucciones tipo I, como “lw” (Load Word), “sw” (Store Word, y “addi” utilizan valores inmediatos que deben ser extendidos de 16 a 32 bits antes de ser utilizador.

Al extender correctamente el signo de los valores de 16 a 32 bits, el módulo nos garantiza que las operaciones se realicen con precisión y los valores se mantengan.

Módulo SL32B (Shift Left 2 Bits)

Este módulo es el encargado de realizar una operación de desplazamiento a la izquierda de 2 bits en un valor de 32 bits; Nos sirve mucho en algunas instrucciones de salto.

Toma un valor de 32 bits como entrada y una salida de 32 bits que es el resultado de desplazar el valor de entrada 2 bits a la izquierda.

Se realiza gracias a: “out = in << 2;”, además está dentro de un always ya que cada vez que la entrada cambie la salida se actualizará.

Al hacer el desplazamiento nos asegura que las direcciones de salto sean correctas y eficientes.

Módulo RAM

Este módulo representa una memoria RAM que tiene la capacidad de leer datos de 32 bits, es muy importante para muchas operaciones, por ejemplo, la de lectura y escritura.

Tiene varios componentes, los de entrada tienen como funcionamiento:

- Dato: Es el dato que se quiere escribir en la memoria.
- Dir: La dirección en la memoria donde se quiere leer o escribir el dato.
- Sel: Es la señal que habilita la escritura en la memoria.
- Sel2: Señal que habilita la escritura en la memoria.

Y de salida:

- Salida: El dato leído de la memoria en la dirección que se especificó.

La memoria interna:

- Dato1: Es un arreglo de 32 palabras de 32 bits cada una.

Las operaciones de lectura y escritura indican que si “Sel” esta activado, el dato en “Dato” se escribe en la dirección ya especificada y si “Sel2” esta activado, el dato en la dirección “Dir” se lee y se iguala a la salida.

Es también importante para las instrucciones tipo I ya que como por ejemplo “lw” (load Word) carga un dato desde una dirección de memoria especifica en un registro.

Módulo PC

Aquí se implementa el program counter, este mantiene la dirección de la próxima instrucción que se va a realizar y se actualiza en cada ciclo de reloj.

Las entradas son:

- PCNext: Es la próxima dirección de instrucción a ser cargada en el PC.
- Clk: Señal de reloj

En este caso solo hay una salida y es:

- PCResult: Es la dirección actual de la instrucción que se está llevando a cabo.

En la inicialización PCResult se inicializa en 0 ya que esto nos asegura que comience la ejecución desde la primera dirección de memoria.

En el bloque always, PCResult se actualiza con el valor de PCNext en cada flanco positivo de la señal del reloj, esto para que se sincronice la actualización del PC con el ciclo de reloj.

El PC asegura que se puedan ejecutar instrucciones de manera secuencial, es esencial para la secuenciación correcta de instrucciones y el flujo de control en el datapath.

Módulo Mux2a15B

Es un multiplexor de 2 a 1 para señales de 5 bits, selecciona entre dos entradas de 5 bits (“D1” y “D2”) dependiendo del valor del selector (SD) y esa señal es dirigida a la salida (output reg Salida).

En la operación del multiplexor hay dos casos:

- 1’b1: La salida se asigna el valor de D1
- 1’b0: La salida se asigna el valor de D2
  El valor default que es en caso de que SD tenga un valor invalido se asigna el valor de D1.

Es un componente fundamental en el datapath ya que permite una selección eficiente entre dos entradas de 5 bits, es esencial para la correcta ejecución de instrucciones así dándonos por seguro que los registros de destino sean seleccionados apropiadamente.

Módulo Mux2a1

Es un multiplexor de 2 a 1 pero ahora para señales de 32 bits, tiene la misma función que el módulo Mux2a15B, toma dos entradas de 32 bits y selecciona una para enviarla a la salida según el selector.

Módulo MemIns

Es el encargado de proporcionar la instrucción de 32 bits en la memoria en la dirección que se especifica por “Adress”. Implementa una memoria de instrucciones que carga y almacena instrucciones desde un archivo.

Contiene una entrada:

- Address: Es la dirección de memoria de la instrucción que se desea recuperar.

Contiene una salida:

- Instrucción: Instrucción completa de 32 bits que se lee de la memoria.

Memoria interna:

Mem: Es una memoria de 255 posiciones donde cada posición almacena 8 bits.

Primeramente, en el bloque initial la memoria mem recibe instrucciones desde un archivo exterior llamado “instrucciones.txt” gracias a $readmemb.
En la lectura de instrucciones, la instrucción de 32 bits se obtiene leyendo cuatro posiciones en la cual se asigna un valor diferente en cada uno, gracias a esto se asegura que la instrucción se recupera correctamente desde la memoria. Una vez que es recuperada es posible ser decodificada y ejecutada, esto incluye instrucciones, como por ejemplo las de tipo I.

La implementación de este módulo nos da como resultado un correcto funcionamiento y control de ejecución en el datapath.

