



Nombre de los alumnos:

Emmanuel Alejandro Barba Lugo

María Emma Valeria González Medina

Diego Alberto Maldonado Meléndez

Octavio Salvador Villegas Navarro

Clave: IL365

Materia: Arquitectura de computadoras

Sección: D03

Profesor: Jorge Ernesto López Arce Delgado

“FASE 1”

Introducción

MIPS32 es una arquitectura de alto desempeño estándar en la industria que provee un set de instrucciones robusto y bien definido, con soporte para una amplia gama de herramientas de desarrollo de hardware y software como compiladores, debuggers, emuladores en circuito, middleware, plataformas de aplicación y referencias de diseño.

La arquitectura MIPS se basa en un set de instrucciones que utiliza un modelo de carga y almacenamiento de datos, en el cual las operaciones se realizan sobre operandos en los registros del procesador. La memoria principal solo es accedida por las instrucciones de carga y almacenamiento.

Esta arquitectura tiene una ventaja substancial en cuanto a costo y rendimiento sobre otras implementaciones de microprocesadores que se basan en arquitecturas tradicionales.

El ISA (Instruction Set Architecture) de MIPS define una familia de instrucciones que operan con datos de 32 bits dentro de la estructura de la arquitectura MIPS. En el ISA se incluyen todas las instrucciones con y sin privilegios con las cuales el programador se comunica con el procesador.

El concepto de Endianness hace referencia al orden en que los bytes de la memoria de una computadora son leídos. La información puede ser representada de dos formas: Big-Endian y Little-Endian.

Big-Endian almacena primero el bit más significativo. Al leerse múltiples bytes, el primer byte o la dirección más baja de memoria será el más significativo. Little-Endian hace lo contrario, almacenando primero el bit menos significativo.

En el diseño de nuestro procesador y el decodificador de instrucciones se utilizará Big-Endian.

Set de instrucciones

Instrucción	Tipo	Sintaxis
Add	R	Add \$rd, \$rs, \$rt
Sub	R	Sub \$rd, \$rs, \$rt
Mul	R	Mul \$rd, \$rs, \$rt
Div	R	Div \$rs, \$rt
Or	R	Or \$rd, \$rs, \$rt

And	R	And \$rd, \$rs, \$rt
Addi	I	Addi \$rt, \$rs, immediate
Subi	I	Subi \$rt, \$rs, immediate
Ori	I	Ori \$rt, \$rs, immediate
Andi	I	And \$rt, \$rs, immediate
Lw	I	Lw \$rt, offset
li	I	li \$rd, value
la	I	la \$rd, exp
Sw	I	Sw \$rt, offset
slt	R	slt \$rd, \$rs, \$rt
Slti	I	Slti \$rt, \$rs, immediate
beq	I	beq \$rs, rt, offset
bne	I	bne \$rs, rt, offset
j	J	J target
jal	J	jal target
jr	J	jr \$rs
nop	R	nop
bgtz	I	bgtz \$rs, offset

Ljubisa Bajic:

Ljubisa Bajic es un veterano de la industria de semiconductores, trabajando exhaustivamente en diseño y debug de VLSI, además de tener amplia experiencia en software y arquitectura de aceleración.

Ljubisa se desempeñó 10 años dentro de AMD como arquitecto de ASIC trabajando en manejo de energía y diseño DSP, antes de entrar a NVIDIA como arquitecto senior, para después regresar a AMD por algunos años.

Dejó AMD para iniciar Tenstorrent en 2016, una compañía que cofundó para traer al mercado soluciones en cómputo de IA con un nuevo acercamiento que responde al crecimiento exponencial de la complejidad de los modelos de IA.

Jim Keller:

James B. Keller es un ingeniero de microprocesadores americano, con un trabajo pionero en la arquitectura de computadoras que abarca tres décadas. Su carrera en el desarrollo de CPUs inició en 1984, siendo parte del equipo que diseñó el chip Alpha de DEC, lanzado en 1992 y que corría a 500 MHz con una memoria caché de 1 GHz.

