



**FACULTAD  
DE INGENIERIA**

Universidad de Buenos Aires

## [66.20] ORGANIZACIÓN DE COMPUTADORAS

### TRABAJO PRÁCTICO 1

1<sup>ER</sup> CUATRIMESTRE 2020

---

# Conjunto de instrucciones MIPS

---

#### AUTORES

Husain Cerruti, Ignacio Santiago. - #90.117

<ihusain@fi.uba.ar>

Rodríguez Florencia - #100.033

<florrr1997@gmail.com>

Torres Dalmas, Nicolás - #98.439

<ntorresdalmas@gmail.com>

#### CÁTEDRA

Dr. Ing. Hamkalo, José Luis.

#### CURSO

Dr. Ing. Juan Heguiabehere

Ing. Tomás Niño Kehoe

Ing. Matías Stahl

#### FECHA DE ENTREGA

28 de mayo de 2020

#### FECHA DE APROBACIÓN

#### CALIFICACIÓN

#### FIRMA DE APROBACIÓN

# Índice

<b>1. Objetivos</b>	<b>3</b>
<b>2. Diseño e implementación del programa</b>	<b>3</b>
2.1. mod() . . . . .	6
2.2. vecinos() . . . . .	8
<b>3. Compilación del programa y portabilidad</b>	<b>13</b>
<b>4. Casos de prueba</b>	<b>15</b>
4.1. Prueba de fuga de memoria . . . . .	15
4.2. Pruebas de errores en arumentos al programa . . . . .	15
4.3. Pruebas en las opciones de programa . . . . .	16
4.4. Pruebas de generación de imágenes . . . . .	19
<b>5. Pruebas de tiempo de ejecución</b>	<b>19</b>
5.1. Glider . . . . .	19
5.2. Sapo . . . . .	19
5.3. Pento . . . . .	20
<b>6. Herramientas de hardware y software utilizadas</b>	<b>20</b>
<b>7. Conclusiones</b>	<b>21</b>
<b>Referencias</b>	<b>22</b>
<b>A. Enunciado del trabajo práctico</b>	<b>23</b>
<b>B. Makefile</b>	<b>30</b>
B.0.1. makefile . . . . .	30
<b>C. Tests</b>	<b>31</b>
C.0.1. run_tests.sh . . . . .	31
<b>D. Header files</b>	<b>37</b>
D.0.1. cmd_line_parser.h . . . . .	37
D.0.2. mod.h . . . . .	38
D.0.3. params_t.h . . . . .	39
D.0.4. tablero.h . . . . .	40
D.0.5. vecinos.h . . . . .	41
<b>E. Código fuente</b>	<b>42</b>
E.0.1. cmd_line_parser.c . . . . .	42
E.0.2. conway.c . . . . .	47
E.0.3. mod.c . . . . .	49

E.0.4.	mod.S . . . . .	50
E.0.5.	tablero.c . . . . .	52
E.0.6.	vecinos.c . . . . .	56
E.0.7.	vecinos.S . . . . .	57

## 1. Objetivos

El presente trabajo tiene los siguientes objetivos:

- Programar el autómata celular diseñado por Conway, conocido como “Juego de la Vida”.
- Utilizar el lenguaje *Assembly* MIPS32 para implementar el cálculo del estado de vecinos y algunas funciones auxiliares.
- Utilizar la ABI presentada por la cátedra para el desarrollo de los códigos en *Assembly*.
- Utilizar el lenguaje de programación C para desarrollar lo necesario para el procesamiento de las opciones de línea de comandos, apertura y cierre de archivos, y reporte de errores, e invocar el código en *Assembly*.
- Compilar el programa en el sistema operativo Linux Debian utilizando una máquina con arquitectura MIPS32.
- Realizar pruebas de caja negra para verificar que el programa está funcionando de manera correcta.
- Realizar mediciones de tiempos de ejecución, y su comparación entre implementaciones en C y ASM MIPS32.