Módulo Registros

Implementa un banco de registros que tiene la capacidad de leer y escribir datos en registros de 32 bits, también proporciona almacenamiento para datos que las instrucciones necesiten procesar.

Tiene 5 entradas:

- RA1: La dirección del primer registro a leer
- RA2: Dirección del segundo registro a leer.
- Dir: Es la dirección del registro donde se escribirá.
- en: Es la señal que habilita la escritura en el registro.
- Dw: Es el dato que se va a escribir en el registro especificado.

Tiene 2 salidas:

- S1: Es dato que se leyó del primer registro anteriormente especificado por RA1
- S2: Dato que se leyó del segundo registro especificado por RA2.

Se inicializa el banco de registros desde un archivo exterior llamado “BR.txt” gracias al readmemb.

Escritura: Si en es == 1, el dato Dw se escribe en el registro especificado por Dir.

Lectura: Los valores de los registros especificados por RA1 y RA2 se leen y se igualan a S2 y S2.

Gracias al bloque always se asegura que las operaciones de lectura y escritura no dependan de un reloj y se puedan realizar de manera combinacional.

Tomando en cuenta las instrucciones de tipo I los registros tienen un papel muy importante ya que algunas instrucciones como “lw” (load Word) y “sw” (store Word) acceden a registros para obtener operandos y almacenar resultados.

Es fundamental para muchas operaciones y de consiguiente, eficientes y correctas en el datapath ya que permite la lectura y escritura de datos en un banco de registros que es esencial para que instrucciones se realicen.

Módulo AluControl

Aquí se incluye un controlador para la ALU el cual determina la operación a realizar según la función y operación de la ALU recibidas como entradas.

Tiene dos entradas:

- Func: Es la función de la operación de ALU, determina que operación se realizara.
- AluOP: El tipo de operación de ALU que era, puede ser tipo R, J o I.

Tiene una salida:

- AluFunc = La señal de control que especifica la operación que la ALU tiene que realizar.

La determinación de la operación de la ALU se define según 3 casos:

- AluOP = 2’b10: Serán para operaciones tipo I.
- AluOP = 2’b00: Serán operaciones tipo R.
- AluOP = 2’b01: Serán operaciones tipo J.

Este módulo tiene una relevancia increíble ya que, sin este, no se podría realizar la operación correcta y así se asegura que las instrucciones se realicen correctamente.

Módulo ALU

Se implementa una Unidad Aritmético-Lógica que es capaz de realizar operaciones, ya sean aritméticas y lógicas según dos operandos de 32 bits, este módulo es el que realiza las operaciones especificadas por las instrucciones.

Realiza la operación sobre dos operandos que son los OP1 y OP2 según la operación especificada por el selector.

Contiene 3 entradas:

OP1: Es el primer operando.

OP2: Segundo operando.

Sel: Selector que determina la operación a realizar.

Contiene 2 salidas:

Resultado: Es el resultado de la operación hecha.

ZF: Es una bandera cero que se establece si el resultado es cero.

Según varias condiciones se realiza la operación

- Operación AND: Sel = 00.
- Operación OR: Sel == 1.
- Operación SUMA: Sel == 2.
- Operación Resta: Sel == 6.
- Operación SET ON LESS THAN: Sel == 7.
- Operación NOR: Sel == 12.

Es crucial para todo tipo de instrucciones, por ejemplo, las tipo I, involucran operaciones aritméticas y lógicas, por ejemplo la instrucción “addi”, necesita realizar operaciones aritméticas y la ALU ejecuta estas operaciones según la especificada.

Módulo Adder

Este módulo implementa un sumador de números de 32 bits, es capaz de realizar operaciones aritméticas como la suma que son comunes en varios tipos de instrucciones.

Realiza la suma de dos números de 32 bits: “E1” y “E2” y nos da el resultado “Sal”. Contiene 2 entradas:

- E1: El primer número que se sumará.
- E2: El segundo número que se sumara

Contiene 1 salida:

- Sal: Es el resultado de la suma de E1 y E2.

La operación del sumador se realiza en un bloque always en donde se realiza la suma de E1 y E2 y el resultado se le asigna a Sal. Se realiza de forma continua cada vez que cambia cualquier entrada.

Nos proporciona una funcionalidad al realizar la suma de dos números de 32 bits, puede funcionar en el ejemplo de la instrucción “addi”. Su implementación nos ayuda mucho para el correcto funcionamiento del datapath y la ejecución exitosa de instrucciones que involucran operaciones aritméticas.

Módulo Add4

Este módulo implementa un sumador que agrega 4 al valor de entrada de 32 bits, pero, ¿esto de que podría servir? Es útil en para realizar operaciones aritméticas como la adición de un valor constante, que es común en algunos tipos de instrucciones.

Suma 4 al valor de entrada de 32 bits “Entrada” y proporciona el resultado “Sal”.

Contiene 1 entrada:

- Entrada: Es el valor de entrada al que se le sumara 4.

Contiene 1 salida:

- Sal: Es el resultado de la suma de Entrada y 4

En el bloque always se realiza la suma y se le asigna el resultado a “Sal”, se realiza de forma continua cada vez que cambia “Entrada”.

