

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA



Laboratorio N 1

Integrantes: Bastián Vera Palacios
Curso: Organización de Computadores
Sección 0-L-3
Profesor(a): Leonel Medina Daza
Ayudante : Ricardo Álvarez Zambrano

23 de Noviembre de 2018

Tabla de contenidos

1. Introducción	1
1.1. Contexto	1
1.2. Problema	1
1.3. Motivación	1
1.4. Objetivos	2
1.4.1. Objetivo general	2
1.4.2. Objetivos específicos	2
1.5. Propuesta de solución	2
1.6. Herramientas	2
1.7. Estructura del informe	2
2. Marco teórico	3
2.1. Librerías	3
2.2. MIPS (Procesador)	3
2.3. Registro	3
2.4. Punteros	3
2.5. Lectura y escritura de archivos	4
2.6. Memoria dinámica	4
2.7. Estructuras	4
3. Desarrollo	5
4. Experimentos a realizar	7
4.1. Resultados obtenidos	7
4.2. Análisis de resultados	8
5. Conclusiones	9
Bibliografía	10

1. Introducción

1.1. Contexto

La necesidad de buscar un medio de entretenimiento ha sido parte de la sociedad desde tiempos remotos. Así mismo es donde bajo este contexto en persia, alrededor del siglo V, se dio la creación del juego Tic Tac Toe, Tres en Línea, Ceros y Cruces, entre otros nombres. Este juego fue difundido por los musulmanes y posteriormente llevado a europa por comerciantes italianos. El juego tuvo un rápido reconocimiento, aunque tuvo un periodo oscuro donde por una asociación de este con rituales paganos macabros, este estuvo prohibido para todos los cristianos de fe catolica. (<https://deblogsyjuegos.wordpress.com>, 2015)

El juego consiste en una cuadrícula de 3x3 dentro de la cual cada jugador debe colocar su símbolo una vez por turno y no debe ser sobre una casilla ya jugada. El ganador del juego es el jugador el cual logre realizar una línea recta o diagonal entre 3 de sus símbolos.

1.2. Problema

En este laboratorio se le solicitó a los estudiantes de ingeniería informática de la Universidad de Santiago de Chile realizar mediante una serie de algoritmos en lenguaje C, un programa que reciba un archivo de entrada con instrucciones en formato MIPS y que permita generar una partida del juego gato, es decir, poder entregar quién fue el ganador de la partida, especificar el caso en que exista un empate o que las instrucciones hayan generado una jugada invalida ,todo esto con el fin de poder medir las habilidades adquiridas durante el desarrollo de éste curso y a la vez entregar la cantidad de veces que se ha utilizado cada etapa del camino de datos.

1.3. Motivación

La comprensión de un bajo nivel de abstracción se vuelve una ventaja comparativa al conocer cómo el software y hardware trabajan en conjunto y más específicamente, como este ultimo ejecuta las instrucciones de programa, ya que permite el desarrollo de programas más robustos, optimizados y que aprovechan al máximo el procesador.

1.4. Objetivos

1.4.1. Objetivo general

Simular el camino de datos de un procesador monociclo y determinar qué unidades funcionales del procesador son utilizadas en secuencias de instrucciones predefinida.

1.4.2. Objetivos específicos

Entender la importancia de las líneas de control en el camino de datos y el paso de las instrucciones a través de estas.

1.5. Propuesta de solución

A primera vista, se propone generar una estructura para definir el formato de una jugada. Posterior a esto se plantea manejar el tablero a través de un arreglo estático el cual maneje las posiciones del tablero. Una vez realizado lo anterior, se deberá realizar la lectura de las instrucciones y a partir de las distintas condiciones, ir insertando o eliminando jugadas, dependiendo del caso. Para finalizar se realizarán las comprobaciones finales en el tablero.

1.6. Herramientas

Dentro de las Herramientas para el desarrollo de este laboratorio se encuentran:

1. Lenguaje de programación C con sus respectivos recursos, tales como punteros y memoria dinámica, entre otros.
2. Librerías asociadas al lenguaje mencionado anteriormente, tales como *stdio.h* y *stdlib.h*.