## 2. Diseño e implementación del programa

Se diseñó un programa en lenguaje ANSI C que implementa el juego de la vida de Conway. El programa se estructura de la siguiente manera:

- Análisis gramatical de la línea de comandos: se analizan las opciones ingresadas por la línea de comandos haciendo uso de la función `parse_cmd_line()`. La misma se encarga de inicializar una estructura del tipo `params_t` utilizada para almacenar las opciones que ingresó el cliente, y cuya definición es

```
1      typedef struct params_t
2      {
3          int i;
4          int N;
5          int M;
6          FILE *inputStream;
7          FILE *outputStream;
8          char *prefix;
9      } params_t;
```

Además, hace uso de la función `getopt_long()` de la biblioteca `getopt.h`. Dicha función provee una forma simple de procesar cada opción que es leída, extrayendo los argumentos de cada una. En caso de que no se encuentre alguna opción, se utiliza su valor por defecto según las especificaciones del trabajo.

Como opcional, se puede utilizar el argumento `-o` para suprimir la generación de imágenes, mostrándose la evolución del juego en la consola.

- Validación de opciones: a medida que se va analizando cada opción de la línea de comandos, se valida cada una de ellas utilizando las funciones

```
1 void option_iterations();
2 void option_rows_size();
3 void option_columns_size();
4 output_state validate_stream_name();
5 output_state option_input_file();
6 output_state option_output_prefix();
```

Además, las mismas realizan la correcta inicialización de las diferentes variables dentro de la estructura `params_t` descrita en el punto anterior, o en caso de que el usuario ingresó las opciones de ayuda e indicación de versión del programa, se imprime por el flujo `stderr` dicha información.

En caso de que se encuentre algún error en el argumento de alguna de las opciones, el usuario es informado por el flujo `stderr`, y se aborta la ejecución del programa utilizando la función `exit()`. Para ello, se creó un tipo enumerativo para simplificar el manejo de errores, definiendo los códigos que pueden devolver las funciones desarrolladas:

```
1 typedef enum output_states_
2 {
3     outOK,
4     outERROR
5 } output_state;
```

- Creación y carga del tablero: Para representar el tablero, se utiliza la siguiente estructura

```
1 typedef struct tablero
2 {
3     unsigned char *tabla;
4     int l;
5     int h;
6 } tablero_t;
```

La carga de la condición inicial del tablero se realiza a través de la función `tablero_cargar_tablero()`, leyendo el archivo de estado inicial indicado por línea de comando, como se muestra a continuación:

```
5 3
5 4
5 5
3 4
4 5
```

siendo cada línea la posición de una célula viva, indicando fila y columna espaciadas entre sí. Como se observa en `tablero_t`, se decidió representar la tabla que contiene estas células como una lista de cadenas de caracteres, ya que de esta forma se facilita el manejo de índices del arreglo en el código assembler a programar posteriormente.

Por último, la creación del tablero se realiza en la función `tablero_crear()` para establecer el estado inicial de la tabla, donde se realiza el pedido de memoria al sistema operativo para poder almacenar la misma.

```
1      tablero->tabla = (unsigned char *)calloc(height * ←
      length, sizeof(unsigned char));
```

- **Generación de imágenes:** a medida que avanza el estado del tablero, se generan imágenes en formato PBM o se imprimen por consola, dependiendo la opción seleccionada por el usuario. La función encargada de dicho procesamiento es `tablero_imprimir()`, y en el caso de generación de archivos de imágenes, se utiliza la función de biblioteca `fopen()` con el flag `wb`.

Para representar los estados de las células se utilizan los colores negro (muerto) y blanco (vivo). Estos se van determinando a medida que se lee el estado actual de la tabla, y están definidos en las variables

```
1      /* Negro */
2      static unsigned char color_vivo[3] = {0, 0, 0};
3      /* Blanco */
4      static unsigned char color_muerto[3] = {255, 255, ←
      255};
```

Otro aspecto importante del programa es la posibilidad de realizar un aumento de tamaño (zoom) a cada célula. El mismo se configura desde la macro `ZOOM_ARCHIVO` en el archivo `tablero.h`, donde la misma representa cuantos “pixels x pixels” utiliza cada célula.

