UNIVERSIDAD DE GUADALAJARA
Red Universitaria de Jalisco

CUCEI

Proyecto Final Fase 01 - Data Path

Arquitectura de computadoras

JORGE ERNESTO LOPEZ ARCE DELGADO

D03

24A

Juan Jose Renteria Haro

Javier Alberto Galindo Parra

Benjamin Lopez Fletes

David Fernando Vazquez Mendoza

REPORTE

SEMANA 1

Javier-Decodificador: Está trabajando en la implementación del programa en Python. Está abriendo y parseando archivos en Python para este propósito.Creando la decodificación necesaria para convertir un archivo en lenguaje ensamblador a archivo binario y obtener la

carga

Juan-Administrador está encargado de la organización del proyecto y está creando un cronograma de actividades y entregas. Cada participante debe entregar las actividades en tiempo y forma según lo planificado y hablado.De igual forma presento un documento en formato PDF en el cual explico la manera correcta de cómo utilizar el repositorio de Github para subir los avances del proyecto

Benjamín-Verilog está creando los módulos de las unidades de control necesarias para la primera implementación del DPTR.

David-Documentación está generando un reporte de actividad diario para documentar el progreso del proyecto y de igual forma haciendo la investigación de conceptos y para el desarrollo fluido de la actividad ,elaborando todos los archivos que no sean códigos

LISTA DE INSTRUCCIONES PRUEBA

Instrucción	Operando 1 Operando 2 Resultado
ADD	$16 (44) $17 (11) $0 (55)
SUB	$20 (55) $15 (4) $1 (51)
SLT (Menor o mayor que)	$11 (77) $19 (25) $2 (0)
AND	$28 (101) $21 (51) $3 (33)
OR	$13 (33) $12 (89) $4 (121)
ADD	$24 (121) $29 (74) $5 (195)
SUB	$24 (121) $29 (74) $6 (47)
SLT (Menor o mayor que) AND OR	$31 (115) $26 (47) $7 (0) $25 (102) $27 (0) $8 (0) $30 (55) $23 (33) $9 (55)

GUIA DE USUARIO-DESARROLLO

Desarrollo-Python

Este proyecto consiste en un convertidor de instrucciones en lenguaje ensamblador a su representación binaria correspondiente. Utiliza una interfaz gráfica de usuario (GUI) creada con la biblioteca Tkinter de Python para proporcionar una forma fácil y conveniente de realizar la conversión.

¿Por qué es Útil el Proyecto?

El convertidor es útil para aquellos que trabajan con lenguaje ensamblador y necesitan

convertir sus instrucciones a binario para su posterior procesamiento en un procesador o simulador. Facilita este proceso al proporcionar una interfaz intuitiva que permite a los usuarios seleccionar archivos de entrada y salida de manera sencilla.

Cómo Comenzar con el Proyecto:

Los usuarios pueden comenzar descargando o clonando el código fuente del proyecto desde un repositorio en línea. Una vez descargado, pueden ejecutar el script principal y la interfaz gráfica aparecerá. Desde allí, pueden seleccionar un archivo de entrada que contenga las instrucciones en lenguaje ensamblador, especificar la ubicación y el nombre del archivo de salida donde se guardarán las instrucciones convertidas, y luego hacer clic en el botón "Convertir" para realizar la conversión.

Dónde Recibir Ayuda con el Proyecto:

Los usuarios pueden recibir ayuda con el proyecto consultando la documentación incluida en el repositorio, buscando en foros en línea relacionados con Python y Tkinter, o incluso solicitando ayuda directamente a los mantenedores y colaboradores del proyecto.

Este código es un programa de Python que utiliza la biblioteca Tkinter para crear una interfaz gráfica de usuario (GUI). La GUI permite al usuario seleccionar un archivo de entrada que contiene instrucciones en lenguaje ensamblador, y luego especificar la ubicación y el nombre del archivo de salida donde se guardarán las instrucciones convertidas a código binario. Todo lo anterior dicho es un proyecto preliminar hablado de la versión 0.3v de dicho proyecto esperando mejorar con las siguientes actualizaciones.

