**Instituto Tecnológico y de Estudios Superiores de Occidente**

**Organización y Arquitectura de Computadoras**

**Forma

Descripción generada automáticamente**

**Práctica 2**

***Juego de la serpiente en RIPES***

**Profesor: Juan Pablo Ibarra Esparza**

**Integrantes:**

**Zamora Vargas Luis Santiago 735430**

**Abraham de León Gutiérrez 739343**

**SNAKE**

Con esta práctica se planea implementar un juego de snake en el simulador de RIPES, el cual emula la arquitectura de RISC-V.

**¿Qué hace?**

El código es bastante sencillo, se divide en funciones para poder organizar mejor todo:

1. Primero se limpia el tablero apagando todos los leds
2. Posteriormente se prenden los dos leds de la serpiente, además de que hacemos que head y tail apunten a los inicios del arreglo, donde guardamos las direcciones de memoria de estas partes.
3. Imprimimos los límites marcados en un color beige, se hacen dos ciclos donde cada ciclo llena la width y heigth del color.
4. Se genera una manzana de una forma pseudorandom, ya que no se podía usar time.h, se hace un ciclo while donde se garantiza que no se genere una manzana en el borde o sobre la serpiente.
5. Se comienza con el juego en un while determinado por la variable game, que se convierte en falso cuando hay colisión.
6. Se comienza a mover la serpiente con la lógica de un array circular donde la head y la tail van avanzando, además se verifican colisiones y si se come la manzana entramos a eat apple.
7. En eatapple, hacemos que la manzana cambie al color de la head y que en este caso no se haga nada con la tail, con eso crece la serpiente.
8. Para detectar los dpads, simeplemente manejamos dx y dy, (0, -1) arriba, (0, 1) abajo, (-1, 0) izquierda, (1, 0), derecha.
9. Al finalizar el juego limpiamos de nuevo el tablero.

*#include* "ripes\_system.h"

*#include* <stdio.h>

*#include* <stdlib.h>

*#define* SW0 (0x01)

*#define* SW1 (0x02)

*#define* SW2 (0x04)

*#define* SW3 (0x08)

*#define* SW4 (0x10)

*#define* SW5 (0x20)

*#define* SW6 (0x40)

*#define* SW7 (0x80)

*#define* LED\_MATRIX\_0\_SIZE   (0xdac)

*#define* LED\_MATRIX\_0\_WIDTH  (0x23)

*#define* LED\_MATRIX\_0\_HEIGHT (0x19)

volatile unsigned int \* led\_base = (volatile unsigned int \*)LED\_MATRIX\_0\_BASE;

volatile unsigned int \* d\_pad\_up = (volatile unsigned int \*)D\_PAD\_0\_UP;

volatile unsigned int \* d\_pad\_do = (volatile unsigned int \*)D\_PAD\_0\_DOWN;

volatile unsigned int \* d\_pad\_le = (volatile unsigned int \*)D\_PAD\_0\_LEFT;

volatile unsigned int \* d\_pad\_ri = (volatile unsigned int \*)D\_PAD\_0\_RIGHT;

*#define* MAX\_SNAKE\_SIZE (LED\_MATRIX\_0\_WIDTH \* LED\_MATRIX\_0\_HEIGHT)

int game = 1;

int head = 1;

int tail = 0;

int randcounter = 50;

int size = 2;

int new\_head\_index;

int dx = 0;

int dy = 1;

volatile unsigned int snakeLEDs[MAX\_SNAKE\_SIZE];

void initSnake() {

    int st1 = (LED\_MATRIX\_0\_WIDTH+2);

    int st2 = (LED\_MATRIX\_0\_WIDTH\*2+2);

    \*(led\_base+st1) = 0x00FF00;

    \*(led\_base+st2) = 0x00FF00;

    snakeLEDs[tail] = st1;

    snakeLEDs[head] = st2;

}

int moveSnake() {

*if* (dy == 1) {

        new\_head\_index = (snakeLEDs[head] + LED\_MATRIX\_0\_WIDTH);

    } *else* *if* (dy == -1) {

        new\_head\_index = (snakeLEDs[head] - LED\_MATRIX\_0\_WIDTH);