1.7. Estructura del informe

El presente escrito presenta a continuación conceptos que fueron la base principal en la que se desarrolló la solución del problema, tales como punteros, memoria dinámica, entre otros, con el fin de exponer la investigación realizada, la descripción de la solución, el análisis de estos y las conclusiones.

2. Marco teórico

2.1. Librerías

Entre las herramientas utilizadas para el desarrollo de este laboratorio, se encuentran las librerías de:

- `stdio.h`: Para la entrada, salida de datos y manejo de archivos.
- `stdlib.h`: Utilizada para el manejo de la memoria dinámica
- `string.h`: Utilizada para el manejo de concatenaciones de caracteres.

2.2. MIPS (Procesador)

Con el nombre de MIPS, se le conoce a la familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies, diseñado para optimizar segmentación en unidades de control y facilitar la generación automática de código máquina por parte de los compiladores.

2.3. Registro

Un registro corresponde al hardware que forma parte del procesador y puede contener una determinada cantidad de bits. Estos ofrecen un nivel de memoria más rápido y acotado que la memoria principal, son empleados para controlar las instrucciones de ejecución, manejar el direccionamiento de memoria y proporcionar capacidad aritmética.

2.4. Punteros

Los punteros son elementos bastante utilizados durante el desarrollo del código debido a que permiten trabajar directamente a las direcciones de memoria de las estructuras de datos que tenemos. Se utilizan tanto como para lectura del archivo como para poder almacenar cada elemento de la instrucción en su estructura correspondiente.

2.5. Lectura y escritura de archivos

Para poder realizar la lectura de archivos anexos a los ingresados manualmente por terminal en nuestro código se utilizan funciones específicas llamadas por la librería `stdio.h` las cuales nos facilitan el acceso a éstos.

2.6. Memoria dinámica

Para un óptimo avance en este código se utilizó memoria dinámica durante la inicialización de las estructuras con el fin de que se pueda trabajar con los valores leídos en el archivo de manera más eficiente. Para esto se utilizó la función `malloc()` para poder asignar el espacio propio de cada instrucción y la función `free()` para ir liberando la memoria una vez que ésta ya ha sido utilizada.

2.7. Estructuras

Las estructuras son un tipo de dato utilizado para poder agrupar distintos tipos de información y manejar datos que sería muy complicado de realizar de manera más primitiva. En el desarrollo de este código se utilizan para la implementación de los nodos del árbol ya que así se puede acceder de manera inmediata a la información contenida en éstos.

3. Desarrollo

Para el desarrollo de este laboratorio lo primero que se realiza es una estructura la cual almacene de manera eficiente para un posterior trabajo la instrucción a leer. Para esto se crea la estructura *Move* la cual posee 4 punteros de tipo *char*, los cuales almacenarán específicamente la instrucción, 2 registros, y un valor numérico (el cual solo se emplea para los registros *addi* y *subi*).

```
typedef struct Move {  
    char * instruccion;  
    char * register_1;  
    char * register_2;  
    char * number;  
} Move;
```

Figura 1: Archivo de salida del camino de datos.

Posterior a esto se realiza la lectura de cada línea del archivo de entrada, donde se revisa la primera palabra de la línea; si esta es una instrucción, dependiendo de su tipo, se realiza una lectura de los registro y datos asociados a este, para así asociarlo a una estructura la cual será almacenada posterior a este proceso en un arreglo que contendrá todas las estructuras. Una vez que todas las instrucciones se encuentren almacenadas de manera correcta en el arreglo de estructuras, se prosigue con la evaluación de cada una de estas y su respectiva función y a la declaración de un tablero, el cual consiste en un arreglo de caracteres con un largo de 9, el cual representará el tablero matricial.