Se utilizaron varias funciones para mayor flexibilidad y poder entender mejor el

código. Funciones de Conversión:

decimal_a_binario(n): Convierte un número decimal a una representación binaria de 5 bits.

assemble(instrucción): Convierte una instrucción en lenguaje ensamblador a su representación binaria correspondiente. Utiliza un diccionario para mapear las operaciones a sus códigos de operación y bits de función, luego extrae los diferentes campos de la instrucción y los combina para formar la representación binaria.

Funciones de Manejo de Eventos:
convertir(): Esta función maneja el evento de hacer clic en el botón "Convertir". Lee las instrucciones desde el archivo de entrada especificado, las convierte a binario utilizando la función assemble() y luego las escribe en el archivo de salida especificado.

seleccionar_archivo_entrada(): Abre un cuadro de diálogo para que el usuario seleccione un archivo de entrada y actualiza el campo de entrada de archivo con la ruta seleccionada.

seleccionar_archivo_salida(): Abre un cuadro de diálogo para que el usuario seleccione la ubicación y el nombre del archivo de salida, y actualiza el campo de entrada de archivo con la ruta seleccionada.

Creación de la Interfaz Gráfica:

Se crea una ventana principal utilizando tk.Tk().

Se crean dos frames para organizar los elementos de entrada y salida.

Se crean etiquetas, campos de entrada y botones para que el usuario pueda interactuar con la GUI.

Se asignan funciones a los botones para manejar los eventos de clic.

Ejecución de la Aplicación:

Se inicia el bucle de eventos de Tkinter con root.mainloop(), lo que hace que la ventana sea visible y responda a las interacciones del usuario.

En resumen, este programa proporciona una interfaz gráfica fácil de usar para convertir instrucciones en lenguaje ensamblador a su representación binaria correspondiente.

Guia de usuario-Python

Convertidor de Instrucciones a Binario

Este programa en Python es una herramienta que convierte instrucciones escritas en lenguaje ensamblador a su equivalente en código binario de 32 bits. Es útil para procesadores que ejecutan instrucciones en lenguaje de máquina y requieren instrucciones binarias para la ejecución.

Requisitos:

Python 3.x instalado en el sistema.

La biblioteca tkinter para la interfaz gráfica de usuario (GUI).

Uso:

Ejecute el archivo convertidor.py.

Seleccione el archivo de entrada que contiene las instrucciones en lenguaje ensamblador (.asm).

Especifique la ubicación y el nombre del archivo de salida donde se guardarán las instrucciones convertidas en binario (.bin).

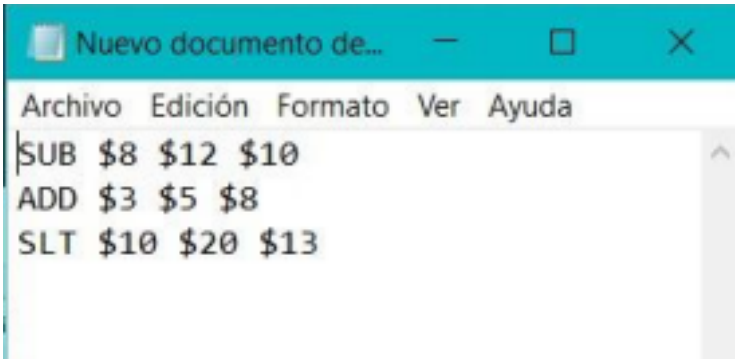
Haga clic en el botón "Convertir" para iniciar el proceso de conversión.

Formato de Archivo de Entrada

El archivo de entrada debe contener instrucciones de una línea por instrucción, siguiendo el formato estándar de lenguaje ensamblador. Por ejemplo:

