UNIVERSIDAD DE GUADALAJARA

CUCEI

Proyecto Final REPORTE

Integrantes

Alcaraz Suarez Gabriel Isaí

Guzman Marquez Alan Sabastian

Quintero González Diego Gerardo

Sanchez Agredano Kevin Manuel

19 de mayo del 2024

Arquitectura de Computadoras

D03, 2024A

Prof. Lopez Arce Delgado Jorge Ernesto

Introducción

En este proyecto final se aborda la implementación en verilog de un dataPath que será capaz de ejecutar instrucciones básicas , incluyendo Brench If Equal (BEQ). Se continúa trabajando con módulos ya usados en prácticas anteriores como memoria de instrucciones, ALU, unidad de control, extensores de signo, multiplexores y memoria de datos. Gracias a esta práctica se logra realizar operaciones y control de flujo, siguiendo el proceso que se muestra en el diagrama que se proporcionará en el apartado de desarrollo final (Fase 3).

Objetivo

Diseñar un “datapath” con arquitectura tipo MIPS de 32 bits capaz de ejecutar las 28 instrucciones previamente definidas por ustedes, complete la tabla 1 y tabla 2 con la sintaxis.

Instruccion	Tipo	sintaxis
Add	R	Add $rd, $rs, $rt
Sub	R	
~~Mul~~	~~R~~	
~~Div~~	~~R~~	
Or	R	
And	R	
Addi	I	
~~Subi~~	~~I~~	
Ori	I	
Andi	I	

Tabla 2

Instruccion	Tipo	sintaxis
Lw	I	
Sw	I	
slt	R	
Slti	I	
beq	I	
~~bne~~	~~I~~	
j	J	
nop	R	
~~bgtz~~	~~I~~	

Tabla 1

Debe de elegir un algoritmo previamente aprobado por su profesor, y que sea posible implementar con el set reducido de instrucciones de la tabla 1 y 2. Este programa previamente definido en ensamblador debe ser codificado a código binario y precargado en la memoria de instrucciones para que el datapath lo ejecute, recuerde definir cada uno de los aspectos de dicho programa, secciones de los registros en el banco de registros para base pointers, resultados, resultado de comparaciones, etc. Así como los datos precargados en su memoria de datos.

Fase 1

Objetivo particular: Diseñar los módulos necesarios para crear el datapath que permita ejecutar las instrucciones de tipo R de la tabla 1 y tabla 2. Vea el archivo Fase1.PDF para más detalles.

Introducción

En el desarrollo de sistemas digitales, la implementación de procesadores es fundamental para el funcionamiento de dispositivos electrónicos. Comprender los elementos básicos de un procesador, como la Memoria, la Unidad Aritmético-Lógica (ALU), y el Banco de Registros, es esencial para diseñar y construir sistemas computacionales eficientes y funcionales.

Una vez entendido el funcionamiento de estos elementos básicos, es posible avanzar en la implementación de procesadores más complejos. En esta fase del proyecto, se aborda la construcción de un "DataPath", que forma parte de una versión simplificada del procesador MIPS. El objetivo principal de este DataPath es implementar y decodificar instrucciones básicas tipo R. Las instrucciones tipo R se caracterizan por tener una estructura de 32 bits, dividida en seis elementos: OpCode, RS, RT, RD, Shamt y Function. El OpCode y el Function son campos de 6 bits cada uno. Al ingresar a los módulos correspondientes, se van separando y procesando para ejecutar la instrucción adecuada.

En resumen, esta práctica tiene como objetivo implementar un DataPath capaz de ejecutar instrucciones tipo R en un procesador, integrando módulos existentes con nuevos módulos de control y selección de datos. El éxito en esta implementación permitirá comprender en detalle el funcionamiento interno de un procesador y sentar las bases para diseños más complejos en el futuro.

Objetivo

Objetivo General

El objetivo principal es crear un DataPath capaz de ejecutar instrucciones de tipo R, para lograrlo será necesario integrar los módulos ya usados en actividades anteriores (ALU, BR y Memoria).

Con nuevos módulos que gestionarán el flujo de los bits en el sistema, estos módulos pueden observarse en la Figura 2.

Objetivos Particulares

- Desarrollar un decodificador en Python que reciba una expresión de tipo R en lenguaje ensamblador y la convierta en lenguaje binario, incluyendo el código de operación (Opcode), los registros

de origen (rs), destino (rd) y destino (rt), y el desplazamiento (shamt) si es aplicable.

- Desarrollar una presentación, en un archivo README.md, en el repositorio remoto en el cual se estarán subiendo los archivos y carpetas que se estarán trabajando a lo largo del proyecto.
- Desarrollar un módulo, el cual tome una entrada de 32 bits, la cual será una expresión en ensamblador traducida a binario, este módulo tendrá varias salidas las cuales serán cada una de las partes que tiene una expresión de tipo R (mencionadas en el punto anterior).
- Desarrollar el módulo de Unidad de Control (UC) con una entrada de 6 bits para recibir el Código de Operación (Opcode) y generar tres salidas de 1 bit (MemToReg, RegWrite y MemToWrite) y una salida de 3 bits (ALUOp) para controlar los diferentes elementos del sistema de acuerdo con el OpCode recibido.
- Implementar el módulo de ALU-Control, el cual recibirá los bits 0:5 de la instrucción y una señal de 3 bits de la UC. Este módulo generará una salida de 3 bits para indicar a la ALU la operación que debe realizar, basándose en las señales enviadas por la UC.
- Diseñar e implementar un Multiplexor 2:1 con dos entradas de 32 bits cada una y una entrada de 1 bit para seleccionar qué entrada se pasa a la salida. La salida del multiplexor será de 32 bits.
- Desarrollar un testbench mínimo para los módulos nuevos (UC, ALU-Control y Multiplexor) para verificar su correcto funcionamiento individual.
- Crear un testbench para el módulo DataPath Tipo-R (DPTR), instanciando todos los módulos desarrollados y probándolos con 10 instrucciones, dos de cada una de las instrucciones mencionadas en la introducción (ALU, SUB, OR, AND, SLT).
- Preparar una tabla con las instrucciones en formato ensamblador y su equivalente en código máquina. Los datos de esta tabla se utilizarán como entrada para el testbench del DPTR.
- Realizar pruebas del módulo DPTR para garantizar su correcto funcionamiento en la ejecución de instrucciones de tipo R, incluyendo diferentes combinaciones de instrucciones y condiciones de entrada.