En 1998 se une a AMD con la idea de integrar el procesador con unidades separadas como la memoria y la transferencia de datos, ideas que fueron implementadas en los chips K8.

Keller también fue pionero en la idea de los diseños de doble núcleo. Además, trabajó en chips basados en PowerPC y en ARM para empresas como Apple y Samsung.

De vuelta en AMD en 2012, Keller sentó las bases para el diseño de la microarquitectura Zen. En 2015 se une a Tesla para el diseño de chips que serían implementados en los automóviles de la compañía. En 2018 se une a Intel como vicepresidente, sin embargo, abandonándolo en junio de 2020.

Raja Koduri

Raja Koduri es un ingeniero y ejecutivo en hardware de gráficos de computadora. Koduri ha dirigido los esfuerzos en gráficos de AMD como vicepresidente y arquitecto en jefe para Radeon Technologies.

Antes de eso, se desempeñó como director en arquitectura de gráficos para Apple, donde ayudó a crear el subsistema de gráficos para las computadoras Macintosh de la compañía. Ha ocupado diversos puestos como líder en el sector de gráficos para computadora en AMD.

Koduri se unió a Intel en 2017 como arquitecto en jefe, donde se concentró en continuar el crecimiento de los esfuerzos de la compañía en materia de gráficos.

“FASE 2”

Introducción

En este reporte explicaremos en que consiste el algoritmo que vamos a utilizar para crear el código en lenguaje ensamblador, también explicaremos los módulos creados en Verilog, diremos nuestros objetivos y daremos una conclusión sobre este proyecto y la materia.

Torres Hanói

Las Torres de Hanói es un rompecabezas o juego matemático inventado en 1883 por el matemático francés Édouard Lucas. Este juego de mesa individual consiste en un número de discos perforados de radio creciente que se apilan insertándose en uno de los tres postes fijados a un tablero. El objetivo del juego es trasladar la pila a otro de los postes siguiendo ciertas reglas, como que no se puede colocar un disco más grande encima de un disco más pequeño. El problema es muy

conocido en la ciencia de la computación y aparece en muchos libros de texto como introducción a la teoría de algoritmos.

La fórmula para encontrar el número de movimientos necesarios para transferir n discos desde un poste a otro es: $2^n - 1$.

Historia

El rompecabezas fue inventado por el matemático francés Édouard Lucas en 1883. Se cuenta una historia sobre un templo en la India en Kashi Vishwanath que contiene una gran sala con tres postes gastados por el tiempo, rodeada de 100 discos dorados. Los sacerdotes de Brahma, actuando bajo el mandato de una antigua profecía, han estado moviendo estos discos de acuerdo con las reglas inmutables de Brahma desde ese momento. Por lo tanto, el acertijo también se conoce como el rompecabezas de la Torre de Brahma. Según la leyenda, cuando se complete el último movimiento del rompecabezas, el mundo se terminará. No está claro si Lucas inventó esta leyenda o si se inspiró en ella.

Si la leyenda fuera cierta, y si los sacerdotes pudieran mover los discos a una velocidad de uno por segundo, utilizando el menor número de movimientos, completar la tarea les llevaría $2^{64} - 1$ segundos, o aproximadamente 585.000 millones de años, que es aproximadamente 42 veces la edad actual del Universo.

Existen muchas variaciones en esta leyenda. Por ejemplo, en algunos relatos el templo es un monasterio, y los sacerdotes son monjes. Se puede decir que el templo o monasterio se encuentra en diferentes partes del mundo, incluidos Hanói, Vietnam, y puede estar asociado con cualquier religión. En algunas versiones, se introducen otros elementos, como el hecho de que la torre fue creada en el comienzo del mundo, o que los sacerdotes o monjes solo pueden hacer un movimiento por día.

¿Cómo se resuelve el problema de Las Torres de Hanoi?

