**INFORME LABORATORIO 2**

**ORGANIZACIÓN DE COMPUTADORES**

**LOYOLA VIDAL LUIS ALBERTO**

**VERGARA ARELLANO VASCO ESTEBAN**

Profesor Guía: Erika Rosas

Ayudantes: Ariel Meriño

Felipe Garay

# TABLA DE CONTENIDOS.

[TABLA DE CONTENIDOS. i](#_Toc375782258)

[TABLA DE FIGURAS. ii](#_Toc375782259)

[CAPÍTULO 1: INTRODUCCIÓN 1](#_Toc375782260)

[1.2 DESCRIPCIÓN DEL PROBLEMA 1](#_Toc375782261)

[1.3 SOLUCIÓN PROPUESTA 2](#_Toc375782262)

[1.4 OBJETIVOS Y ALCANCES DEL LABORATORIO 2](#_Toc375782263)

[1.4.1 Objetivo general 2](#_Toc375782264)

[1.5 metodologías y herramientas utilizadas 3](#_Toc375782265)

[1.6 organización del documento 3](#_Toc375782266)

[CAPÍTULO 2: MARCO TEÓRICO 5](#_Toc375782267)

[CAPÍTULO 3: DESARROLLO 7](#_Toc375782268)

[3.1 REPRESENTACIÓN DEL TABLERO 7](#_Toc375782269)

[3.2 GRÁFICA 8](#_Toc375782270)

[3.3 CONTROL DEL JUEGO 9](#_Toc375782271)

[CAPÍTULO 4: CONCLUSIÓN 14](#_Toc375782272)

[CAPÍTULO 5: REFERENCIAS 15](#_Toc375782273)

# TABLA DE FIGURAS.

Figura 3-1: Representación del tablero hexadecimal con una matriz de 2 dimensiones. 7

Figura 3-2: Matriz del tablero con celdas nulas alrededor. 8

Figura 3-4: Muestra los vecinos cercanos y lejanos de una celda en una columna impar y su representación en la matriz del tablero. 12

Figura 3-3: Muestra los vecinos cercanos y lejanos de una celda en una columna par y su representación en la matriz del tablero. 12

# CAPÍTULO 1: INTRODUCCIÓN

**1.1 ANTECEDENTES Y MOTIVACIÓN**

El motivo para el que hacemos este laboratorio es para que aprendamos a desarrollar un juego en MIPS Assembly, es decir un lenguaje ensamblador, que se diferencia bastante de los lenguajes vistos en otros cursos (como por ejemplo C y Java) principalmente porque el procesador almacena una pequeña cantidad de datos en los registros, y el resto está en memoria, por lo que cada vez que se necesita llamar una función se deben guardar los datos que se están usando en memoria usando el Stack Pointer para luego cargarlos cuando la función se termina y así los valores no se pierdan1. Por esta razón es mucho más difícil desarrollar un juego en MIPS que en un lenguaje como C o Java, ya quelos operandos en lenguaje de máquina están restringidos, para el caso de MIPS este posee 32 registros1.

## 1.2 DESCRIPCIÓN DEL PROBLEMA

El problema tratado en este informe corresponde al segundo laboratorio de la asignatura Organización de Computadores, el cual consiste en reconstruir el juego Hexxagon usando MIPS. El juego consiste en un tablero con celdas hexagonales, donde 2 jugadores intentan convertir las fichas del otro jugador en piezas propias y dominar el tablero. Un jugador puede mover cualquiera de sus piezas a cualquier celda vecina considerada como vecino cercano, o sea, que está en contacto con ésta, o a cualquier celda considerada como vecino lejano, o sea, los vecinos cercanos de los vecinos cercanos.

Si salta a los vecinos cercanos, la pieza de duplica, mientras que si se mueve a un vecino lejano, la pieza abandona su posición actual para ocupar un nuevo lugar en el tablero.

Cada vez que una pieza se mueve convierte a todas las piezas a su alrededor en suyas.

