**Laboratorio 3 “Pipeline y Hazard”**

**organización de computadores**

|  |  |  |
| --- | --- | --- |
| Nombre: | Matias Coronado |  |
| Profesor: | Leonel Medina |  |
| Ayudante: | Ricardo Álvarez |  |
| Fecha de Entrega: | 23 de noviembre |  |

Santiago de Chile

2 - 2017

Tabla de contenido

[Capítulo 1. Introducción 1](#_Toc533197109)

[1.1 Enunciado del problema 1](#_Toc533197110)

[1.2 Motivación 1](#_Toc533197111)

[1.3 Objetivos 1](#_Toc533197112)

[1.3.1 Objetivo general 1](#_Toc533197113)

[1.3.2 Objetivos específicos 1](#_Toc533197114)

[1.4 Herramientas 2](#_Toc533197115)

[1.5 Estructura del informe 2](#_Toc533197116)

[Capítulo 2. Marco teórico 3](#_Toc533197117)

[Capítulo 3. Desarrollo 4](#_Toc533197118)

[Capítulo 4. Experimentos a realizar 6](#_Toc533197119)

[4.1 Resultados 6](#_Toc533197120)

[4.2 Análisis de resultados 7](#_Toc533197121)

[Capítulo 5. Conclusión 9](#_Toc533197122)

[Capítulo 6. Referencias 11](#_Toc533197123)

Índice de Figuras

Figura 4.1 Salida Hazard y Hazard solución …………………………… 6

Figura 4.2 Salida registros ……...………………………………………. 6

Figura 4.3 Instrucciones de entrada ………………...…………………... 7

Índice de tablas

# Introducción

## Enunciado del problema

Como segundo enunciado del laboratorio de organización de computadores, se solicitó conformar un programa capaz de leer múltiples instrucciones MIPS, tomando en consideración a los posibles “Hazard” que puedan existir en la lectura del “camino de datos”. Además, cada problema deberá ser “arreglado” por medio de elementos a presentar en el siguiente capítulo.

## Motivación

La correcta implementación del programa conllevara a un mejor conocimiento en cuento a “camino de datos tipo pipeline”, lo cual da la posibilidad pulir y comprender de una mejor manera su funcionamiento.

## Objetivos

### Objetivo general

1. Simular correctamente el camino de datos para un procesador multi-ciclo tipo pipeline.
2. Simular funcionalidades de las instrucciones tipo MIPS.

### Objetivos específicos

1. Profundizar conocimiento sobre el funcionamiento de este tipo de camino de datos.
2. Mejorar la capacidad de trabajar con el lenguaje C, tomando en consideración tanto en la simplicidad que debe tener las funciones, además del uso correcto de memoria.

## Herramientas

Se utilizará el lenguaje de programación C, por medio del paradigma imperativo. Los archivos del programa se crearon y editaron por medio del editor de texto Sublime Text 3.1.1, por otro lado, los archivos fueron compilados utilizando el implementador de compiladores MINGW 5.3

## Estructura del informe

El presente informe está compuesto de 5 capítulos, dentro del actual se entrega el contexto del problema, indicando que motivación se tiene para resolverlo y las herramientas que se disponen para ello, además, se incluye de los principales objetivos que se desean cumplir. Dentro del segundo capítulo se describe los principales conceptos que son necesarios manejar el contenido problema. En el tercero se especificarán los elementos que dieron cabida a la creación de la solución del problema. En el cuarto, se presentarán las pruebas pertinentes a cada funcionalidad del programa, para finalmente entregar una conclusión en el último capítulo.

# Marco teórico

En este capítulo se dará a conocer los principales elementos a tener en consideración para entender el contexto del problema. Se tiene que MIPS es una arquitectura diseñada para leer “instrucciones” haciendo uso de “registros”, los cuales vendrían a ser la principal forma de almacenar variables. Pueden existir 3 tipos de instrucciones: **Tipo I:** Carga o almacenamiento de datos (SW, LW) y operaciones aritméticas con números inmediatos (ADDI, SUBI, ETC); **Tipo R**: Operaciones aritméticas (ADD, MUL, ETC) y **Tipo J:** Salto de instrucciones (JUMP, ETC).

MIPS hace uso de señales de control para diferenciar el funcionamiento de las instrucciones ingresadas, y de unidades de control, las cuales corresponden a las estructuras internas que manejan las instrucciones. Se tienen 2 tipos de lectura de instrucciones, los monociclos (lee una instrucción a la vez) y los tipos pipeline (múltiples instrucciones a la vez). Dentro de este laboratorio se dará uso del pipeline de 5 etapas:

1. **IF:** Busca cual instrucción se debe leer
2. **ID:** Se leen los registros involucrados
3. **EX**: Ejecución de la operación de la instrucción
4. **MEM:** Acceso a la memoria principal, con el fin de escribir o leer datos.
5. **WB**: Escribir resultados en registro.