Para resolver el problema de Las Torres de Hanoi, se deben seguir ciertas reglas:

- Se tienen tres varillas y una pila de discos de diferentes tamaños en una de las varillas.
- El objetivo es mover toda la pila de discos a otra de las varillas, utilizando la tercera varilla como auxiliar.
- Solo se puede mover un disco a la vez y no se puede colocar un disco más grande sobre uno más pequeño.

La solución se puede lograr utilizando la recursividad, donde se mueve la pila de discos como si fuera una unidad, y se repite el proceso para una pila de discos más pequeña.

El algoritmo básico para resolver Las Torres de Hanoi con N discos sería:

- Mover la pila de discos superiores (N-1) de la varilla inicial a la varilla auxiliar.
- Mover el disco más grande (N) de la varilla inicial a la varilla de destino.
- Mover la pila de discos superiores (N-1) de la varilla auxiliar a la varilla de destino.

Pseudocódigo:

def mover(discos, origen, destino, auxiliar):

if discos > 0

mover(discos-1, origen, auxiliar, destino)

mover(discos-1, auxiliar, destino, origen)

“FASE 3”

Introducción

En la presente fase agregaremos cinco buffers a las fases anteriores para que cada que haya un flanco de subida de PC y con esto, lo que entre a cada buffer salga. Para ello, es necesario conocer el concepto de Pipeline.

La técnica de Pipeline, en español tubería, se trata de la implementación en la que múltiples instrucciones son superpuestas en la ejecución, en estos días esta técnica es muy utilizada.

El no-pipeline puede verse en un sencillo ejemplo de lavandería, en el que se ejecutan los siguientes pasos:

1. Coloque una carga de ropa sucia en la lavadora.
2. Cuando termine la lavadora, coloque la carga húmeda en la secadora.
3. Cuando termine la secadora, coloque la carga seca sobre una mesa y dóblela.
4. Cuando termine de doblar, pídale a su compañero de cuarto que guarde la ropa.

Cuando su compañero de cuarto haya terminado, comienza de nuevo con la siguiente carga sucia.

Por el contrario, el pipeline tomaría mucho menos tiempo ya que, en este caso, a medida que la lavadora se termina con la primera carga y se coloca en la secadora, usted carga la lavadora con la segunda carga sucia. Cuando la primera carga esté seca, la coloca en la mesa para comenzar a doblar, mueva la carga húmeda a la

secadora y coloque la siguiente carga sucia en la lavadora. A continuación, le pide a su compañero de cuarto que guarde la primera carga, comienza doblando la segunda carga, la secadora tiene la tercera carga, y pone la cuarta carga en la lavadora. En este punto, todos los pasos, llamados etapas en el pipeline, están operando. Al mismo tiempo siempre que tengamos recursos separados para cada etapa podemos canalizar las tareas.

El secreto en que el pipeline sea mucho más rápido que otros métodos, es que al hacer que todo funcione en paralelo, las instrucciones se ejecutan de manera que no se desperdicia tiempo y siempre está en función el programa. Por ello, la canalización mejora el rendimiento de nuestro sistema.

En el ejemplo de la lavandería, el pipeline no disminuiría el tiempo para completar una carga de ropa, pero cuando tenemos muchas cargas de ropa que lavar, la mejora en el rendimiento, disminuyendo así el tiempo total para completar el trabajo.