## 1.3 SOLUCIÓN PROPUESTA

La solución propuesta, acorde con los motivos por los que hacemos este laboratorio, será un programa en MIPS con la implementación del juego, basándose en las reglas ya explicadas.

Para ello se debe implementar un conjunto de funciones que pueda manejar la parte gráfica para poder mostrar dibujos.

También hay que representar el tablero dentro del juego, por lo que se usa un arreglo de largo x\*y (con “x” e “y” tamaño del tablero). Esto se explica con mayor detalle en el capítulo 3.

## 1.4 OBJETIVOS Y ALCANCES DEL LABORATORIO

### 1.4.1 Objetivo general

El objetivo general es lograr implementar el juego Hexxagon en MIPS, y que este funcione con la totalidad de sus reglas y sea cómodo para los usuarios.

**1.4.2 Objetivos específicos**

Poder realizar un programa elaborado que requiera mejores habilidades de programación, en comparación con los primeros programas realizados en clases como forma introductoria al lenguaje MIPS.

**1.4.3 Alcances**

Será parte del trabajo de este laboratorio desarrollar algoritmos para luego implementarlos en MIPS y desarrollar un manejo de datos mucho más elaborado que los presentes en el primer laboratorio1.

## 1.5 metodologías y herramientas utilizadas

Acorde a los objetivos del laboratorio el algoritmo se programa utilizando MIPS, por lo que se usa el software MARS para su implementación, además se requiere de la interfaz gráfica que ofrece MARS, para mostrar las imágenes del juego.

## 1.6 organización del documento

En este informe primero se detallan los métodos y las herramientas utilizadas durante el laboratorio, así como los objetivos y los conceptos clave de este. Luego se presenta el marco teórico de este informe. Después se muestra el implementado en MIPS. Finalmente se obtienen las conclusiones del laboratorio y se adjunta un manual de usuario con la implementación del algoritmo.

# CAPÍTULO 2: MARCO TEÓRICO

MIPS (siglas de *Microprocessor without Interlocked Pipeline Stages*) se conoce a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

MIPS es derivada de una arquitectura diseñada en Stanford con propósitos educacionales. Las primeras arquitecturas MIPS fueron implementadas en 32 bits (generalmente rutas de datos y registros de 32 bits de ancho), si bien versiones posteriores fueron implementadas en 64 bits. Existen cinco revisiones compatibles hacia atrás del conjunto de instrucciones del MIPS, llamadas MIPS I, MIPS II, MIPS III, MIPS IV y MIPS 32/64. En la última de ellas, se define un conjunto de control de registros. Además están disponibles varias "extensiones", tales como la MIPS-3D, consistente en un simple conjunto de instrucciones SIMD en coma flotante dedicadas a tareas 3D comunes, la MDMX(MaDMaX) compuesta por un conjunto más extenso de instrucciones SIMD enteras que utilizan los registros de coma flotante de 64 bits, la MIPS16 que añade compresión al flujo de instrucciones para hacer que los programas ocupen menos espacio (presuntamente como respuesta a la tecnología de compresión Thumb de la arquitectura ARM) o la reciente MIPS MT que añade funcionalidades multithreading similares a la tecnología HyperThreading de los procesadores Intel Pentium 4. 1

Cada instrucción aritmética de MIPS debe tener exactamente 3 variables. MIPS tiene 32 registros de 32 bits, a los cuales cada uno de estos se les llama *palabras*. Los operados de las instrucciones aritméticas de MIPS deben ser elegidos entre alguno de los 32 registros.

# CAPÍTULO 3: DESARROLLO

Este capítulo está dividido en varias secciones: representación del tablero, mostrar imágenes y control del juego.

**3.1 REPRESENTACIÓN DEL TABLERO**

Para representar la matriz hexagonal se trata como si fuera una matriz de 2 dimensiones, donde las columnas pares están desfasadas una cierta distancia hacia abajo.

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

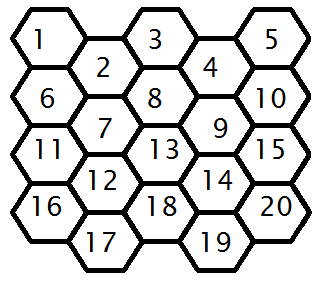


Figura 3-1: Representación del tablero hexadecimal con una matriz de 2 dimensiones.