- **Cálculo de vecinos:** en el cálculo de la cantidad de vecinos vivos, se hizo uso del operador módulo para evitar los problemas de contorno (ya que se trata de una matriz

toroidal). Debido a que el operador módulo% no funcionaba como se espera cuando se utilizan números negativos, se desarrolló una función que solucionara estos casos de borde. La misma se encuentra implementada de la siguiente forma:

```

1      int mod(int x, int m)
2      {
3          int r = x % m;
4          return r < 0 ? r + m : r;
5      }

```

- **Terminación del programa:** una vez finalizado el procesamiento, se liberan los recursos utilizados (cierre de archivos y liberación de memoria dinámica), y se devuelve el control al sistema operativo.

El código fuente desarrollado en ASM MIPS32 y en C se encuentra en el apéndice (E) con sus respectivos *headers* en (D).

A continuación se describen las secciones importantes de las funciones desarrolladas y se muestra para cada una de ellas el stack que deben crear según la convención descrita en [5]. Se recuerda que se debe agregar padding para alinear el stack a 8 bytes como se requiere en la convención de llamadas a función, por lo que hay algunos stacks que incorporan las variables con nombre PADDING\_X para mostrar esto. Además, se define su tamaño dependiendo de algunas constantes que se debieron utilizar para parametrizar el problema, por lo que se los calculó en función de ellas para una mejor comprensión.

Se programaron las funciones mod() y vecinos() en ASM MIPS32. A pesar de que la primera no es solicitada por el enunciado, nos pareció interesante hacerlo ya que es una función hoja, y también para hacer uso del operador módulo en código ASM. Por prolijidad, y para no proveer archivos fuente demasiado extensos, se separaron sus declaraciones en 2 archivos .h. Las declaraciones son

```

1      extern int mod(int x, int m);
2      extern unsigned int vecinos(unsigned char *a, unsigned int ←
          i, unsigned int j, unsigned int M, unsigned int N);

```

para indicarle al compilador que dichos símbolos están definidos en otros archivos (los .S).

Por otro lado, como ventaja fue fundamental dicha división ya que permitió realizar depuraciones más simples cuando tuvimos diferentes errores en la codificación del programa.

## 2.1. mod()

La función mod() realiza la operación

```

1      int r = x % m;
2      return r < 0 ? r + m : r;

```

El stack que crea la función se muestra en la figura 1. El código se encuentra en el apéndice E.0.4. En particular, ver las directivas en las primeras líneas donde se define el tamaño del stack

```

1  # Local and Temporary Area (LTA).
2  #define r 0
3  #define PADDING_LTA_0 r + 4
4
5  # Saved-registers area (SRA).
6  #define GP PADDING_LTA_0 + 4
7  #define FP GP + 4
8
9  # Caller ABA.
10 #define x FP + 4
11 #define m x + 4
12
13 #define STACK_SIZE FP + 4

```

Además, al ser una función hoja, no hace falta guardar el registro ra, ni tampoco crear la *Argument Building Area*.

	Local and Temporary Area (LTA)
0	r
4	PADDING_LTA_0
	Saved-registers area (SRA)
8	GP
12	FP
	Stack caller - caller's ABA
16	x
20	m

Tabla 1: Diseño del stack de la función `mod()`. La primer columna se corresponde con el offset en bytes respecto del frame pointer. Se muestra parte del stack del caller, ya que es donde la callee debe guardar los argumentos por convención.

El tamaño del stack queda definido por la directiva

```

1  #define STACK_SIZE FP + 4

```

y toma el valor de 16 bytes.

Durante el preámbulo de la función hay que crear el stack y guardar los registros necesarios, como se muestra en el siguiente fragmento de código



```

1      subu    sp, sp, STACK_SIZE
2      sw      fp, FP(sp)
3      .cprestore GP # Equivalent to      sw      gp, GP(sp).
4      move    fp, sp

```

Debido a que la función recibe dos argumentos por los registros a0 y a1, hay que guardar dichos valores en el stack del caller según la convención utilizada

```

1      sw      a0, x(fp)
2      sw      a1, m(fp)