Desarrollo

Descripción de Python

Este código en Python está diseñado para decodificar instrucciones de ensamblador MIPS a su equivalente binario. Comienza con un diccionario llamado codigos_TipoR que mapea operaciones de ensamblador como “add” y “sub” a sus respectivos códigos de operación (op) y función (funct) en binario.

La función decodificar_instrucciones toma dos argumentos: un archivo de entrada con instrucciones en ensamblador y un archivo de salida donde se escribirán las instrucciones decodificadas. Dentro de esta función, se abre el archivo de entrada y se lee línea por línea. Cada línea se pasa a la función decodificar_linea, que divide la instrucción en 4 partes (op, rs, rd, rt), busca los códigos binarios correspondientes en el diccionario y devuelve la instrucción en formato binario.

Finalmente, esta instrucción binaria se escribe en el archivo de salida. La estructura del código facilita poder agregar mas funciones y diccionarios para distintos tipos de instrucciones en las siguientes semanas.

Decodificación de Instrucciones en Python

Para decodificar instrucciones de ensamblador MIPS a su equivalente binario. Este código decodifica instrucciones de ensamblador MIPS en formato op rs rt rd a su equivalente binario. Cada instrucción se busca en el diccionario codigos_TipoR para obtener sus códigos de operación (op) y función (funct) en binario. Luego, se escribe la instrucción binaria en un archivo de salida.

La estructura del código permite agregar más funciones y diccionarios para distintos tipos de instrucciones en el futuro, lo que facilita su extensión y mantenimiento.

Descripción de las instrucciones tipo R

Las instrucciones tipo R son un tipo de instrucción en lenguaje de máquina utilizadas en arquitecturas de computadoras, especialmente en procesadores tipo MIPS. Estas instrucciones involucran registros y operaciones aritméticas o lógicas. En una instrucción tipo R, los campos de la instrucción están organizados de la siguiente manera:

- OpCode: 6 bits

- RS: 5 bits (registro fuente 1)

- RT: 5 bits (registro fuente 2)

- RD: 5 bits (registro destino)

- Shamt: 5 bits (desplazamiento)

- Function: 6 bits (código de función específico de la operación)

La instrucción SLT (Set on Less Than) se utiliza para comparar dos valores y establecer un registro con el valor 1 si el primer valor es menor que el segundo, o con el valor 0 en caso contrario. Su formato en lenguaje ensamblador es:

SLT $rd, $rs, $rt

Donde $rd es el registro destino, $rs es el registro fuente 1 y $rt es el registro fuente 2. La operación SLT es útil en la implementación de estructuras de control de flujo condicionales.

Investigación sobre la operación ternaria

La operación ternaria es una operación condicional que se encuentra en muchos lenguajes de programación. También se conoce como operador ternario. Su forma general es:

condición ? expresión1 : expresión2

Donde la condición es evaluada primero. Si es verdadera, se evalúa expresión1 y se devuelve su valor. Si es falsa, se evalúa expresión2 y se devuelve su valor. La operación ternaria es una forma concisa de escribir una estructura condicional if-else en una sola línea.

Investigación sobre el proceso de compilación

El proceso de compilación es el proceso mediante el cual un programa escrito en un lenguaje de alto nivel se traduce a un lenguaje de máquina que la computadora puede entender y ejecutar. El proceso de compilación consta de varios pasos:

1. Análisis léxico: El código fuente se divide en tokens (palabras clave, identificadores, literales, etc.).

2. Análisis sintáctico: Se verifica la estructura gramatical del código fuente para garantizar que cumpla con las reglas del lenguaje.

3. Análisis semántico: Se verifica el significado del código para detectar errores semánticos.

4. Generación de código intermedio: Se genera un código intermedio que representa el programa de manera más abstracta.

5. Optimización de código: Se aplican diversas técnicas para mejorar el código intermedio y hacerlo más eficiente.

6. Generación de código objeto: Se genera el código objeto específico de la arquitectura de la computadora.

7. Enlazado: Se combinan los diferentes módulos de código objeto y bibliotecas para formar un programa ejecutable.

El proceso de compilación es fundamental para la programación en lenguajes de alto nivel, ya que permite que los programas escritos por los programadores se ejecuten en la computadora de manera eficiente.

Unidad de Control (UC)

La Unidad de Control recibe una entrada de 6 bits correspondiente al Código de Operación (OpCode) y tiene tres salidas de 1 bit: MemToReg, RegWrite y MemToWrite. Además, tiene una salida de 3 bits, ALUOp, que se conecta a la ALU-Control. Esta unidad define las señales que se enviarán a los diferentes elementos del sistema dependiendo del OpCode que reciba.