A cada celda en la matriz hexagonal, le corresponde una celda de la matriz de 2 dimensiones.

Para saber cuáles celdas corresponden a los vecinos, se usa un algoritmo explicado en la sección de algoritmos.

La matriz puede tener 4 valores, 0 para una celda nula o inexistente, 1 para una celda vacía, 2 y 3 para celdas que tengan una ficha del jugador 1 y 2 respectivamente.

Como una celda tiene un grupo de vecinos cercanos y un grupo de vecinos lejanos, que siempre están a máximo 1 y 2 celdas de distancia, respectivamente, la matriz tiene 2 bordes de celdas inexistentes para poder simplificar el algoritmo de buscar de vecinos, así en el caso de que se busque los vecinos de una celda cercana a los bordes, no hay problemas de acceder a espacios inexistente ya que estos contienen celdas nulas.

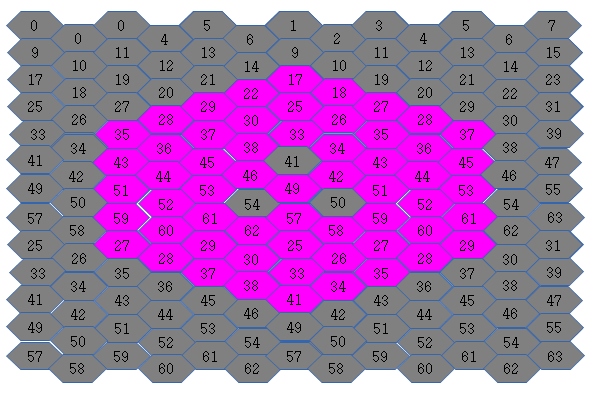


Figura 3-2: Matriz del tablero con celdas nulas alrededor.

**3.2 GRÁFICA**

Para poder mostrar imágenes dentro del programa se trabaja con el modulo “Bitmap Display” del simulador MARS. Para utilizar este *display* se requiere una etiqueta al principio de la sección *.data* que contenga x\*y palabras (donde “x” e “y” son las dimensiones del display.)

Se implementaron funciones que permiten dibujar una imagen en una coordenada X, Y del display. La imagen debe estar guardada en una etiqueta donde cada pixel es una palabra en hexadecimal. Utilizando el hexadecimal 0x01000000 como salto de línea, y 2 saltos de líneas seguidos para representar el final de la imagen y el valor 0 como pixel vacio, el cual no debe ser pintado en el display.

Para facilitar el traspaso de imágenes a este formato personalizado, se utilizó el programa “ImageRGB.jar”3.