En primera instancia se revisan las primeras dos instrucciones y las define como los jugadores (jugador X y jugador O), ambas entregadas por una instrucción de tipo *addi*. En la siguiente línea se verifica que se pida el almacenamiento para las jugadas a insertar. Posterior a estas 3 revisiones principales, se recorre el arreglo leyendo el tipo de instrucción que posee cada una de las instrucciones y según ésta, define qué acción realizará. Antes de realizar la acción, se le realiza la consulta al tablero si la jugada, ya sea insertar o eliminar un símbolo, exista y que a la vez, esta sea propia del jugador que quiere realizar la acción. En el caso que se realice

una jugada no válida, tal como la inserción en alguna casilla no existente o la sobrescritura respecto una jugada del otro jugador, el programa entrega un mensaje que la partida está incompleta y procede al cierre de este.

Al final de la lectura de todas las instrucciones del arreglo, se realiza una comprobación respecto a si hay un ganador (indicando el nombre de este en caso de existir) o si se genera un empate. Para realizar el cálculo de cuántas veces pasa cada instrucción por cada etapa en el camino de datos, se realizan sumas de valor 1 por cada etapa que realiza de manera constante cada instrucción, es decir, se asume que si es una instrucción de tipo *addi* o *subi*, estas pasarán por las 5 etapas del camino de datos, en cambio, una instrucción de tipo *sw* solamente utiliza 4 etapas del camino de datos (se excluye Mem). A modo de finalización, el programa entrega dos documentos, uno llamado *resultado.txt*, el cual posee la respuesta del ganador de la partida en caso de existir y el tablero con las jugadas realizadas, y un documento llamado *Etapas.txt*, el cual entrega la cantidad de veces que cada instrucción pasa por cada etapa del camino de datos del procesador monociclo.

4. Experimentos a realizar

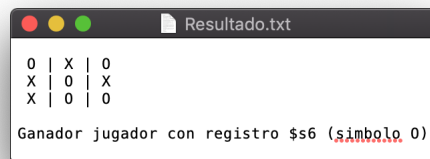
4.1. Resultados obtenidos

A continuación se presentan los archivos de salida resultantes posterior a un caso de prueba.



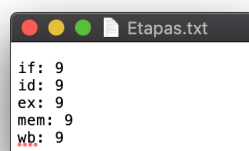
```
TicTacToe git:(master) $ make
gcc main.c programa.c lecturaArchivo.c -o main -Wall
TicTacToe git:(master) $ make run
./main
Ingrese nombre archivo en MIPS (con extension):
> Prueba.txt
Archivos de salida generados correctamente :)
TicTacToe git:(master) $
```

Figura 2: Ejemplo Salida al compilar el código.



```
0 | X | 0
X | 0 | X
X | 0 | 0
Ganador jugador con registro $s6 (símbolo 0)
```

Figura 3: Ejemplo de Salida del resultado del juego.



```
if: 9
id: 9
ex: 9
mem: 9
wb: 9
```

Figura 4: Archivo de salida del camino de datos.

4.2. Análisis de resultados

Se ha logrado que el programa sea capaz de leer las instrucciones y las líneas de control que se presentan en un archivo de texto plano, distinguiendo en este los comentarios y valores inválidos dentro de una jugada, para luego simular su ejecución completando el tablero de manera óptima, en caso que las instrucciones se encuentren insertadas de manera correcta. El programa logra identificar de manera adecuada cuando una jugada corresponde a la victoria de un jugador en particular, un empate o un error de instrucción, el cuál entrega un mensaje de "incompleto".

5. Conclusiones

Gracias a el desarrollo de este laboratorio se puede apreciar de mejor manera el manejo de los conocimientos ya adquiridos durante el desarrollo de las clases, ya que en cada uno de éstos, se profundizaron temas que ahora nos facilitan la compresión del camino de datos y como este es realizado por los distintos tipos de instrucciones que se pueden realizar. Referente a los algoritmos utilizados estos trabajan de manera correcta, gracias a una correcta implementación de la estructura *Move*, el cual facilitó el manejo independiente de cada elemento agrupado en este. Respecto a objetivos planteados en este laboratorio, el programa funciona de manera correcta entregando los valores del camino de datos empleados y el resultado de la partida.

Bibliografía

<https://deblogsyjuegos.wordpress.com> (2015). Tic-tac-toe. [Online] <https://deblogsyjuegos.wordpress.com/2015/11/15/tic-tac-toe/comment-page-1/>.