ALU-Control

Esta unidad recibe una entrada de 6 bits, que corresponde a los bits 0:5 de la instrucción, y una entrada de 3 bits que llega de la UC. Tiene una salida de 3 bits que se envía a la ALU para indicar la operación que debe realizar. La ALU-Control escucha a la UC y, dependiendo del código que reciba, hace caso o no de los bits 5:0 de la instrucción.

Mux 2:1

Este multiplexor tiene dos entradas de 32 bits cada una y una entrada de 1 bit para seleccionar qué entrada se pasa a la salida. La salida es de 32 bits, donde se canaliza el dato seleccionado.

TestBench

Para el TestBench del DataPath Tipo-R (DPTR), se debe hacer un test para los módulos nuevos. Se debe probar el DPTR enviándole 10 instrucciones, dos de cada una de las instrucciones mencionadas en la introducción. Se debe incluir una tabla con las

instrucciones en formato ensamblador y otra con el código máquina correspondiente.

Conclusiones

Kevin Manuel Sanchez Agredano

Se ha trabajado en la implementación de un DataPath en Verilog para ejecutar instrucciones básicas tipo R en un procesador, así como en la creación de un código en Python para decodificar instrucciones de ensamblador MIPS a su equivalente binario.

Este proyecto ha sido una oportunidad para aplicar conocimientos teóricos en el diseño y la implementación de sistemas digitales, así como en el desarrollo de herramientas de software para facilitar tareas relacionadas con el procesamiento de instrucciones de ensamblador. Ha permitido también explorar la integración de hardware y software en el contexto de sistemas informáticos, lo cual es fundamental en el campo de la arquitectura de computadoras.

Diego Gerardo Quintero Gonzalez

En esta práctica, logramos diseñar e implementar un DataPath capaz de ejecutar instrucciones básicas tipo R. Se integraron los módulos existentes con nuevos módulos de control, permitiendo el funcionamiento adecuado del procesador en la ejecución de las instrucciones mencionadas.

Además se desarrolló un decodificador que tiene la finalidad de convertir instrucciones MIPS en código binario. El cuál diseñado para ser utilizado con archivos de texto que contienen instrucciones MIPS escritas en un formato específico.

Creo que como 1er fase pudimos lograr de buena forma los objetivos que se tenían planteados para poder presentar una primera etapa de calidad y a tiempo.

Gabriel Isaí Alcaraz Suarez

Al haber trabajado con Python para hacer el decodificador me he dado cuenta de que a Python le dan igual los archivos, o mejor dicho sus extensiones, podría invertir las entradas y salidas para el decodificador, y las escribiría sin problemas. Me he dado cuenta de que si es muy versátil este lenguaje, porque no solo maneja los

archivos si no también permite tener cosas gráficas con otras librerías.

Alan Sabastian Guzmán Marquez

Al hacer los archivos de verilog, me di cuenta de que el problema principal se centraba en que tenía un mal planteamiento del problema, atravesando por distintas dificultades como instancias innecesarias y falta de visión para el contenido interno de los módulos. Me apoye mucho de mis compañeros de equipo que me explicaron paso a paso cómo hacer para crear los módulos UnidadDeControl y ALU_Control, así como las instancias para el DataPath.v, que fue la parte final de verilog en la primera fase. Se sigue trabajando en correcciones de los códigos y un correcto manejo del entorno.

Fase 2

Objetivo particular: Agregar los módulos necesarios al datapath para poder ejecutar las instrucciones tipo R de la tabla 1 y tabla 2.

Introducción

En esta fase del proyecto después de haber realizado un DataPath de las instrucciones tipo R, ahora realizaremos un “single cycle DathaPath” que es una implementación de procesador en la que cada instrucción se ejecuta en un solo ciclo de reloj, es decir debemos lograr que toda una instrucción pase por todos los módulos en un solo ciclo de reloj.

Además en esta etapa del proyecto se tocaron temas como endianess, Tipos de Arquitectura Von Newman, Harvard, las instrucciones de tipo R en MIPS, la operación ternaria y el proceso de compilación.

Objetivo

•Objetivo General

El objetivo principal es agregar e instanciar los otros módulos para completar la Figura 1 y 2 y hacer que la parte de las instrucciones de tipo R y operaciones (DataPath) corra en un solo ciclo.

Con nuevos módulos como la memoria de instrucciones, PC y otros multiplexores, estos módulos pueden observarse en la Figura 1 y 2.

•Objetivos Particulares

- Agregar al decodificador la posibilidad de abrir los archivos en ensamblador y modificarlos en una ventana de texto, guardar los cambios y convertir a binario.
- Agregar la opción al decodificador para generar un archivo para inicializar el banco de registros y memoria de datos. Este archivo debe incluir direcciones del banco de registros y datos de 32 bits para operar.
- Tener una guía de usuario de la fase 1.2 del decodificador

Desarrollo

Ljubisa Bajic

Ljubisa Bajic es un investigador y profesional en el campo de la arquitectura de computadoras.

En cuanto a las contribuciones específicas de Ljubisa Bajic, aquí hay algunas patentes y trabajos relacionados con su investigación:

Dinámica de reducción de potencia controlada y circuito para un procesador gráfico: L. Bajic y J. Fry obtuvieron una patente relacionada con la reducción de potencia en procesadores gráficos.

Cálculo dinámico de peso en un sistema de estimación y gestión de energía digital: L. Bajic también tiene una patente relacionada con el cálculo de peso en sistemas de estimación y gestión de energía.