**3.3 CONTROL DEL JUEGO**

Para efectos del control del juego, primero se desarrolla la función INDEX para poder calcular la dirección de una posición en la matriz o en el display usando sus coordenadas, esta función se usará múltiples veces a los largo del programa para poder mostrar celdas seleccionadas o para transformar la matriz. Luego se debe implementar la función para poder jugar un turno, que debe recibir como parámetro el jugador al que le toca jugar. Esta función debe permitirle al usuario elegir una ficha para poder moverla o desplazarla. Para ello cuando el usuario se posiciona en una de sus fichas debe mostrarle las celdas en las cuales se puede duplicar, los vecinos cercanos y las celdas a las que se puede desplazar, los vecinos lejanos. Para obtener estas celtas se desarrollan 2 funciones, una que obtenga los vecinos cercanos y otra que obtenga los vecinos lejanos. Junto a estas funciones se debe desarrollar una función que filtre el arreglo de vecinos obtenido dejando solo el tipo de fichas que necesitamos. Esto es para dejar solamente las celdas vacías para el usuario elija donde moverse, ya que no puede moverse a una celda que este ocupada. Esta función será muy útil para la transformación, ya que se pueden obtener las celdas vecinas ocupadas por fichas enemigas para luego transformarlas. Después que el usuario elija que ficha usará y si elegirá la opción de duplicarse o desplazarse, se deben desarrollar las 2 funciones que ejecutaran el conjunto de pasos asociados a estas opciones, para evitar complicaciones el programa no le mostrara como opciones al usuario aquellas fichas que le pertenezcan a él pero no tengan espacio para moverse, es decir, que no tenga vecinos cercanos ni vecinos lejanos. Para ambos casos al momento de elegir una opción se le permitirá al usuario recorrer los vecinos asociados a esa opción para que elija uno y haga su jugada. En caso que elija una opción que tenga 0 vecinos asociados, por ejemplo elegir duplicarse sin vecinos cercanos, el programa volverá a preguntar hasta que el usuario elija una opción válida, para el caso del ejemplo sería que elija desplazarse. Cuando el usuario elija una celda, la función asociada transformará dicha celda y todas las fichas del jugador oponente en celdas adyacentes (los vecinos cercanos de dicha celda) en fichas del jugador. En caso de haber escogido desplazarse la función deberá eliminar la celda original que se movió y transformarla en una celda vacía. En resumen se deben hacer 2 funciones, una para en caso que el usuario decida usar uno de los vecinos cercanos y otra en caso que quiera usar los vecinos lejanos y se debe ejecutar una de ellas dependiendo de cuales elija usar el usuario.

Una vez terminada la función que ejecuta un turno, se necesita buscar si aún existen celdas vacías en el tablero, en caso que no las haya se debe terminar el juego y elegir un ganador. Para ello se implementará una función que recorra la matriz y en caso de encontrar una celda vacía retorne 1(verdadero) y en caso que no la encuentre retorne 0(falso). Luego se debe desarrollar una función que verifique que el siguiente jugador pueda jugar, es decir si alguna de sus fichas tiene celdas vacías en sus vecinos cercanos y/o vecinos lejanos asociados. En caso que no los tenga la función debe retornar 0(falso), al encontrar alguna celda vacía en uno de los vecinos de una ficha del jugador que va a jugar, la función debe retornar 1(verdadero).

Por último se desarrolla la función Game Loop que repite los turnos alternando los jugadores, y al final de cada turno primero refresca el tablero con los nuevos datos, y luego revisa si existen celdas vacías, para luego hacer un Branch, si el retorno de la función es 0 salta la función que compara los puntajes de ambos jugadores para elegir un ganador. Después se llama a la función que verifica si el siguiente jugador puede jugar, y se hace un Branch con el retorno de dicha función, que en caso de ser 0 termina el juego y le da la victoria al último jugador que jugó.

**3.4 Algoritmos**

3.4.1. Mostrar

*Mostrar* recibe como argumento la etiqueta de una imagen a mostrar, la dirección del display donde mostrar, las coordenadas X e Y.

Primero toma la dirección del display y realiza el offset correspondiente a las coordenadas X e Y, este offset se calcula como Y\*512\*4 + X\*4. En la nueva posición del display se copia y pega cada palabra de la etiqueta de imagen al display, si la palabra a copiar tiene el valor 0, entonces no se copia y se pasa al a siguiente palabra. Cuando se detecta un salto de línea se agregan 512\*4 por el número de fila al valor del display original para saltar las líneas correspondientes. Cuando se detectan 2 saltos de línea seguidos la función se detiene.

3.4.2. Calcular Vecinos.

Para encontrar los vecinos de una celda en específico se utiliza el siguiente criterio:

Los vecinos cercanos de la celda (i,j) son (i,j+1),(i,j-1),(i+1,j),(i-1,j), que corresponden a los vecinos básicos (arriba, abajo, izquierda y derecha), los otros 2 vecinos depende si la celda está en una columna par. (j es par) entonces los otros 2 vecinos son (i-1,j-1),(i-1,j+1), si j es impar entonces (i+1,j-1),(i+1,j+1).

Para los vecinos más lejanos a (i,j) son : (i-1,j-2), (i,j-2), (i+1,j-2), (i-1,j+2), (i,j+2), (i+1,j+2). Si j es par, los demás vecinos son (i-2,j-1), (i-2,j), (i-2,j+1), (i+1,j-1), (i+1,j+1), (i+2,j).