La lectura de múltiples instrucciones conlleva a la aparición de problemas que causan resultados incorrectos, tales son denominados como “Hazard”, y se dividen en:

1. **Hazard estructural**: Relacionado cuando el hardware del procesador no tiene la capacidad se soportar más de una instrucción por ciclo de reloj.
2. **Hazard de datos:** Cuando se requiere de datos que aún no están disponibles, debido a que aún se encuentran trabajando en instrucciones anteriores
3. **Hazard de control**: Cuando existen cambios en el flujo de ejecución de las instrucciones (tales como en los casos de JUMP, BEQ, ETC)

Se dispone de las siguientes herramientas para “arreglar” los problemas:

1. **NOP:** Aplaza el ciclo de instrucciones en 1 ciclo, por medio de la agregación de una “nula”.
2. **FLUSH/NOP:** Método que se encarga de vaciar los buffers intermedios, agregando, además, un NOP, que permita solucionar los Hazard de control.
3. **Forwarding:** Método que sirve para resolver Hazard de datos al recuperar los faltantes por medio de buffers intermedios a las unidades de control.

# Desarrollo

A razón de entregarle un mayor orden (y entendimiento) al código, se decidió dividirlo en los archivos: funciones.c, main.c, estructuras.h, cabeceras.h y constantes.h. Para conformar la solución del problema se creó un lector MIPS del tipo pipeline 5 etapas, al que se le agregaba uno a uno las instrucciones. Antes de agregar una instrucción se procedía a realizar una verificación si existía algún Hazard de datos con las 2 subsiguientes, o un Hazard de control con la instrucción misma. Las estructuras que se utilizaron son:

1. **Instrucción**: Estructura donde se guardan los elementos de cada instrucción, tales como: tipo; operación; registros involucrados. Esta tiene la capacidad de almacenar los 3 tipos de instrucciones existentes.
2. **Memoria de datos**: Estructura en donde se almacena los valores de los arreglos relacionados a cada registro. Esta hace uso de una matriz de enteros, en donde cada columna indica cuales son los elementos de cada registro.
3. **Registros:** Estructura que almacena los valores de los registros por medio de una lista de enteros.
4. **Forwarding:** Estructura que se encuentra dentro de cada instrucción. Esta se encarga de almacenar los valores representativos al Forwarding que necesita la instrucción, estos pueden ser: posición de la instrucción problema, registro problema, tipo de solución (EX/MEM -> ID/EX o MEM/WB -> ID/EX).
5. **Pipeline:** Estructura que se encarga de almacenar los punteros a las instrucciones relacionadas a su correspondiente punto dentro del pipeline, como, por ejemplo, la estructura tendrá un puntero ID, para la instrucción A cuando sea el caso de que tenga que pasar por el punto ID.
6. **Buffer:** Esta estructura se encarga de almacenar los valores de las instrucciones que pasaron de una etapa a otra, sea el caso para: (IF/ID, ID/EX, EX/MEM y MEM/WB). Cuando existe un Forwarding, el programa utilizara esta estructura para acceder a los registros que se soliciten.

El programa inicia solicitando que se ingrese el nombre del archivo que contiene las instrucciones, una vez ingresado, se creara una lista con todas las instrucciones de este. Esta última será traspasada a la función *iniciarPipeline ()* la cual hará uso de un *contadorPrograma* para saber a cuál instrucción debe acceder, para luego ingresarla dentro del pipeline. El proceso anterior se realizará tantas veces sea necesario, hasta alcanzar la última instrucción guardada. Se destaca, que se realiza una verificación antes de agregar una nueva instrucción, esto con el fin de verificar si existe algún Hazard al ingresarla.

Se verifican los Hazard de control únicamente analizando si la instrucción ingresada es de tipo JUMP o BEQ, si es el caso, se tiene que:

* **Hazard control JUMP:** Solucionable agregando un FLUSH del buffer IF/ID, para luego agregar un NOP.
* **Hazard control BEQ**: Solucionable agregando un FLUSH del buffer IF/ID, ID/EX para luego agregar 2 NOP.

Por otro lado, para verificar los Hazard de datos, se debe realizar un análisis de las 2 instrucciones siguientes en los ciclos del pipeline, en búsqueda de registros que puedan traer perdida de datos. A criterio del programa, se procederán a agregar los NOP pertinentes para solucionarlo, además de indicarse a sí mismo cuales son los registros que deben modificarse.