Gestión de energía entre múltiples procesadores que comparten una plataforma térmica: L. Bajic y otros colaboradores obtuvieron una patente relacionada con la gestión de energía en sistemas con múltiples procesadores.

Jim Keller

Jim Keller, un destacado diseñador de arquitecturas de CPU, ha dejado su huella en la industria tecnológica a lo largo de los años.

Arquitectura Zen de AMD:

Jim Keller fue uno de los principales diseñadores detrás de la microarquitectura Zen de AMD.

La arquitectura Zen revolucionó el mercado de procesadores al ofrecer un rendimiento competitivo frente a Intel.

Los procesadores AMD Ryzen se basan en la arquitectura Zen y han sido muy exitosos.

Microarquitectura K8:

Keller también estuvo involucrado en la creación de la microarquitectura K8, que dio vida a los procesadores Athlon 64 de AMD.

Los Athlon 64 fueron los primeros procesadores fabricados en 64 bits y tuvieron un gran impacto en la industria.

Microarquitectura K7:

Antes de su trabajo en AMD, Keller contribuyó a la microarquitectura K7, que fue la primera en superar la frecuencia de 1 GHz en procesadores.

La K7 también fue la primera arquitectura que preocupó a Intel.

Ingeniería de Silicio en Intel:

En 2018, Jim Keller se unió a Intel como vicepresidente senior de Intel Technology, Systems Architecture & Client Group (TSCG) y

gerente general de ingeniería de silicio.

Reveló que Intel tenía una estrategia para actualizar la arquitectura y la tecnología de la CPU cada dos años, además de desarrollar una nueva arquitectura CPU cada 5 años.

Raja Koduri

Raja Koduri es un ingeniero informático indio y un ejecutivo especializado en hardware de gráficos por computadora. A lo largo de su carrera, ha desempeñado roles clave en empresas líderes de la industria. Aquí tienes algunos aspectos destacados de su trayectoria:

Carrera temprana:

Nacido el 31 de agosto de 1968 en Kovvur, distrito de West Godavari, Andhra Pradesh, India, Raja Koduri proviene de una familia telugu.

Su primo es el conocido director de cine S. S. Rajamouli, y otros miembros de su familia también trabajan en la industria cinematográfica como escritores, compositores de música y cantantes.

Obtuvo su licenciatura en electrónica y comunicaciones de la Universidad de Andhra y una Maestría en Tecnología de la IIT Kharagpur1.

Contribuciones en la industria de gráficos:

En 1996, Koduri se unió a S3 Graphics y luego se convirtió en director de desarrollo de tecnología avanzada en ATI Technologies.

Después de la adquisición de ATI por parte de Advanced Micro Devices (AMD) en 2006, se desempeñó como Director de Tecnología para gráficos en AMD hasta 2009.

Durante su tiempo en S3 y ATI, contribuyó significativamente al diseño de varias generaciones de arquitecturas de GPU, desde DirectX Ver 3 hasta Ver 11.

Luego, trabajó en Apple Inc., donde se centró en hardware de gráficos, lo que permitió a Apple adoptar pantallas Retina de alta resolución en sus computadoras Mac1.

AMD y Radeon Technologies Group (RTG):

En 2013, Raja Koduri regresó a AMD como Vicepresidente de Visual Computing, supervisando tanto el hardware como el software de GPU.

En 2015, AMD reorganizó su división de gráficos y ascendió a Koduri al nivel ejecutivo, nombrándolo Vicepresidente Senior y Arquitecto Jefe del recién formado Radeon Technologies Group.

Bajo su liderazgo, se transformó la arquitectura de las GPU con las generaciones Polaris, Vega y Navi, que se utilizaron en PC, Mac y consolas de juegos (como Xbox y PlayStation).

En 2017, Koduri tomó un año sabático para pasar tiempo con su familia, y durante su ausencia, la CEO de AMD, Lisa Su, asumió el liderazgo de RTG.

Sin embargo, en noviembre de 2017, anunció su renuncia a AMD1.

Intel y arquitectura Xe:

Posteriormente, Raja Koduri se unió a Intel como Arquitecto Jefe y Vicepresidente Ejecutivo de la división de arquitectura, gráficos y software (IAGS).

Durante su tiempo en Intel, lideró el desarrollo de la arquitectura Xe, que se destaca por su rendimiento, escalabilidad, eficiencia y compatibilidad con tecnologías de última generación.

En marzo de 2023, después de seis años en el sector de las GPU, Koduri dejó Intel para fundar una start-up de software de juegos generativos basada en inteligencia artificial.

Endianess

La endianness (o byte order) se refiere al orden en el que se almacenan los bytes en la memoria de una computadora. Es un concepto fundamental para comprender cómo los datos se representan internamente.

Big-Endian (BE):

En big-endian, el byte más significativo (o “big end”) de un valor de datos multibyte se almacena en la dirección de memoria más baja.

Imagina que tienes un número de 32 bits (4 bytes) como 0x12345678. En big-endian, se almacenaría como:

0x12 0x34 0x56 0x78

El byte 0x12 (el más significativo) se encuentra en la dirección de memoria más baja.

Little-Endian (LE):

En little-endian, el byte menos significativo (o “little end”) de un valor de datos multibyte se almacena en la dirección de memoria más baja.

Siguiendo el mismo ejemplo, en little-endian, el mismo número se almacenaría como:

0x78 0x56 0x34 0x12

El byte 0x78 (el menos significativo) está en la dirección de memoria más baja.