```

Al salir de la función, se restauran dichos registros, utilizando las siguientes instrucciones, donde se nota que el valor de retorno es devuelto a través del registro v0:

```

1      add     v0, t0, a1
2      # -----
3      exit_function:
4      # Stack frame unwinding.
5      lw      fp, FP(sp)
6      lw      gp, GP(sp)
7      addu    sp, sp, STACK_SIZE
8      jr      ra

```

Es una función de pocas líneas de código, donde la operación “resto” se realiza en la instrucción

```

1      remu    t0, a0, a1

```

## 2.2. vecinos()

La función vecinos() es la siguiente

```

1      unsigned int vecinos(unsigned char *a, unsigned int i,
2      unsigned int j, unsigned int M,
3      unsigned int N)
4      {
5      int contador = 0;
6      int c1 = 0;
7      int c2 = 0;
8      int f = 0;
9      int c = 0;
10
11     for (c1 = -1; c1 <= 1; ++c1)
12     {

```

```

13     f = mod(i + c1, N);
14
15     for (c2 = -1; c2 <= 1; ++c2)
16     {
17         if (c1 == 0 && c2 == 0)
18         {
19             continue;
20         }
21         c = mod(j + c2, M);
22         if (a[f * M + c] == 1)
23         {
24             contador++;
25         }
26     }
27 }
28 return contador;
29 }

```

El stack que crea la función se muestra en la figura 2, y el código ASM de la misma se encuentra en el apéndice E.0.7. En particular, ver las directivas en las primeras líneas donde se define el tamaño del stack

```

1     # Argument building area (ABA).
2     #define ARG0                (0)
3     #define ARG1                (ARG0 + 4)
4     #define ARG2                (ARG1 + 4)
5     #define ARG3                (ARG2 + 4)
6     #define PADDING_ABA0       (ARG3 + 4)
7
8     # Local and Temporary Area (LTA).
9     #define contador           (PADDING_ABA0 + 4)
10    #define c1                  (contador + 4)
11    #define c2                  (c1 + 4)
12    #define f                   (c2 + 4)
13    #define c                   (f + 4)
14    #define PADDING_LTA_0       (c + 4)
15
16    # Saved-registers area (SRA).
17    #define GP                   (PADDING_LTA_0 + 4)
18    #define FP                   (GP + 4)
19    #define RA                   (FP + 4)
20    #define PADDING_SRA0        (RA + 4)
21
22    # Caller ABA.
23    #define a                    (PADDING_SRA0 + 4)
24    #define i                    (a + 4)
25    #define j                    (i + 4)

```

```
26  #define M                (j + 4)
27  #define N                (M + 4)
28
29  #define STACK_SIZE       (PADDING_SRA0 + 4)
```

Además, al ser una función no hoja, hace falta guardar el registro ra, y crear la *Argument Building Area*, ya que va a llamar a la función mod().

	<b>Argumento Building Area (ABA)</b>
0	ARG0
4	ARG1
8	ARG2
12	ARG3
16	PADDING_ABA0
	<b>Local and Temporary Area (LTA)</b>
20	contador
24	c1
28	c2
32	f
36	c
40	PADDING_LTA_0
	<b>Saved-registers area (SRA)</b>
44	GP
48	FP
52	RA
56	PADDING_SRA0
	<b>Stack caller - caller's ABA</b>
60	a
64	i
68	j
72	M
76	N

Tabla 2: Diseño del stack de la función `vecinos()`. La primer columna se corresponde con el offset en bytes respecto del frame pointer. Se muestra parte del stack del caller, ya que es donde la callee debe guardar los argumentos por convención.

El tamaño del stack queda definido por la directiva

```
1  #define STACK_SIZE (PADDING_SRA0 + 4)
```

y toma el valor de 80 bytes.