Nos puede ayudar en una variedad de contextos dentro, incluidas algunas operaciones de instrucciones I, es eficiente y gracias a eso contribuye al funcionamiento general del datapath.

Módulo Main

Este es el módulo principal que conecta varios de los módulos anteriores para formar un procesador.
Las funcionalidades principales son gracias a todos los módulos anteriores ya que se fueron instanciando uno por uno.

- Clk: Es el reloj que controla el flujo y operaciones.
- Add4(A4): Suma 4 al valor de entrada, nos sirve para calcular la siguiente dirección de instrucción.
- Adder(ADD): Suma la dirección de la próxima instrucción con un desplazamiento para así poder ver/calcular la próxima dirección de instrucción o para acceder a datos en memoria.
- ALU(A): Es el que realiza operaciones aritméticas y lógicas dependiendo la funcion de la instrucción y el tipo de operación.
- AluControl(AC): Controla la función de la ALU en función del tipo de instrucción.
- Registros(BR): Es para poder almacenar datos.
- MeIns(MI): Es la memoria de instrucciones en donde se almacenan las instrucciones del programa.
- Mux1, Mux2a15B, Mux3, Mux4, Mux5: Son multiplexores que seleccionan según el controlador de la unidad de control datos o direcciones de memoria.
- PC(ProgramCounter): Es el que mantiene la dirección de la próxima instrucción que se realizara.
- RAM(R1): Memoria de datos en donde se almacenan los datos del programa.
- ShiftLeft(Shifleft), ShiftLeft2(ShiftLeft2): Estos realizan desplazamientos a la izquierda de datos.
- SE(SignExtend): Extiende el campo inmediato de una instrucción para su uso en operaciones aritméticas.
- UnidadControl(UC): Controla el comportamiento en función del código de operación de la instrucción actual.

Este modulo principal organiza y conecta muchos componentes para la posibilidad de poder funcionar en conjunto, siguiendo señales de control y demás proporcionadas por la unidad de control y el flujo de datos a través del datapath.

Guía de uso

1.- Instalación del software: Es necesario ModelSim así que asegurarse de tenerlo instalado es importante.

2.- Configuración del proyecto: Crear un nuevo proyecto en ModelSim e importar los archivos del código. Ve a “Project” -> “Add to Project” -> “Existing File...”.

3.- Compilación: Compilar el proyecto para verificar si este no contiene algún error y lleva integridad en este mismo.

4.- Cargar archivos de memoria: Se crean documentos, en este caso “instrucciones.txt” que contiene las instrucciones a cargar en el simulador y otro archivo “BR.txt” que contiene la configuración inicial del banco de registros.

5.- Simulación: Cargar los archivos de memoria y ejecutar la simulación en la parte de library.

6.- Visualización: Para poder ver la simulación visualizada necesitaras abrir la parte de y objects, copia todos los objects a la parte de wave y en la parte superior ejecutar “Run” tiene como icono un papel.

Resolución de problemas

Errores comunes

Error de sintaxis:

Son comunes al cargar algún archivo o al escribir código, para esto necesitas verificar la sintaxis de tus archivos de memoria y en la parte de edición.
Presta atención a la estructura y el formato de los archivos de memoria.

Fallos en la simulación:

Verifica las conexiones estén establecidas correctamente.
Revisa las señales de control y asegurarse que estén configuradas apropiadamente.

NOTA: Si es necesario ayuda externa, no dudes en contactar a cualquier integrante de la creación del proyecto.

Conclusión

Esta guía de usuario proporciona una visión completa sobre el diseño, simulación e implementación de este código.
Desde la descripción de los módulos individuales hasta la integración en un sistema completo, los usuarios encontraran instrucciones paso a paso, claras y concisas para poder aplicar y utilizar este código.

Al abordar temas de arquitectura de computadoras que mucha gente no conoce, la resolución de problemas comunes y el contacto para la ayuda siempre es útil.
Se presenta como una herramienta muy importante para estudiantes, ingenieros o gente curiosa que desea explorar y trabajar en este mundo de arquitecturas de computadoras.

Si estas aprendiendo o buscando aprender más, esta guía te proporciona los recursos necesarios para poder visualizar más y más proyectos que tengas en mente. Con un enfoque claro en la comprensión de los conceptos fundamentales, te equipa con herramientas necesarias para más trabajos como estos.

Referencias

darkdans. (13 de febrero de 2017). El camino de datos (DataPath). darkdans. Recuperado de http://darkdans.blogspot.com/2017/02/el-camino-de-datos-datapath.html

Díaz Pérez, A. (s.f.). Single Datapath. Recuperado de https://www.tamps.cinvestav.mx/~adiaz/ArqComp/08-SingleDatapath.pdf

Universidad Complutense de Madrid. (s.f.). Arquitectura MIPS: Formato de la instrucción máquina. Recuperado de http://www.fdi.ucm.es/profesor/jjruz/ec-is/temas/Tema%205%20-%20Repaso%20ruta%20de%20datos.pdf