ADD $t0, $t1, $t2

SUB $t3, $t4, $t5
AND $t6, $t7, $t8



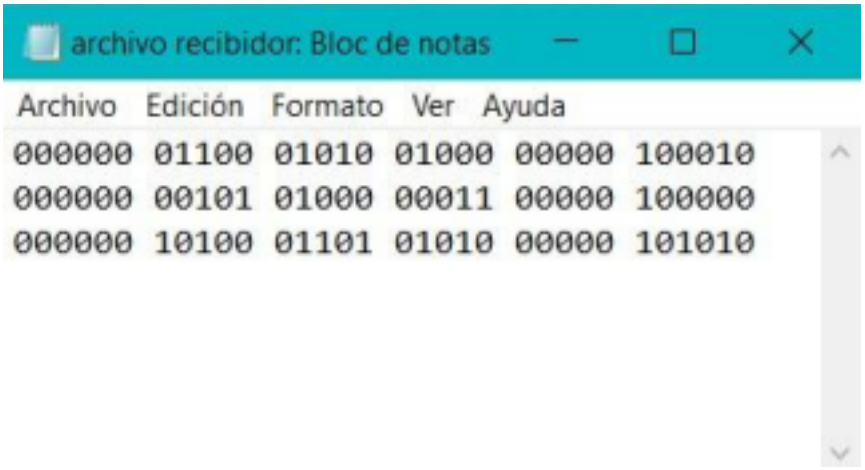
Formato de Archivo de Salida

El archivo de salida contendrá las instrucciones convertidas en su representación binaria de 32 bits, una por línea. Por ejemplo:

000000 00001 00010 01000 00000 100000

000000 00011 00100 00101 00000 100010

000000 00111 01000 01100 00000 100100



Notas

Este convertidor admite un conjunto limitado de instrucciones. Asegúrese de que las instrucciones en el archivo de entrada estén soportadas por el programa.

Desarrollo-Verilog

En este repositorio, encontrarás una implementación en Verilog de un DPTR (DATA PATH TYPE REGISTER). El DPTR realiza operaciones aritméticas y lógicas, manipula registros y
accede a una memoria RAM. Sus componentes principales son la ALU (Arithmetic Logic

Unit), el Controlador de ALU (AluControl) y los Registros.

La ALU lleva a cabo operaciones como AND, OR, suma, resta, entre otras, y emite un resultado. El Controlador de ALU determina las operaciones de la ALU según códigos específicos. Los Registros gestionan un banco de registros de 32 bits para lecturas y escrituras.

Estos módulos se integran para formar la DPTR, una unidad de procesamiento completa con la capacidad de realizar diversas operaciones de procesamiento de datos. Además de los componentes principales, también se incluyen otros módulos como un multiplexor, una memoria RAM y un controlador de unidad.

Para probar el funcionamiento de la DPTR, se proporciona un banco de pruebas (DPTR_TB) que simula el procesamiento de datos cargando datos desde archivos de texto y ejecutando operaciones simuladas

Guia de usuario-Verilog

1. DPTR

DPTR (DATA PATH TYPE REGISTER) es como una calculadora gigante que puede hacer muchas cosas. Tiene partes importantes que trabajan juntas, como una ALU (que hace cálculos), unos registros (que guardan datos) y un controlador (que dice qué hacer).

2. ¿Qué puede hacer la DPTR?

Operaciones Matemáticas: Puede sumar, restar, multiplicar y dividir números grandes.

Operaciones Lógicas: También puede comparar números y decir si son iguales o

diferentes. Guardar Datos: Puede recordar números importantes para usarlos después.

Acceder a Memoria: Puede guardar información en una especie de "memoria" y recuperarla más tarde.

3. ¿Cómo se usa?

Preparación: Asegurarse de que todos los cables estén conectados correctamente en la DPTR.

Datos de Entrada: Ingresa los números que quieres calcular y guardar en los

registros. Operación: Decide qué quieres hacer, sumar, restar, etc.

¡Hora del cálculo! ¡La DPTR hará el trabajo duro por ti y te dará la

respuesta! 4. Consejo Final:

¡Experimentar esta increíble! Prueba diferentes números y operaciones para ver qué sucede.

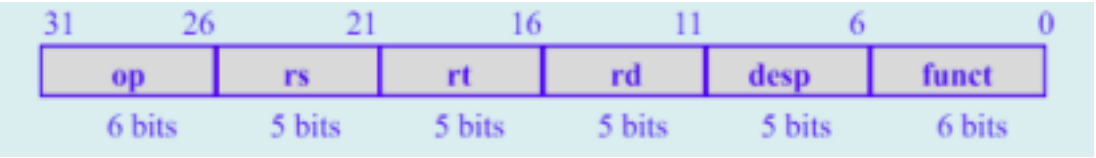
Objetivos

El objetivo principal es generar un Data-Path para la implementación/simulación de las instrucciones de tipo R (Registros).Este debe ser capaz de leer instrucciones cómo ADD, SUB, AND, OR, y SLT. Para la implementación se usarán módulos ya creados en prácticas anteriores como la ALU,Memoria y el Banco de registros, a su vez que se implementaran 3 modulos mas de control que son ALU-CONTROL,UNIDAD-CONTROL,MULTIPLEXOR 2 A 1,como el nombre lo dice ambas son para mandar las instrucciones que la memoria debe implementar a base de lo que desee el usuario,como un filtro para que el programa haga su funcionalidad en base a los bits significativos

Introducción

Descripción de la instrucción tipo R, con énfasis en explicar la instrucción SLT.