Durante el preámbulo de la función hay que crear el stack y guardar los registros neces-

rios, como se muestra en el siguiente fragmento de código

```

1      subu    sp, sp, STACK_SIZE
2      sw      ra, RA(sp)
3      sw      fp, FP(sp)
4      .cprestore GP # Alternative: sw      gp, GP(sp)
5      move    fp, sp

```

Debido a que la función recibe cinco argumentos, los primeros cuatro por los registros a0 a a3 y el quinto en el stack de la función caller, hay que guardar los primeros cuatro valores en el stack del caller, según la convención utilizada:

```

1      sw      a0, a(fp)
2      sw      a1, i(fp)
3      sw      a2, j(fp)
4      sw      a3, M(fp)

```

Al salir de la función, se restauran dichos registros, utilizando las siguientes instrucciones, donde se nota que el valor de retorno es devuelto a través del registro v0:

```

1      lw      v0, contador(fp)
2      lw      ra, RA(sp)
3      lw      fp, FP(sp)
4      lw      gp, GP(sp)
5      addu    sp, sp, STACK_SIZE
6      jr      ra

```

Si bien la función es relativamente extensa, se destacan dos puntos importantes de la misma. La primera, es cómo se realiza el llamado a la función mod():

```

1      # f = mod(i + c1, N)
2      lw      t0, i(fp)
3      lw      t1, c1(fp)
4      add     a0, t1, t0
5      lw      a1, N(fp)
6      jal     mod
7      sw      v0, f(fp)

```

donde se nota que por un lado para acceder al argumento N, se lo hace a través de la ABA del caller, y donde además, se van cargando los registros a0 y a1 por donde va a recibir los argumentos mod(). Luego de realizar el llamado con la instrucción jal, solamente hay que leer el registro v0 donde se almacenó el resultado producido por mod(), ya que la misma devuelve un entero de tamaño word.

Por otro lado, en el siguiente fragmento de código se muestra cómo se realiza el acceso al arreglo que define la línea correspondiente en el tablero. En código C la operación es

```
1  a[f * M + c]
```

mientras que en ASM se debe realizar lo siguiente

```
1  # (a[f * M + c] == 1)
2  lw      t0, f(fp)
3  lw      t1, M(fp)
4  lw      t2, c(fp)
5  mul     t0, t0, t1
6  addu    t0, t0, t2
7  # access a[ ].
8  lw      t1, a(fp)
9  addu    t0, t1, t0
10 lbu     t0, 0(t0)
11 sll     t0, t0, 24 # Stay with lower byte.
12 sra     t0, t0, 24
```

donde se ve que al ser un arreglo de char, se debe realizar el acceso mediante la instrucción `lbu` para acceder al byte correspondiente, y no a un word (como cuando se usa la instrucción `lw`). Además, se realiza un corrimiento a izquierda y derecha con `sll` y `sra` para descartar cualquier byte superior que exista en el registro `t0`.

### 3. Compilación del programa y portabilidad

Debido al requerimiento de utilizar el programa en una computadora con arquitectura MIPS32, se utilizó el emulador `qemu`, utilizando una máquina virtual que contiene el sistema operativo Linux Debian 4.9.65-3 (2017-12-03) mips64 GNU/Linux con las herramientas `gcc` y `make` para compilar el programa desarrollado. Se utilizó el lenguaje de programación C, procurando utilizar únicamente funciones standard que provee el lenguaje. De esta forma, si bien el programa debería poder compilarse independientemente del sistema operativo, se lo compila en un sistema tipo UNIX.

Para obtener un ejecutable, se creó un archivo `Makefile` cuyo contenido se puede ver en la sección B.0.1. Se provee la posibilidad de utilizar los códigos fuente en lenguaje C, o si no utilizar las implementaciones correspondientes en lenguaje ASM MIPS32. Para el primer caso, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
1  $ make
```

Para el segundo caso, ejecutar el siguiente comando

```
1  $ make use_S_files
```

En el mismo puede verse que se utilizan los archivos .S que contienen el código MIPS32 assembly, que serán utilizados directamente por el ensamblador. La declaración de los mismos se realiza en la variable `_SRC1`

```
1  _SRC1_S = conway.c tablero.c cmd_line_parser.c vecinos.S ↵  
    mod.S
```

Los flags de compilación utilizados para todos los casos son

```
1  CFLAGS = -Wall -O0 -g -mips32 -mlong32
```

Para eliminar todos los archivos generados, ejecutar

