|  |  |
| --- | --- |
| **UNIVERSIDAD DE SANTIAGO DE CHILE**  **FACULTAD DE INGENIERÍA**  **DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA** | http://guiaweb.usach.cl/sites/default/files/inegrotrz_0.png |

**ORGANIZACIÓN DE COMPUTADORES**

INFORME LABORATORIO Nº1

Nombre Estudiante: Ignacio Zúñiga

Nombre Profesor: Nicolás Hidalgo Nombre Ayudante: Felipe Garay

Santiago – Chile 07 de Mayo, 2015

**Tabla de Contenidos:**

[CAPÍTULO 1. INTRODUCCIÓN 3](#_Toc418804370)

[CAPÍTULO 2. MARCO TEORICO 3](#_Toc418804371)

[CAPÍTULO 3. DESARROLLO 4](#_Toc418804372)

[CAPÍTULO 4. CONCLUSIONES 6](#_Toc418804373)

[CAPÍTULO 5. BIBLIOGRAFÍA 7](#_Toc418804374)

# CAPÍTULO 1. INTRODUCCIÓN

Este informe corresponde al primer laboratorio del curso Organización de Computadores, en el cual se va a tratar sobre la implementación del juego “Snake” en el lenguaje ensamblador MIPS.

Para llevar a cabo la implementación del programa se debe aprender a programar en lenguaje MIPS utilizando el simulador Mars4\_4.

Principalmente, cabe destacar que la técnica utilizada para llevar a cabo las funcionalidades implementadas del programa “Snake”, se utilizó Polling, el cual consiste en leer periódicamente una dirección de memoria y ver si se ha presionado alguna tecla. Se lee la dirección de memoria 0x0000ffff4, y en ella se encuentra el último carácter leído desde el teclado. Lo demás son ciclos y operaciones que se utilizaron para programar.

# CAPÍTULO 2. MARCO TEORICO

MIPS, de las siglas Microprocessor without Interlocked Pipeline Stages, es una ISA (Instruction Set Architecture) de tipo RISC (Reduced Instruction Set Computer), que consta de 32 registros de 32 bits cada uno, donde se pueden encontrar distintos tipos de éstos.

Que sea una arquitectura de tipo RISC significa que posee instrucciones simples que pueden ser ejecutadas en la menor cantidad de ciclos de reloj y que posee menos instrucciones en comparación con los de tipo CISC, pero esto hace que los códigos resulten más largos, aunque el programador tiene acceso a lenguajes de alto nivel, por lo que el impacto no es tan grande.

La convención para los registros en MIPS es usar un nombre de dos caracteres luego de un signo peso. Entre estos registros se usarán $sn para variables, con n desde 0 hasta 7, $tn para temporales, con n desde 0 hasta 7 también, $an para parámetros, con n desde 0 hasta 3, y $v0 y $v1 para retornos.

En el presente laboratorio se plantea programar un juego, en el cual se utilizan ciclos llevados a lenguaje ensamblador, además de otras técnicas como la utilización de un *display* para generar la vista, y la técnica de Polling para obtener los datos leídos desde el teclado. De esta forma se pretende implementar el juego “Snake”, el cual será explicado a continuación,

# CAPÍTULO 3. DESARROLLO

**3.1 FUNCIONES UTILIZADAS**

En la siguiente sección se especificarán las funciones utilizadas, dando una descripción de lo que realiza, la entrada, eventualmente una salida y lo que aporta a la solución del problema.

**3.1.1 Función “main”**

La función “main” básicamente define una variable que será utilizada para poder generar el efecto movimiento, es decir, mover el pixel hacia el siguiente espacio y pintar el bloque que deja como rastro, así da lugar al movimiento. No tiene entrada, sólo es apuntada por el PC, y tampoco tiene salida ya que sólo asigna una variable saved temporary.

**3.1.2 Función “while”**

La función “while” hace que el juego esté en permanente funcionamiento (ya que no se implementaron las restricciones de un eventual fin del juego), posee instrucciones JAL (Jump And Link), es decir, después de saltar a la etiqueta, se devuelve a la instrucción en la que se dio la instrucción para seguir con la próxima. El Link (jr $ra), hace que el Program Counter apunte a la instrucción anterior al salto + 4, es decir, a la siguiente instrucción original. No posee entrada ni salida, ya que sólo está encargada de mantener el juego en permanente funcionamiento haciendo las llamadas a las demás funciones.