Si todas las etapas toman aproximadamente la misma cantidad de tiempo y hay suficiente trabajo por hacer, entonces la aceleración debido a la canalización es igual al número de etapas en el pipeline, en este caso cuatro: lavado, secado, doblado y guardado. Por lo tanto, la ropa con pipeline es potencialmente cuatro veces más rápida que la que no tiene pipeline: 20 cargas toman alrededor de 5 veces más que 1 carga, mientras que 20 cargas de ropa secuencial toma 20 veces más largo que 1 carga. Esto lo podemos ver gráficamente en la siguiente imagen:

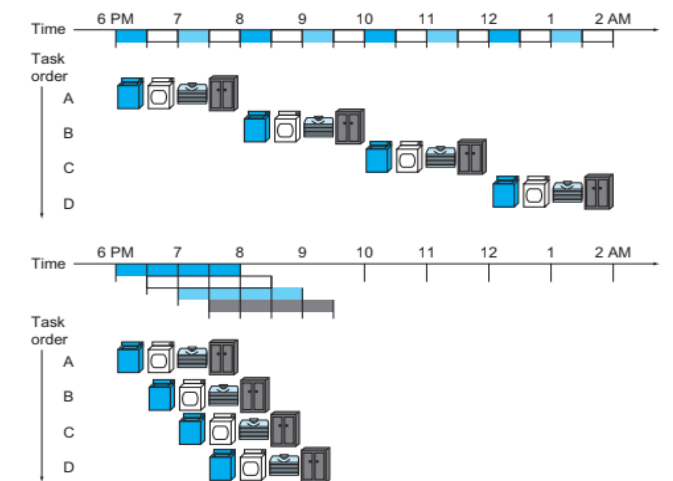


FIGURE 4.25 The laundry analogy for pipelining. Ann, Brian, Cathy, and Don each have dirty clothes to be washed, dried, folded, and put away. The washer, dryer, "folder," and "storer" each take 30 minutes for their task. Sequential laundry takes 8 hours for 4 loads of wash, while pipelined laundry takes just 3.5 hours. We show the pipeline stage of different loads over time by showing copies of the four resources on this two-dimensional time line, but we really have just one of each resource.

Como ya lo vimos anteriormente, los mismos principios se aplican a los procesadores en los que canalizamos la ejecución de instrucciones. Las instrucciones MIPS clásicamente toman cinco pasos:

1. Obtener instrucciones de la memoria.
2. Lea registros mientras decodifica la instrucción. El formato normal de las instrucciones MIPS permite que la lectura y la decodificación ocurran simultáneamente.
3. Ejecuta la operación o calcula una dirección.
4. Accede a un operando en la memoria de datos.
5. Escribe el resultado en un registro.

En resumen, el pipeline permite que múltiples instrucciones se ejecuten simultáneamente, lo que puede mejorar significativamente el rendimiento en comparación con la ejecución secuencial de instrucciones.

“FASE 1”

Objetivos

Objetivo general

- Implementar el single datapath de un microprocesador con arquitectura MIPS de 32 bits.

Objetivos particulares

- Ser capaz de ejecutar instrucciones de tipo R.
- Definir el set de instrucciones que se ejecutarán en el diseño del procesador.

“FASE 2”

Objetivos

Objetivo general

- Implementar el single datapath de un microprocesador con arquitectura MIPS de 32 bits.

Objetivos particulares

- Agregar los módulos necesarios al datapath para poder ejecutar las instrucciones tipo I de la tabla 1 y tabla 2.

“FASE 3”

Objetivos

Objetivo general

- Implementar el single datapath de un microprocesador con arquitectura MIPS de 32 bits.

Objetivos particulares

- Ejecutar instrucciones de tipo J.
- Redacción y descripción del desarrollo de los módulos que tienen el datapath.

“FASE 1”

Desarrollo

En esta fase se implementó el single datapath de nuestro procesador MIPS de 32 bits. Este cuenta con los siguientes módulos:

- PC o Program Counter, el cual consta de un registro que se encarga de llevar seguimiento de la secuencia del programa.
- La memoria de instrucciones, la que almacena la instrucción actual.
- El adder, que incrementa el PC una cantidad predeterminada cada ciclo de reloj.
- La unidad de control, que envía señales a las otras unidades del datapath para indicarles la respuesta adecuada a cada instrucción.
- El banco de registros, el cual contiene los registros para almacenar información. Tiene puertos de lectura y escritura.
- La ALU, que ejecuta una operación aritmética sobre dos operandos.