Ejemplo visual:

Consideremos el número decimal 1,100 (representado en binario como 1100).

Si lo almacenamos en 3 bytes, en big-endian sería:

0b00000000 0b00000100 0b01011100

Y en little-endian:

0b01011100 0b00000100 0b00000000

En resumen, big-endian y little-endian son dos formas diferentes de organizar los bytes en la memoria.

Tipos de Arquitectura Von Newman y Harvard.

En el mundo de la informática, existen dos arquitecturas fundamentales que han influido en el diseño de sistemas informáticos: la arquitectura de Von Neumann y la arquitectura de Harvard. A continuación, te explicaré las diferencias entre ambas y cómo se aplican en las computadoras modernas:

Arquitectura de Von Neumann:

La arquitectura de Von Neumann, nombrada en honor a su creador, John von Neumann, es el modelo más comúnmente utilizado en los sistemas actuales.

Características:

Utiliza una única memoria para almacenar tanto instrucciones como datos.

La CPU accede a una ubicación de memoria específica para leer la siguiente instrucción y luego ejecutarla.

Programa y datos comparten la misma memoria y bus de datos.

Ventajas:

Es simple y fácil de implementar.

Permite la modificación del programa durante la ejecución.

Uso en computadoras modernas: La arquitectura de Von Neumann es la más común y se encuentra en la mayoría de los ordenadores personales.

Arquitectura de Harvard:

La arquitectura de Harvard separa físicamente la memoria para instrucciones y la memoria para datos.

Características:

Utiliza memorias separadas para instrucciones y datos.

Permite que la CPU acceda simultáneamente a instrucciones y datos, mejorando el rendimiento potencialmente.

Utiliza buses de datos separados para instrucciones y datos.

Ventajas:

Puede ser más eficiente en términos de rendimiento debido a la separación de memorias y buses de datos.

Uso en computadoras modernas: La arquitectura de Harvard se encuentra más en dispositivos embebidos y sistemas especializados.

El MIPS

El procesador MIPS (Microprocessor without Interlocked Pipeline Stages) es una familia de microprocesadores de arquitectura RISC (Reduced Instruction Set Computer) desarrollados por MIPS Technologies. A continuación algunos datos sobre los elementos y características generales del procesador MIPS de 32 bits:

Arquitectura de 32 bits:

Las primeras arquitecturas MIPS fueron de 32 bits, con rutas de datos y registros de 32 bits de ancho.

Posteriormente, se implementaron versiones en 64 bits.

Conjunto de instrucciones:

El conjunto de instrucciones MIPS se compone de operaciones básicas que se ejecutan en un solo ciclo de reloj.

Las instrucciones son de tamaño fijo y presentan diferentes formatos, como las instrucciones de carga y almacenamiento, operaciones aritméticas y lógicas, y bifurcaciones (instrucciones tipo R, I y J).

Modos de funcionamiento:

El procesador MIPS tiene varios modos de funcionamiento:

Usuario: para ejecutar programas de usuario.

Núcleo: para operaciones privilegiadas del sistema operativo.

Supervisor: para tareas de administración y control.

Depuración: para depurar programas.

Instrucciones de carga y almacenamiento:

Las instrucciones de carga y almacenamiento permiten leer y escribir en memoria utilizando registros.

Ejemplos:

lw $t0, dir: Carga en el registro $t0 el contenido de la palabra de memoria cuya dirección es dir.

sw $t0, dir: Almacena en la memoria direccionada por dir el contenido del registro $t0.

Bifurcaciones (control de flujo):

Las bifurcaciones permiten cambiar el flujo de ejecución del programa.

Ejemplos:

beq $t0, $t1, destino: Bifurca si los registros $t0 y $t1 son iguales.

bne $t0, $t1, destino: Bifurca si los registros $t0 y $t1 no son iguales.

Convenciones de uso de registros:

En MIPS, existen convenciones para el uso de registros,

principalmente en llamadas a procedimientos.

La convención predeterminada sigue las pautas utilizadas por el compilador GCC.

Instrucciones tipo R:

Las instrucciones tipo R son un conjunto de instrucciones aritmético-lógicas utilizadas en arquitecturas de procesadores como MIPS.

Estas instrucciones operan exclusivamente en registros y siguen un formato específico.

El formato de una instrucción tipo R consta de los siguientes campos:

Función (6 bits): Indica la operación a realizar (por ejemplo, suma, resta, AND, OR, etc.).

SHAMT (5 bits): Representa el desplazamiento de datos en operaciones de rotación.

Registros (3 campos de 5 bits cada uno): Especifican las direcciones de los registros involucrados en la operación.

OpCode (6 bits): Siempre es “000000” para instrucciones tipo R.

Ejemplo: La instrucción “add” realiza una suma aritmética entre dos registros1.

Operación Ternaria:

La operación ternaria es una expresión que involucra tres operandos y un operador.

Ejemplo: condición ? valor_si_verdadero : valor_si_falso.

Si la condición es verdadera, se evalúa el primer valor; de lo contrario, se evalúa el segundo valor.

Proceso de Compilación:

El proceso de compilación transforma código fuente en lenguaje de máquina ejecutable.

Fases del compilador:

Análisis léxico: Divide el código en tokens.

Análisis sintáctico: Construye un árbol de sintaxis abstracta.

Análisis semántico: Verifica la coherencia semántica.

Generación de código intermedio: Crea una representación intermedia.

Optimización de código: Mejora el rendimiento.

Generación de código final: Produce el código ejecutable.