**3.1.3 Función “Snake”**

La función “Snake” parte creando un pixel con color hexadecimal, luego ante un eventual movimiento, se reubica este pixel mediante la función SW (Store Word) y se borra el “rastro”, con otro SW se pinta el rastro de color negro para generar el efecto movimiento. El registro base del posicionamiento nuevo del pixel ($s7), es dinámico, ya que mediante la tecla que se recibe, se opera la coordenada dependiendo el tipo de movimiento y ubica el pixel en esta nueva coordenada. La entrada es el display, ya que sobre este se ubica el pixel y el eventual movimiento. La salida es una nueva coordenada del pixel ubicado en el display.

**3.1.4 Función “Punto”**

Se genera un pixel en la posición 2400 del display, se intentó “comer” el punto con un BEQ (hacer que cuando los dos pixeles tuvieran el mismo valor, se borre el punto y se agregue al Snake), además está implementada la función RANDOM, con la que se pretendía generar un nuevo valor para el punto luego de ser devorado. No posee entrada, y su salida corresponde sólo a ubicar un pixel en el tablero.

**3.1.5 Función “Tablero”**

La función Tablero se encarga de pintar las esquinas del display, y permitir el paso del pixel (Snake) hacia el otro lado cuando atraviesa una pared. Básicamente, se le asigna a $t4 el valor hexadecimal del color (llamativo por lo demás), y este es cargado en el display. No posee entrada (sólo el display al igual que las demás funciones), y su salida corresponde a pixeles ubicados en las esquinas del display.

**3.1.6 Función “Polling”**

La función Polling es la encargada de recibir un valor mediante el *Keyboard and Display Simulator* e interpretarla mediante los llamados a las funciones de movimiento. Luego de interpretar un valor por teclado, salta nuevamente a la misma etiqueta para recibir el siguiente carácter, y así continúa sucesivamente. La entrada es un valor desde el teclado, luego la interpreta, para dar salida a un llamado de movimiento.

**3.1.7 Funciones “MovDer”, “MovIzq”, “MovUp”, “MovDown”**

Las funciones mencionadas anteriormente son las encargadas de realizar la acción de movimiento, dada la llamada desde la función Polling. Básicamente, le suma (o resta) la coordenada dado el movimiento que se quiere realizar. Recibe un llamado desde Polling, y ejecuta la operación para dar salida a una nueva coordenada

**3.2 LÓGICA UTILIZADA**

Antes de comenzar a implementar el programa, se planteó la idea de utilizar Polling para recibir datos por el teclado y así poder ejecutar los movimientos. Es por esto que cabe destacar que después de cada función siempre se utilizó un JR $ra, para volver al Link creado por el JAL y de esta forma mantener en permanente utilización de la entrada por teclado. Se pensó que para el movimiento se debía calcular una nueva coordenada, tanto para la cabeza como para la cola, dependiendo del tipo de movimiento que se recibía y borrando todo rastro de la coordenada anterior. Además se había pensado implementar, además de la cabeza, una cola que fuera la encargada de crecer después de comer un punto, esto se pensó con la idea de hacer que se agregara un nuevo pixel en la coordenada anterior a la cola dando así una cantidad considerable de restricciones.

# CAPÍTULO 4. CONCLUSIONES

Para la implementación del programa se tuvo bastantes complicaciones a la hora de implementar en lenguaje de programación MIPS. Al ser este un lenguaje ensamblador, la complejidad de programar solamente con direcciones de memoria es alta y no existe tanta documentación asociada como ocurre con los demás lenguajes.

El problema pensado como algoritmo no resultó catastrófico, sino más bien bastante lógico. La complejidad surgió al momento de pasar ese algoritmo a MIPS, ya que es un lenguaje con otro tipo de complejidad. Después de varias investigaciones y tropiezos, se pudo implementar parte del programa para poder satisfacer al menos algunos puntos a evaluar.

Por otro lado, se aprendió a utilizar la técnica de Polling para la entrada de datos por teclado y a utilizar un display a modo de recurso gráfico. Estos conceptos, si bien fueron otorgados por el ayudante, no están explícitos a la hora de transformarlo en un código más extenso y de mayor complejidad como la que presenta el juego. De una u otra forma, se aprendió a utilizar ciclos para el manejo del Bit Display y del Keyboard, lo que podría ser útil eventualmente a la hora de trabajar con lenguaje ensamblador.

# CAPÍTULO 5. BIBLIOGRAFÍA

Patterson, D. y Hennessy. J., (2009), Computer Organization and Design: The Hardware/Software Interface. Burlington, USA: Elsevier - Morgan Kaufmann.

Apuntes de la plataforma UsachVirtual entregados por el ayudante <http://www.usachvirtual.cl/moodle/file.php/2753/Laboratorio/apuntes/juegos_mips.pdf>

Para la reutilización de funciones, y resolver dudas con respecto a algún ciclo

<http://stackoverflow.com/>