“FASE 2”

Desarrollo

Fase 1:

Lo primero que se creo fue el ciclo Fetch, el módulo PC se comporta como el buffer, lo que recibe manda pero como su entrada es un cable que está conectado al módulo del sumador que se encarga de siempre sumar 4 bits (porque cada set de instrucciones está compuesto por 4 direcciones de memoria del módulo Mem Inst que sirve como un run y donde se cargan el archivo de instrucciones a ejecutar) pero este módulo recibe de entrada la salida del módulo PC, por lo tanto es necesario utilizar un initial para hacer funcionar el módulo PC. Una vez hecho el ciclo Fetc la salida del módulo Mem Inst se divide para ser la entrada de distintos

módulos, una parte de bits que representan el OP se dirige al módulo UC donde se decide si los datos se van a escribir, leer, guardar por lo tanto sus salidas se conectan al módulo Banco de registros, Mem Datos y un Multiplexor, otra cantidad de bits de la salida Mem Inst se va al módulo del Banco de registro, estos representan: RS, RT y RD, los primeros dos son los operandos de la operación que va a realizar la ALU y se mandan al módulo de la ALU y el último será la dirección de memoria donde se guardara el dato, y la última parte de bits del módulo Mem Inst se va al módulo de ALU Control, este representa el Function y decide qué operación realiza la ALU, por lo tanto su salida se conecta con el módulo ALU, este módulo realizara la operación y su salida se conecta al módulo Mem Datos que es como el módulo RAM pero en lugar de tener una sola entrada que decide si lee o escribe, este tiene una entrada para cada una de esas indicaciones y esas entradas reciben el dato del módulo UC, su salida se conecta al módulo Multiplexor que luego se conecta su salida al módulo Banco de Registros, por lo tanto todos los módulos de integran en un solo módulo que solamente tendrá el input el cual es CLK y se conecta al módulo PC, y este módulo es el utilizado en el TB.

La forma de revisar su funcionamiento es dando un vistazo a la memoria del módulo Banco de registros después de haber realizado el TB, el cual solamente necesita tener un input para que inicialice CLK que a su vez va a inicializar el módulo PC.

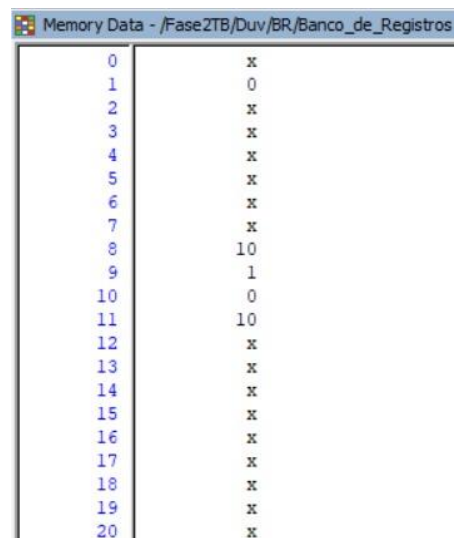
Address	Value
0	25
1	145
2	200
3	5
4	7
5	7
6	x
7	x
8	x
9	x
10	150
11	0
12	5
13	7
14	x
15	x
16	x
17	x
18	x
19	x
20	x
21	x
22	x
23	x
24	x

Fase 2:

Para la fase 2 se tuvo que modificar el módulo UC, se añadieron nuevos case que tienen el OP de los distintos datos tipo I y se añaden nuevas salidas, lo mismo ocurre con el módulo de ALU y ALU Control, también se crean los siguientes módulos:

módulo Sign-extend que recibe los bits 15:0 de la salida del módulo Mem Inst y luego los convierte en 32 bits, para eso se multiplica 16 veces el bit 15 y se concatena con la entrada del módulo y dicha salida se va a 2 nuevos módulos, uno es un multiplexor que recibe la de entrada esta salida y también recibe de entrada el segundo dato de salida del Banco de registro y para decidir cuál dato sale se recibe de entrada la salida ALUSrc del módulo UC. El segundo módulo al que se dirige la salida del Sign-extend es al nuevo módulo Shift left 2 que hace una corrida de 2 bits que luego se dirige su salida a un nuevo módulo ADD que lo recibe de entrada y también recibe de entrada la salida del módulo Sumador, este módulo realiza la suma y manda la salida a un nuevo módulo que es otro multiplexor que lo recibe de entrada junto con la salida del módulo Sumador, para decidir cuál pasa se recibe la salida del nuevo módulo Branch que recibe de entrada la salida Branch del módulo UC y la salida Zero del módulo ALU y este multiplexor manda su salida a la entrada del módulo PC. Finalmente se crea un módulo que es un multiplexor que recibe de entrada los bits 20:16 y 15:11 de la salida del módulo Mem Inst y para decidir cuál pasa se recibe de entrada la salida RegDst del módulo UC y después el multiplexor manda la salida a la entrada WA del módulo Banco de registros.

El TB es igual que el de la fase anterior porque el módulo que engloba todos los módulos creados solamente tiene de entrada el CLK para el módulo PC.



Memory Data - /Fase2TB/Duv/BR/Banco_de_Registros

0	x
1	0
2	x
3	x
4	x
5	x
6	x
7	x
8	10
9	1
10	0
11	10
12	x
13	x
14	x
15	x
16	x
17	x
18	x
19	x
20	x

“FASE 3”

Desarrollo

En esta fase se implementó el single datapath de nuestro procesador MIPS de 32 bits, el cual se adaptó para que, además de ejecutar las instrucciones R e I, se

ejecutan las instrucciones tipo J. Este cuenta con nuevos módulos nombrados a continuación:

- Shift-left 2
- Jump en la unidad de control
- Mux

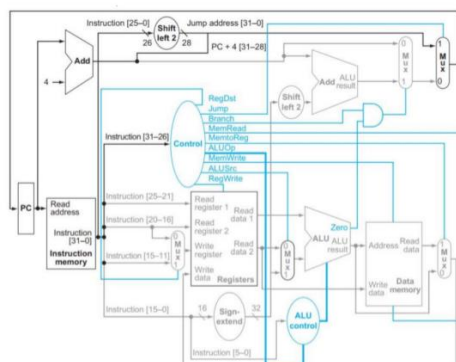
El nuevo shift-left 2 hace un desplazamiento binario en el que todos los bits de un valor se mueven dos posiciones hacia la izquierda. En esta operación, se insertan dos ceros en los bits menos significativos y los dos bits más significativos se descartan. Esta operación se ejecuta con Instrucción en la cual, se tienen los bits del 25 al 0, recibiendo así 26 bits y dando por salida 28 bits; los cuales mediante un jump adress puede saltar a una dirección en memoria que este entre 31 a 0 bits, es decir, 32 bits posibles.

Así mismo, es importante recordar que cada set de instrucciones está compuesto por 4 direcciones de memoria del módulo Mem Ins; por lo que, en PC puede utilizar el espacio en memoria de 31 a 28. El resultado de esto entra al nuevo multiplexor como 1 y el 0 de este multiplexor es el resultado del multiplexor anterior implementado con el Branch en la fase 2. Como 'selector' de este mux, se implementará una nueva instrucción en la unidad de control llamada Jump la cual, será la encargada de transferir el control del flujo de ejecución del programa a una ubicación de memoria específica.

El resultado del nuevo multiplexor irá directamente a la entrada de PC.

De esta manera, para la fase 3 se modificó el módulo UC, se añadió un nuevo case que tienen el OP para ejecutar la instrucción tipo J y se añadieron nuevas salidas, lo mismo ocurre con el módulo de ALU y ALU Control.