Si j es impar entonces los demás vecinos lejanos son (i+2,j-1), (i+2,j), (i+2,j+1), (i-1,j-1), (i-1,j+1), (i-2,j).

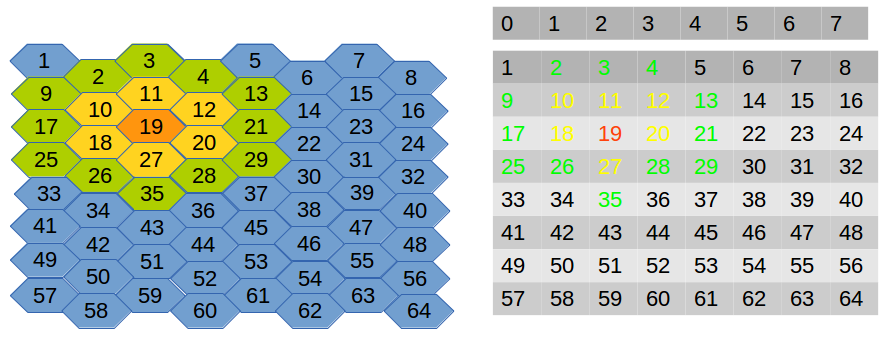
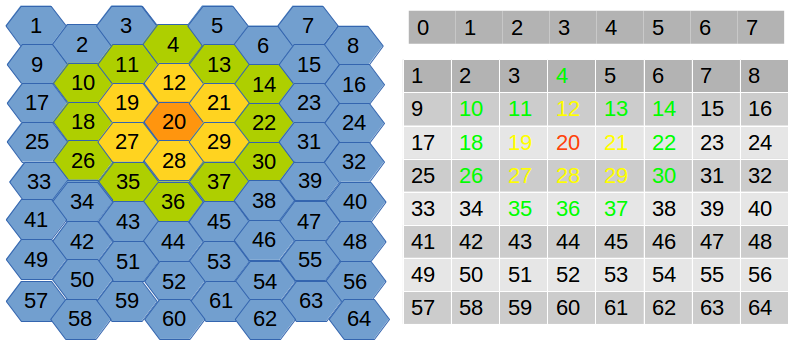


Figura 3-4: Muestra los vecinos cercanos y lejanos de una celda en una columna impar y su representación en la matriz del tablero.

Figura 3-3: Muestra los vecinos cercanos y lejanos de una celda en una columna par y su representación en la matriz del tablero.

# CAPÍTULO 4: CONCLUSIÓN

Se puede concluir que se cumplen los objetivos del laboratorio, ya que se logró implementar el juego pedido, entregándolo con todas sus reglas implementadas. Podemos darnos cuenta que es muy distinto programar en un lenguaje de bajo nivel que a los lenguajes de como C y Java a los que estamos acostumbrados, ya que instrucciones que en dichos lenguajes son mucho más simples, en MIPS requieren un número mucho mayor de líneas para implementarse. Pero como hemos visto en el transcurso del curso el tener un número más reducido de instrucciones hace que su diseño sea más simple y barato, por lo que aumenta su eficiencia en ese sentido. Por ello es importante que hayamos cumplido el objetivo del laboratorio, ya que poder trabajar con un lenguaje de bajo nivel puede llevarnos a un ahorro económico en el futuro, que es una de los objetivos más importantes de los ingenieros, optimizar los recursos.

# CAPÍTULO 5: REFERENCIAS

[1] Datos específicos de MIPS [en línea]. Recuperado el día 25 de diciembre del 2013, de <http://www.usachvirtual.cl/moodle/file.php/2753/Clases/Clase3.pdf>

[2] Información general de MIPS [en línea]. Recuperado el día 19 de noviembre del 2013, de: <http://es.wikipedia.org/wiki/MIPS_(procesador)>

[3] ImageRGB.jar, programa de código abierto desarrollado por Luis Loyola V. para ayudar al desarrollo de este laboratorio.

..