Definición de las instrucciones para nuestro proyecto:

Las instrucciones de nuestro programa serán en Big endian, pues escribir y leer de izquierda a derecha es lo más conveniente para como tenemos creadas las memorias y bancos de registro, y no tener que estar dando vuelta los bits.

Y en cuanto las instrucciones en ensamblador, pretendemos usar las abreviaciones correspondientes a la operación que se va a realizar, y el número del directorio del banco de registros, todo separado por comas y cada instrucción en una línea diferente.

Conclusiones

Kevin Manuel Sanchez Agredano

La funcionalidad agregada al proyecto permite una gestión completa de archivos, facilitando su visualización, modificación y conversión a binario. La capacidad de generar un archivo binario para cargar la memoria de instrucciones con un ancho de palabra específico (8 bits) y opciones para inicializar el banco de registros y la memoria de datos añade un nivel de control y flexibilidad adicional al sistema. Estas características son fundamentales para el desarrollo y la depuración de programas en lenguaje ensamblador, facilitando la creación y prueba de software para la arquitectura objetivo. Con estas funcionalidades, el proyecto se vuelve más completo y útil para el desarrollo de programas en ensamblador.

Diego Gerardo Quintero González

En esta etapa del proyecto se han logrado avances al desarrollar y probar los módulos de varios componentes nuevos, como un multiplexor de 5 bits, un modulo PC, un sumador de 32 bits, un desplazador a la izquierda de 2 bits, una extensión de signo y una unidad de control con nuevas salidas. La creación de testbenches para cada uno de estos módulos aseguró su correcto funcionamiento. Además, se logró la integración de muy buena forma de estos nuevos módulos con los existentes de etapas anteriores, estableciendo la realizacion completa del esquema establecido para esta fase del proyecto.

Gabriel Isaí Alcaraz Suarez

En esta etapa que me tocó hacer la parte de la investigación y guía de usuario, fue fácil para mí comparado con hacer los modulos de verilog o lo que se implementó en el decodificador, y con endianess me dí cuenta que al igual que en el mundo hay países que leen y escriben de derecha a izquierda, también hay programas que se leen así por la computadora.

Alan Sebastián Guzmán Márquez

En esta fase se logró un avance en todos los aspectos que abarca el proyecto, empezando con que en la parte de decodificador se

logró crear un entorno que permite una visualización más agradable y fácil de entender. Gracias a la documentación se llegó a la conclusión que es mejor utilizar Big endian para este proyecto ya que no está pensado en que los bits estén volteados. Finalmente por la parte de Verilog se implementaron nuevos módulos para darle más optimización al proyecto

Bibliografía

https://scholar.google.com/citations?user=zuhBWukAAAAJ

https://www.studocu.com/es-mx/document/instituto-de-estudios-universitarios-ac/arquitectura-de-computadoras/procesador-mips-3modos-de-direccionamiento-y-cpu/77928875

https://informatecdigital.com/hardware/arquitectura-computadoras-introduccion-a-su-evolucion-y-diseno/

https://informatica.uv.es/seguia/FAC/Archivos/MIPS32.pdf

https://www.docsity.com/es/principales-caracteristicas-del-procesador-mips-y-de-la-arquitectura-que-implementa/7986545/

https://elchapuzasinformatico.com/2020/02/jim-keller-revela-que-creara-una-nueva-arquitectura-cpu-para-intel-cada-5-anos/

https://hardzone.es/2018/04/26/intel-ficha-jim-keller/

https://www.muycomputer.com/2023/03/22/raja-koduri-y-su-aventura-en-intel-una-mirada-a-fondo/

https://www.freecodecamp.org/news/what-is-endianness-big-endian-vs-little-endian/

https://www.baeldung.com/cs/big-endian-vs-little-endian

https://www.rapidtables.com/prog/endianess.html

https://jmjinformatico.es/arquitecturas-von-neumann-y-harvard

Fase 3

FINAL

Objetivo particular:

- Agregar los módulos necesarios para completar el datapath de MIPS de 32 bits para que sea capaz de ejecutar las instrucciones tipo I y 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. (Factorial o Fibonacci o alguno similar que use ciclo(s))
- Dicho código debe ser “decodificado” a código binario/maquina, el cual será precargado a la memoria de instrucciones para que el datapath lo ejecute.
- Crear un programa/script que realice la “decodificación” de su código ensamblador (se recomienda Python), este decodificador debe tener GUI, poder precargar un archivo de texto con código ASM y /o editar dicho archivo y guardar los cambios.

Desarrollo

Descripción de los módulos del dataPath:

alu

En este módulo se obtienen los valores de op1 y op2 para después con un case poder elegir qué operación se hará con estos dos valores: or, suma, resta, slt o xor.

ALU_Control

Se encarga de convertir los valores binarios que se presentan de 6 bits, en los 4 bits que la alu necesita como entradas del selector.

Banco de registros

Implementa un conjunto de registros de 32 bits que pueden almacenar datos temporales para la CPU. Este módulo permite leer dos registros (ra1 y ra2) y escribir en un registro (dir) cuando está habilitado (ena). Los valores iniciales de los registros se cargan desde un archivo llamado "memoria.txt". Cuando ena está activo, se escribe el dato de entrada (di) en el registro especificado por dir y se actualizan las salidas de lectura (dr1 y dr2) con los valores de los registros especificados por ra1 y ra2.

Compuerta_AND

Realiza la operación and entre a y b, con una salida llamada salida.