    } *else* *if* (dx == 1) {

        new\_head\_index = (snakeLEDs[head] + 1);

    } *else* *if* (dx == -1) {

        new\_head\_index = (snakeLEDs[head] -1);

    }

*//The game ends when in contact with the colors of the edges of the map or the snake itself*

*if* (\*(led\_base+new\_head\_index)==0xF9F6B9 || \*(led\_base+new\_head\_index)==0x00FF00 ) {

*return* 0;

    } *else* *if* (\*(led\_base+new\_head\_index)==0xFF0000) {

        eatApple();

        generateApple();

*return* 1;

    }

    head = (head + 1) % MAX\_SNAKE\_SIZE;

    snakeLEDs[head] = new\_head\_index;

    \*(led\_base + new\_head\_index) = 0x00FF00;

    \*(led\_base + snakeLEDs[tail]) = 0x0;

    tail = (tail + 1) % MAX\_SNAKE\_SIZE;

*return* 1;

}

void printlimits(int *color*) {

*for* (int i = 0; i < LED\_MATRIX\_0\_WIDTH+1; i++)

    {

        \*(led\_base+i)=*color*;

        \*(led\_base+(LED\_MATRIX\_0\_HEIGHT-1) \* LED\_MATRIX\_0\_WIDTH + i) = *color*;

    }

*for* (int i = 0; i < LED\_MATRIX\_0\_HEIGHT; i++) {

        \*(led\_base + i \* LED\_MATRIX\_0\_WIDTH) = *color*;

        \*(led\_base + i \* LED\_MATRIX\_0\_WIDTH-1) = *color*;

    }

}

void cleanBoard() {

*for* (int i = 0; i < MAX\_SNAKE\_SIZE; i++) {

        \*(led\_base+i) = 0x0;

    }

}

void changeDirection(int *dex*, int *dey*) {

    dx = *dex*;

    dy = *dey*;

}

void eatApple() {

    head = (head + 1) % MAX\_SNAKE\_SIZE;

    snakeLEDs[head] = new\_head\_index;

    \*(led\_base + new\_head\_index) = 0x00FF00;

}

void generateApple() {

    int flag = 1;

*while* (flag){

        randcounter += 5;

        srand(randcounter);

        int random\_x = 2 + rand() % (LED\_MATRIX\_0\_WIDTH -4);

        int random\_y = 2 + rand() % (LED\_MATRIX\_0\_HEIGHT -4);

        int position = random\_y \* LED\_MATRIX\_0\_WIDTH + random\_x;

*if* (\*(led\_base+position) != 0xF9F6B9 || \*(led\_base+position) != 0x00FF00) {

            \*(led\_base+position) = 0xFF0000;

            flag = 0;

        }

    }

}

void main() {

    cleanBoard();

    initSnake();

    printlimits(0xF9F6B9);

    generateApple();

*while* (game){

        game = moveSnake();

*for* (int i = 0; i < 10000; i++) {

        }

*if*(\*d\_pad\_up == 1) changeDirection(0, -1);

*if*(\*d\_pad\_do == 1) changeDirection(0, 1);

*if*(\*d\_pad\_le == 1) changeDirection(-1, 0);

*if*(\*d\_pad\_ri == 1) changeDirection(1, 0);

    }

    cleanBoard();

}

**DIAGRAMA DE FLUJO**

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza mediaMain()

initSnake()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

moveSnake()

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

printLimits()

Interfaz de usuario gráfica

Descripción generada automáticamente

cleanBoard()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

changeDirection()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteeatApple()

generateApple()

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

**CACHÉ**

En base a las siguientes conﬁguraciones de caché, ¿Cuál es el hit rate de cada conﬁguración?