Las instrucciones tipo R tienen la siguiente estructura



Donde shamt(5 bits) para esta ocasión no se utiliza en instrucciones tipo R,esta instrucción se considera aritmético-lógicas

- op: Código de operación que identifica la instrucción.
- rs, rt: Identificadores de los registros fuente que contienen los operandos para la operación.
- rd: Identificador del registro destino donde se almacenará el resultado de la operación.
- shamt: Cantidad a desplazar en operaciones de desplazamiento. En las instrucciones tipo R, este campo generalmente no se utiliza y se reserva para desplazamientos.
- funct: Identificador de la operación aritmética o lógica a realizar. Este campo permite que diferentes operaciones se realicen utilizando la misma instrucción básica, diferenciándose por el valor de funct.

Además de estos campos, algunas instrucciones de tipo R pueden incluir campos

inmediatos o de dirección, dependiendo de la operación específica que estén realizando.
Por ejemplo, en una operación de turno o en algunas operaciones aritméticas y lógicas, puede utilizar el campo inmediato para especificar el turno o el valor inmediato que se va a operar.
La instrucción SLT compara los valores contenidos en los registros rs y rt. Si el valor en rs es menor que el valor en rt, el bit menos significativo (LSB) del registro rd se establece en 1; de lo contrario, se establece en 0. En otras palabras, la instrucción SLT realiza una operación de comparación aritmética y establece el registro de destino en 1 si el primer operando es menor que el segundo operando y en 0 en caso contrario.

Investigación sobre la operación ternaria.

La sintaxis de la operación ternaria sería la siguiente :
condición ? expresión_si_verdadero : expresión_si_falso

Donde:

- condición es una expresión booleana que se evalúa como verdadera o falsa. ● expresión_si_verdadero es el valor o la expresión que se devuelve si la condición es verdadera.
- expresión_si_falso es el valor o la expresión que se devuelve si la condición es falsa.

Básicamente esta operación se utiliza para realizar una condición booleana y establecer el proceso que llevará a cabo si se cumple,y a su vez también establecer qué hacer si no se cumple,bien puede hacer un proceso completo o solo retorna/asigna un valor

Investigación sobre el proceso de Compilación.

La compilación es fundamental para cualquier lenguaje de programación y consta que un lenguaje es traducido para que la máquina o procesador sea capaz de leerlo y ejecutarlo para cuestionar su funcionamiento

Este proceso consta de 6 etapas:

1-Análisis léxico (Scanner)
En esta etapa, el código fuente se divide en tokens, que son unidades de vocabulario básico, como palabras clave, identificadores, constantes y operadores. También se eliminan los comentarios y se cuidan los espacios en blanco. El resultado es una serie de tokens que pasarán a la siguiente etapa.
2-Análisis de sintaxis (Parser)
El análisis sintáctico verifica que la secuencia de tokens generada por el análisis léxico sigue la sintaxis del lenguaje de programación. Cree un árbol de sintaxis abstracta (AST) para representar la estructura jerárquica del código fuente. Si el código no se ajusta a la sintaxis del idioma, se producirá un error de sintaxis.

3-Análisis Semántico
En esta etapa, se realizan comprobaciones semánticas más avanzadas en el AST para garantizar que el código se ajuste a las reglas semánticas del lenguaje de programación. Esto incluye verificación de tipos, verificación de alcance variable y otras reglas semánticas del lenguaje. Si se detecta una violación de estas reglas, se generará un error.
4-Generación de código intermedio
Después de pasar las verificaciones semánticas, el compilador genera un código intermedio que representa el programa en una forma más abstracta y portátil. Este código intermedio puede ser independiente de la arquitectura del procesador objetivo y facilita la optimización y la portabilidad del código.
5-Optimización
Durante esta fase, se aplican varias técnicas de optimización al código intermedio generado para mejorar su eficiencia en términos de tiempo de ejecución, uso de memoria y consumo de energía. Las optimizaciones incluyen eliminación de códigos muertos, reordenamiento de instrucciones, optimización de bucles, etc.
6-Generación
Finalmente, el compilador genera código objeto en lenguaje de máquina específico para la arquitectura del procesador de destino. Este código objeto puede tener la forma de código de máquina nativo, código ensamblador o código de máquina virtual intermedio. Una vez que se compila un programa completo, se vincula con bibliotecas externas y se empaqueta en un archivo ejecutable.