DataPath_TipoR

Se implementa una ruta de datos para instrucciones de tipo R en un procesador. Decodifica y ejecuta operaciones aritméticas y lógicas, gestionando registros, ALU y memoria. En el TB se presentara un número binario de 32 bits donde se desmenuzara la cifra.

Memoria

El modulo crea una memoria donde se puede leer y escribir datos en un archivo externo llamado data.txt, organizados como un arreglo. En cada ciclo de reloj se verifica si “sel” es 1 se podra escribir y si es 0 se podra leer.

PC

PC simula un registro de programa que actualiza su salida (salida_PC) en cada flanco positivo del reloj (CLK). Inicializa salida_PC a 0 y, en cada ciclo de reloj, asigna el valor de entrada_PC a salida_PC si entrada_PC no es 0; de lo contrario, mantiene salida_PC en 0.

Sumador

En el modulo sumador se declara entrada 1, entrada 2 y salida. Se suma entrada 1 y entrada 2 y el resultado se guarda en salida.

Sign Extended

Sign_Extend extiende un número de 16 bits (a) a un número de 32 bits (salida). Si el bit más significativo de a (a[15]) es 1, llena los 16 bits superiores de salida con 1s; si es 0, llena los 16 bits superiores con 0s, manteniendo los 16 bits inferiores iguales a a.

Multiplexor de 5 bit 2 a 1

Multiplexor_5_bits_2_a_1 es un multiplexor de 5 bits que selecciona entre dos entradas (a y b) basándose en el valor del selector. Si selector es 0, la salida (salida) es igual a a; si selector es 1, la salida es igual a b.

Memoria de instrucciones

Carga una memoria de instrucciones de 256 bytes desde el archivo "memoria.txt". Luego, asigna manualmente valores binarios a las primeras posiciones de la memoria. En cada ciclo, la salida (salidaMemoriaDeInstrucciones) se compone de cuatro bytes de la memoria, comenzando en la dirección especificada por dir.

Multiplexor

Es un multiplexor de 32 bits que selecciona entre dos entradas (a y b) basándose en el valor del selector. Si selector es 0, la salida (salida) es igual a a; si selector es 1, la salida es igual a b.

Shift left

Shift_left toma una entrada de 26 bits (in) y produce una salida de 32 bits (out). Realiza un desplazamiento a la izquierda de 2 posiciones en la entrada y asigna el resultado a los 28 bits inferiores de la salida. Los 4 bits superiores de la salida quedan en un estado indefinido ya que no se asignan explícitamente.

Unidad de Control

UnidadDeControl genera señales de control para instrucciones MIPS según el código de operación (op). Configura las señales (RegDst, Branch, MemToRead, MemToReg, MemToWrite, ALUSrc, RegWrite, Jump, AluOp) para controlar el datapath del procesador. Dependiendo del valor de op, estas señales determinan cómo se procesará la instrucción.

Multiplexor de 6 bit 2 a 1

Multiplexor_6_bits_2_a_1 es un multiplexor de 6 bits que selecciona entre dos entradas (a y b) basándose en el valor del selector. Si selector es 0, la salida (salida) es igual a a; si selector es 1, la salida es igual a b.

Shift left 2

Shift_left_2 toma una entrada de 32 bits y la desplaza a la izquierda 2 posiciones, resultando en una salida de 32 bits. Este desplazamiento equivale a multiplicar el valor de entrada por 4. La salida se asigna directamente usando una operación de desplazamiento.