● Mapeo directo 4 líneas

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

HR: 0.6628

● Asociativa con 2 conjuntos (lines) y 2 vías (ways)

Interfaz de usuario gráfica

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

HR: 0.808

● Totalmente asociativa con 4 vías (ways)

Interfaz de usuario gráfica

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

HR: 0.8112

● Mapeo directo 16 líneas

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Gráfico, Gráfico de líneas

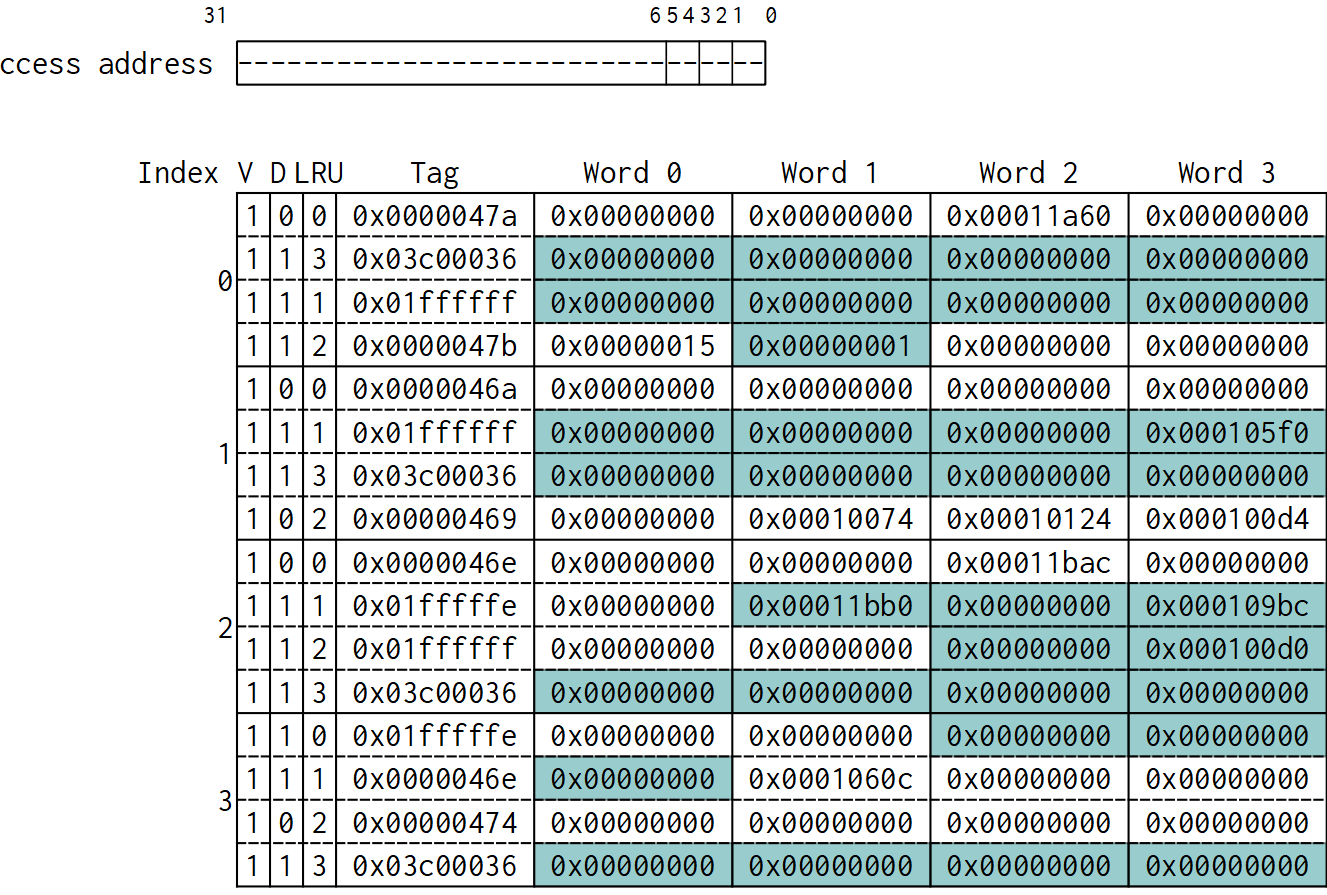
Descripción generada automáticamente

HR: 0.8101

● Asociativa con 4 conjuntos (lines) y 4 vías (ways)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente



Gráfico

Descripción generada automáticamente

HR: 0.857

● Totalmente asociativa con 16 vías

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

HR: 0.8566

¿Cuál de las siguientes conﬁguraciones tiene el mejor hit rate?

La asociativa con 4 conjuntos (lines) y 4 vías (ways) y por poco la totalmente asociativa con 16 vías.