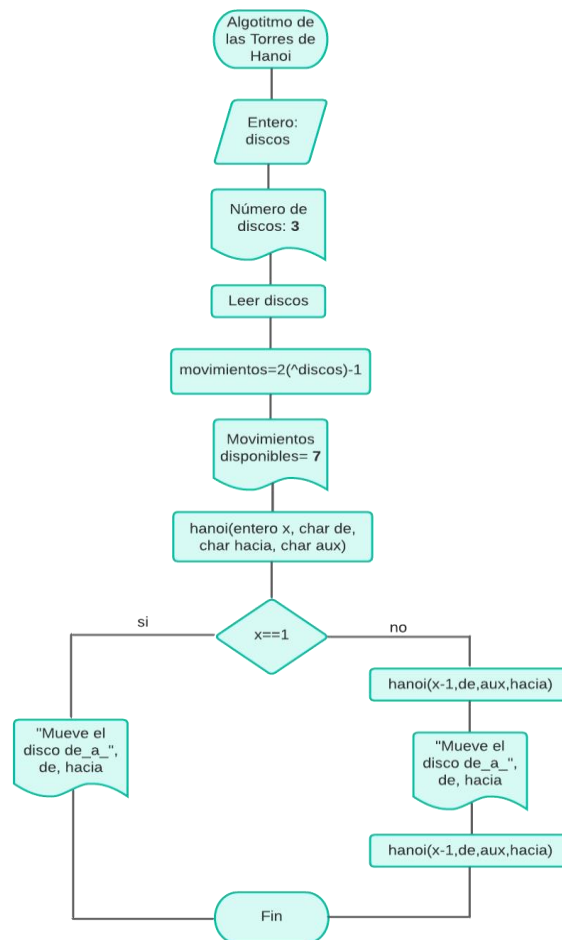
Algo nuevo que se implementó en esta fase fueron los buffers que, se utilizan para mejorar el rendimiento y la eficiencia del procesador. De igual manera, como en las fases anteriores las conexiones de módulos se hicieron en el módulo top llamado Fase 3, se agregan las conexiones anteriores y se implementan las nuevas conexiones. Para la comprobación se realizó un Test Bench, en el que se encuentra el CLK que está conecta al módulo PC.



Comprobación del programa:

0	0
1	10
2	10
3	10
4	0
5	10
6	x
7	x
8	10
9	1
10	0
11	10
12	x
13	15
14	x
15	x
16	x
17	x
18	x
19	x
20	x
21	x
22	x
23	x
24	x
25	x
26	x
27	x
28	x
29	x
30	x
31	x

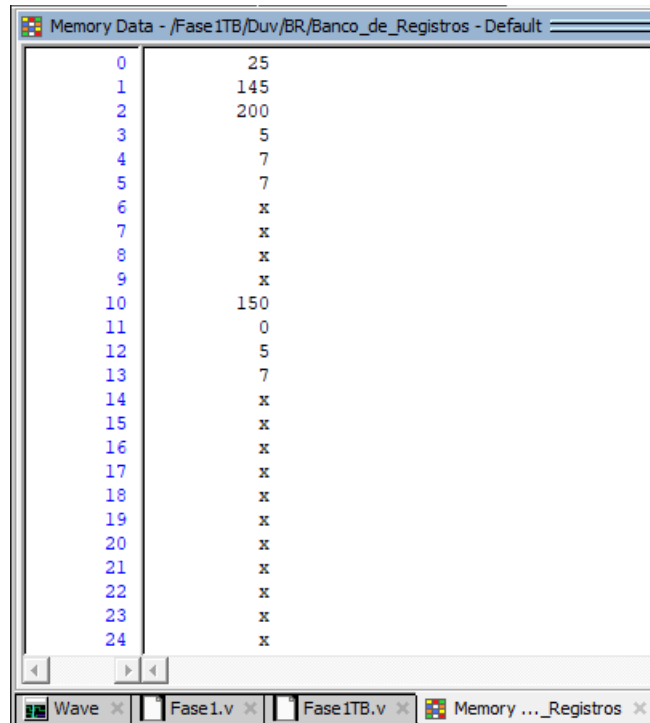
Diagrama de flujo del algoritmo a implementar en código ensamblador es:



“FASE 1”

Conclusiones

Con la fase 1 implementada, ahora debería de poder ejecutar instrucciones de tipo R. Para evaluar el funcionamiento del sistema se precargaron instrucciones en código binario a la memoria de instrucciones.