Conclusión

Javier:El Convertidor de Instrucciones a Binario proporciona una herramienta eficiente y fácil de usar para convertir instrucciones escritas en lenguaje ensamblador a su representación binaria de 32 bits. Este programa es útil para programadores y diseñadores de hardware que necesitan traducir instrucciones legibles por humanos en un formato que pueda ser ejecutado por un procesador.Durante el desarrollo de este proyecto, se han logrado los siguientes objetivos:

Implementación de una interfaz gráfica de usuario (GUI) intuitiva utilizando la biblioteca tkinter de Python, lo que facilita la interacción con el programa.

Diseño de un algoritmo eficiente para convertir las instrucciones ensamblador a su equivalente binario, utilizando un conjunto limitado de instrucciones compatibles.

Manejo de errores y excepciones para proporcionar una experiencia de usuario fluida y evitar fallos inesperados durante la conversión.

Aunque este convertidor actualmente admite un conjunto limitado de instrucciones, puede ser expandido en el futuro para incluir soporte para más instrucciones y funcionalidades avanzadas.

En resumen, el Convertidor de Instrucciones a Binario es una herramienta valiosa que simplifica el proceso de conversión de instrucciones ensamblador a binario, ayudando a los desarrolladores a trabajar de manera más eficiente y precisa en proyectos relacionados con la programación de bajo nivel y el diseño de hardware.

Juan:Para finalizar con este trabajo creo que debería redactar o explicar mejor las indicaciones ya que fueron muy ambiguas y al momento de realizar se hizo una cosa, pero al momento de entregar pidió otra entonces, para reflexionar.

En cuanto al trabajo en general realizar los módulos fue sencillo, crear lo de Python fue enredoso porque no teníamos bien claro que hacer pero en general creo que a todos nos quedó bien claro que hacer en las próximas entregas, en cuanto a la parte que a mi me tocó todo muy sencillo exceptuando la parte de GitHub que ni yo sabia que hacer jaja saludos.

David:El desarrollo de un software estilo simulación como el que se realizó en esta actividad es de alta dedicación por lo que sí puede llevar su tiempo la implementación de un proyecto que se patente de manera oficial o profesional,los algoritmos que se implementaron se conectan entre sí para la escritura y lectura de archivos de datos cargados haciendo la función de instrucciones tipo Registro y más con la implementación o el manejo de archivos en lenguaje ensamblador y/o binario hace el codigo/programa más universal y pienso que se puede seguir desarrollando para crear cosas más específicas y rigurosas que lleven procesos mucho más complicados y complejos

Benjamin:En resumen, este proyecto de la DPTR en Verilog ha sido un verdadero desafío. Nos ha llevado tiempo entender cómo funciona cada parte y cómo hacer que todo funcione correctamente. Ha habido momentos frustrantes cuando las cosas no salían como esperábamos, pero también ha habido momentos emocionantes cuando logramos que la DPTR hiciera lo que queríamos. A lo largo del proyecto, hemos aprendido mucho sobre programación, circuitos y cómo se comunican entre sí.

FUENTES BIBLIOGRÁFICAS

https://riunet.upv.es/bitstream/handle/10251/173846/Todorov%20-%20Diseno%20e%20implementacion%20de%20un%20procesador%20MIPS%20R2000%20con%20el%20simulador%20de%20circuitos%20logic....pdf

Smith, J. (2024, 24 febrero). *Phases of Compiler with Example: Compilation Process & Steps*. Guru99. https://www.guru99.com/es/compiler-design-phases-of-compiler.html

https://hades.mech.northwestern.edu/images/1/16/MIPS32_Architecture_Volume_II-A_Instruction_Set.pdf

Operador condicional (ternario) - JavaScript | MDN. (2023, 16 noviembre). MDN Web Docs.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_operator