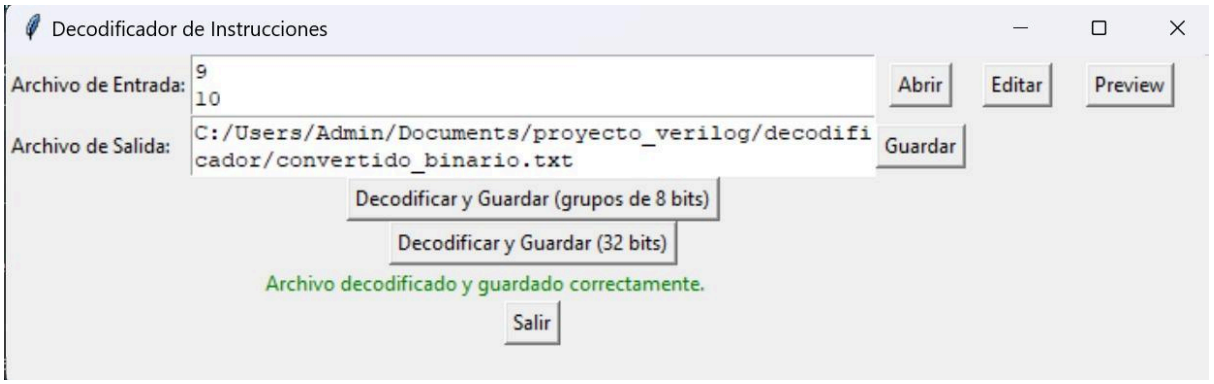
Test Bench DataPath

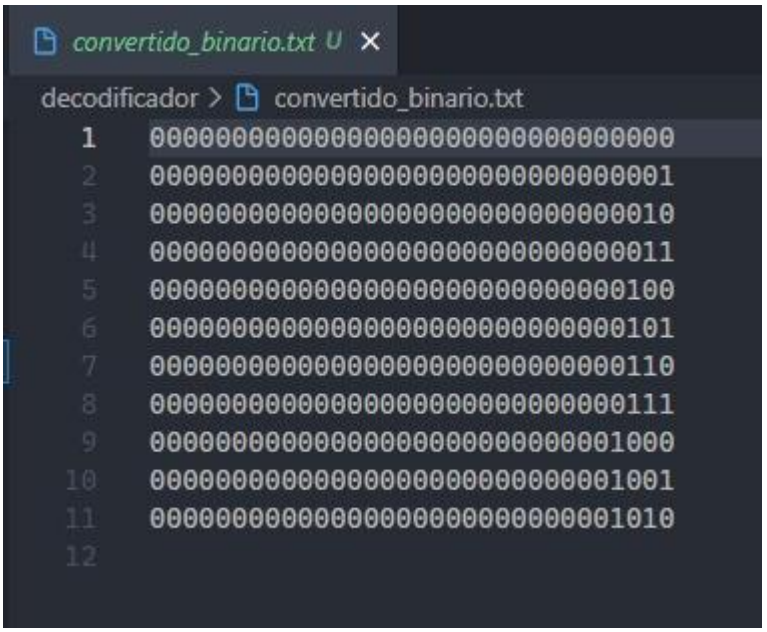
El modulo tb_DataPath verifica el módulo DataPath_TipoR alternando una señal de reloj (CLK) cada 200 unidades de tiempo. Observa la salida del datapath (tr_salida_final) para validar su funcionamiento. Inicializa el reloj a 0 y lo alterna entre 0 y 1 para simular el procesamiento del DataPath_TipoR.

EVIDENCIA


Decodificador de Instrucciones
Archivo de Entrada:
Abrir
Editar
Preview
Archivo de Salida:
Guardar
Decodificar y Guardar (grupos de 8 bits)
Decodificar y Guardar (32 bits)
Salir


decimal.asm U
decodificador > decimal.asm
0
1
2
3
4
5
6
7
8
9
10


Decodificador de Instrucciones
Archivo de Entrada:
9
10
Abrir
Editar
Preview
Archivo de Salida:
C:/Users/Admin/Documents/proyecto_verilog/decodificador/convertido_binario.txt
Guardar
Decodificar y Guardar (grupos de 8 bits)
Decodificar y Guardar (32 bits)
Archivo decodificado y guardado correctamente.
Salir


convertido_binario.txt U
decodificador > convertido_binario.txt
1 00000000000000000000000000000000
2 00000000000000000000000000000001
3 00000000000000000000000000000010
4 00000000000000000000000000000011
5 00000000000000000000000000000100
6 00000000000000000000000000000101
7 00000000000000000000000000000110
8 00000000000000000000000000000111
9 00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12

Conclusiones

Kevin Manuel Sanchez Agredano

En esta última fase todo se complicó, ya que al tener tantas cosas que hacer en un tiempo menor al acordado, todo fue pesado, la falta de tiempo para acoplarse al código que se tenia avanzado, y sobre todo la gran cantidad de implementaciones que se tenían planeadas para estas dos ultimas fases combinadas en una, fue lo que detono la bomba de cosas por realizar. Con esto se pudo aprender que cada cosa que se realice se tiene que tener una aproximación de tiempos y sobre todo una documentación en la cual basarse, para no tener complicaciones a la hora de volver a adaptarte a un nuevo rol.

Diego Gerardo Quintero González

En esta ultima fase me permitió ver como funcionaba el decodificador ya que a lo largo de todas las fases fue el apartado que menos ví en su momento, lo anterior me dió una gran idea de la manera que funcionaban cada una de las funciones y librerias que se estuvieron utilizando a lo largo del proyecto para el decodificador. Me pareció interesante y frustrante ya que muchas ocasiones me funcionaron cosas que queria implementar sin dedicarle mucho tiempo y para otras funcionalidades me tardaba y estresaba por el motivo de que era la seccion del proyecto que menos me habia enfocado y tenia menos experiencia. Al final me gustó como quedo y me dio felicidad el saber que pude completar los objetivos que se tenian planeados.

Respecto al proyecto final en general, creo que aunque se tuvieron complicaciones, se pudo llegar al objetivo final aún y a pesar de la carga y complejidad que implicaba esta ultima fase final.

Gabriel Isaí Alcaraz Suarez

Trabajar como el Manager esta semana fue complicado y estrezante para mi por los tiempos pues todas las materías buscaron cerrar antes, así que siento que descuidé mi papel, y en parte por eso no logramos terminar el proyecto

Alan Sebastián Guzmán Márquez

En esta última etapa del proyecto aparecieron nuevos problemas que tuvimos que ir resolviendo, pero en general sirvió para mejorar

nuestras habilidades como equipo, ya que aun con el tiempo en sima se logró terminar la fase final. El rol que mas me costo trabajo fue el de verilog ya que necesiten mucha de la ayuda de mis compañeros para sacar adelante la fase 1, y gracias a dios fue la fase 1 y no la 3 o la 2, que personalmente vi más difíciles. En la parte del trabajo en equipo fue interesante ver como se implementó esta idea de cambio de roles porque nos hizo salir de nuestra zona de confort y trabajar realmente en equipo, apoyando nuestros compañeros en lo que sea que necesitaran. Es una experiencia claramente nueva pero que nos deja grandes enseñanzas, esperamos que no se vuelva a repetir.

Bibliografía:

- Hernandez, A. (2010, 30 noviembre). Arquitectura MIPS. Universidad de Valladolid. Recuperado 20 de mayo de 2024, de https://www.infor.uva.es/~bastida/OC/TRABAJO2_
- MIPS.pdf Ramirez, M. E. (s. f.). Representación de instrucciones. UTM. https://www.utm.mx/~merg/AC/2009/3.3-representacion_instrucciones.html