# Experimentos a realizar

## Resultados

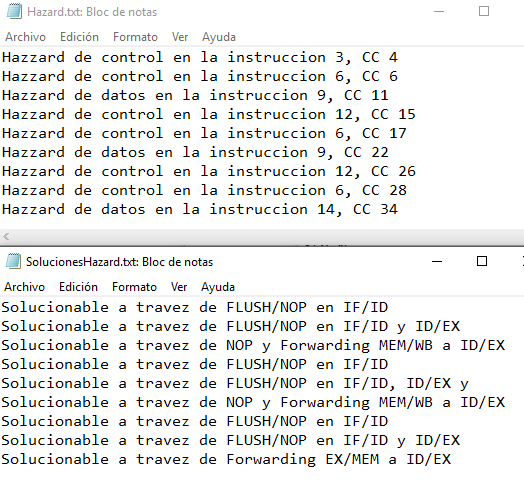
Con el fin de saber si el programa funciona correctamente, se decidió realizar una prueba con el archivo entregado junto al enunciado. Esta prueba intentara abarcar la mayor cantidad de variaciones de entradas posible, con el fin de analizar cómo se comporta el programa. A continuación, se presentará la entrada ingresada, junto con las salidas que se obtuvieron:

Figura 4.1 Salida Hazard y Hazard solución

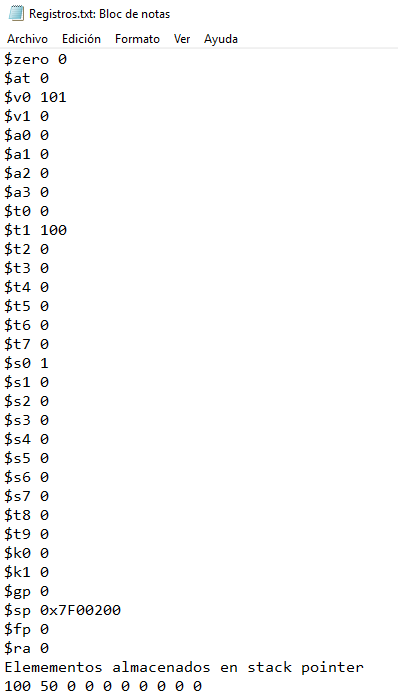


Figura 4.2 Salida registros

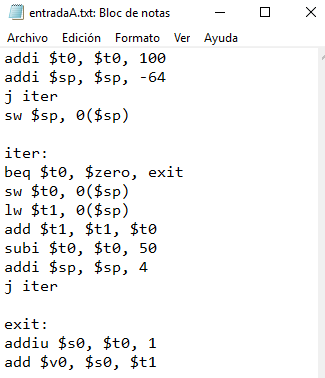


Figura 4.3 Instrucciones de entrada

## Análisis de resultados

A razón de los resultados obtenidos se puede afirmar que el programa funciona correctamente, dado los siguientes puntos:

* Como se puede observar en la figura 4.1, el programa reconoce la instrucción que posee el Hazard, además de indicar en cual ciclo de produjo.
* Dentro de la misma figura se puede observar que el programa se encarga de entregar todas las soluciones a cada uno de los Hazard presentados.
* Comparando lo obtenido con los resultados hipotéticos (entregados junto con el enunciado del laboratorio), se puede ver que los resultados de los registros son exactamente iguales. A diferencia de los elementos almacenados del “stack pointer”, que se representó de una manera diferente, pero alude a los mismos resultados.
* Comparando con los resultados hipotéticos se tiene que el reconocimiento de los Hazard fue el mismo, pero se tuvo una diferencia en cómo se trató los Hazard de control producidos por la instrucción *beq $t0, $zero, exit*, esto a raíz de que se tomó una medida diferente de cómo solucionarlo (Branch no taken), mientras que en el programa se aplico (Flush/NOP)

# conclusióN

A modo de conclusión, se alcanzó a cumplir con todos los objetivos propuestos, tanto en el sentido de implementar correctamente la lectura de instrucciones MIPS, como de crear un pipeline con la capacidad de leer paralelamente tales elementos. Se pudo comprender completamente cómo funciona el pipeline al tener que implementar las diferentes estructuras que lo conforman junto con la solución de los diferentes Hazard presentados.

Si bien fue difícil ver como seria la implementación del pipeline, este se pudo lograr de una manera correcta al dividir el problema principal en subproblemas de menor nivel. De igual forma, se tenía un avance previo a la realización del laboratorio, dado que el pasado trataba de un procesador monociclo que utilizaba el mismo sistema de instrucciones MIPS.

Se logro implementar todos los elementos escritos en “paper”, obteniendo un programa que cumple con lo solicitado, tanto en el sentido del orden que se obtuvo, como en el correcto uso de memoria. Se espera que en próximos laboratorios se puedan realizar de una manera más eficiente, dado lo aprendido en el presente.

# REFERENCIAS

M. Martin. (1998). MIPS Instruction reference. 13/10, de University of Idao Sitio web: <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>

J.M. Mendías. (2003). Diseño de la ruta de datos y la unidad de control. 15/10, de

Universidad Complutense de Madrid Sitio web: www.fdi.ucm.es/profesor/mendias/512/docs/tema16.pdf