Address	Value
0	25
1	145
2	200
3	5
4	7
5	7
6	x
7	x
8	x
9	x
10	150
11	0
12	5
13	7
14	x
15	x
16	x
17	x
18	x
19	x
20	x
21	x
22	x
23	x
24	x

“FASE 2”

Conclusiones

Este proyecto me ayudo a darme cuenta de cosas que no había notado antes y a reforzar conocimientos, por ejemplo, los case que utilizamos cuando creamos el módulo ALU por primera vez en realidad son los ALUOP y cuando creamos el módulo Banco de registros tuvimos que poner que siempre lea datos a pesar de que no siempre escriba porque de no ser así cuando usemos este módulo en el proyecto daría fallos.

El formato de la clase me gusta, pero hubiera preferido que el profesor hiciera el código con nosotros porque en ocasiones el problema no era que no supiera qué hacer, sino que desconocía la sintaxis del lenguaje Verilog lo que provocaba ciertos retos y disgustos.

“FASE 3”

Conclusiones

En conclusión, consideramos como equipo que la fase 3 fue un poco más difícil a comparación de las fases anteriores ya que, nos resultó complejo la implementación del jump adress en Verilog y los buffers.

Por otro lado, nos resultó bastante interesante la instrucción tipo J (Jump) que se utiliza para realizar saltos incondicionales en el flujo de ejecución del programa. Nos gustó que esta instrucción permiten transferir el control del programa a una dirección de memoria específica, sin basarse en condiciones o resultados de cálculos previos.

Y ahora que tenemos nuestro proyecto final concluido entendemos la importancia de cada una de las clases que tuvimos a lo largo del semestre, sin duda alguna sin esas clases no habiéramos podido lograr completar nuestro proyecto.

Referencias

- Patterson, D. A., & Hennessy, J. L. (2013). *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann.
- MIPS (2014). MIPS Architecture For Programmers Volume I-A: Introduction to the MIPS32 Architecture. Obtenido de <https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00082-2B-MIPS32INT-AFP-06.01.pdf>
- MIPS (2016). MIPS Architecture For Programmers Volume II-A: The MIPS32 Instruction Set Manual. Obtenido de <https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf>
- Ahmad, M. (2021). The story of Jim Keller and his pioneering work on chip design and architecture. Obtenido de <https://www.edn.com/the-story-of-jim-keller-and-his-pioneering-work-on-chip-design-and-architecture/>
- Cutress, I. (2021). An Interview with Tenstorrent: CEO Ljubisa Bajic and CTO Jim Keller. Obtenido de <https://www.anandtech.com/show/16709/an-interview-with-tenstorrent-ceo-ljubisa-bajic-and-cto-jim-keller>
- Martin, D. (2023). Raja Koduri, Main Driver Of Intel's GPU Efforts, To Leave Company. Obtenido de <https://www.crn.com/news/components-peripherals/raja-koduri-main-driver-of-intel-s-gpu-efforts-to-leave-company>
- Parr, K. (2021). What is Endianness? Big-Endian vs Little-Endian Explained with Examples. Obtenido de <https://www.freecodecamp.org/news/what-is-endianness-big-endian-vs-little-endian/>

- Arumals. (2023, 19 mayo). Funciones Recursivas, Las Torres de Hanoi. *Apuntes de Programación*. <https://apuntes.de/golang-estructuras-de-datos-y-algoritmos/funciones-recursivas-las-torres-de-hanoi/#gsc.tab=0>