```
1  $ make clean
```

El programa ejecutable aparecerá en el directorio donde se ejecutó el comando make, con el nombre `conway`. Las imágenes generadas por el programa son guardadas en el directorio `output/`.

## 4. Casos de prueba

Se muestran los resultados de las distintas pruebas de caja negra que se realizaron sobre el programa para determinar su robustez y fiabilidad ante diferentes tipos de entradas. Además, se muestra el resultado de utilizar Valgrind para detectar fugas de memoria, ya que se hace uso de funciones que gestionan memoria dinámica.

### 4.1. Prueba de fuga de memoria

Para verificar si existen bloques de memoria no liberados por el programa, se ejecuta el siguiente comando

```
1  valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all -v ./conway 10 20 20 glider -o estado
```

y el resultado obtenido es

```
1  ==1837==
2  ==1837== HEAP SUMMARY:
3  ==1837==      in use at exit: 0 bytes in 0 blocks
4  ==1837==    total heap usage: 45 allocs, 45 frees, 56,728 bytes allocated
5  ==1837==
6  ==1837== All heap blocks were freed -- no leaks are possible
7  ==1837==
8  ==1837== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
9  ==1837== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

indicando que no existen fugas de memoria.

### 4.2. Pruebas de errores en arumentos al programa

Se creó un script en lenguaje Bash para automatizar las pruebas del programa. El código del script se encuentra en la sección C.0.1, y está compuesto por 5 test.

La salida del script se divide en 2 secciones, cada una con un encabezado indicando el inicio del nuevo test y su nombre, y varias líneas por cada test. La primera línea del cuerpo del test es el comando ejecutado, indicado con la etiqueta `Testing`. La segunda indica si el test fue exitoso o no mediante la etiqueta `PASSED/FAILED` en color verde o rojo respectivamente, y las siguientes líneas son los resultados que produce el programa (mensajes de error, etc...). Por ejemplo, para la prueba de la opción “-o”, se tiene lo siguiente:

```
1  -----
```



```

2  TEST11: no 'output' option arg.
3  -----
4  Testing: ./conway -o
5  PASSED
6  PROGRAM OUTPUT:
7  ./conway: option requires an argument -- 'o'

```

donde se ve que el test fue satisfactorio ya que no se introdujo un nombre de prefijo de archivos de salida.

El script con las pruebas debe ejecutarse dentro del sistema operativo guest. Para ello deberán cargarse los códigos fuentes del programa, compilar los mismos, y finalmente posicionarse en el directorio `src/` para ejecutar el comando `./run_tests.sh`.

Si todos los tests son exitosos, entonces al final de la ejecución se debe obtener el siguiente mensaje.

```

1  -----
2  Test suite ended.
3  -----
4  All tests passed.

```

En caso de que no sean exitosos todos los tests, el script indicará en color rojo cuántos y cuales de estos no lo hicieron, similar al siguiente mensaje

```

1  -----
2  TEST11: no 'input' option parameters.
3  -----
4  Testing: ./conway 10 20 20 glider -o \
5  FAILED
6  PROGRAM OUTPUT:
7  ./tp0: option requires an argument -- 'i'
8
9  -----
10 Test suite ended.
11 -----
12 Failed tests: 1

```

### 4.3. Pruebas en las opciones de programa

En el script de tests se prueban diferentes combinaciones de las opciones de entrada para verificar si el programa es capaz de detectar errores. Los tests son validaciones utilizando opciones y parámetros inválidos, donde se verifica que al intentar ejecutarlo, el programa termina y retorna un mensaje que indique el motivo de la ejecución fallida. El test `test3_valid_parameters` se corresponde con ejecuciones que retornan un código de éxito.

Las salidas arrojadas por el script fueron las siguientes:

```
1 -----
2 TEST1: inexistent 'input' stream.
3 -----
4 Testing: ./conway 10 20 20 invalid
5 PASSED
6     PROGRAM OUTPUT:
7         ERROR: Can't open input stream.
8 -----
9 TEST11: no 'output' option arg.
10 -----
11 Testing: ./conway -o
12 PASSED
13     PROGRAM OUTPUT:
14         ./conway: option requires an argument -- 'o'
15 -----
16 TEST2: invalid 'out' stream.
17 -----
18 Testing: ./conway 10 20 20 glider -o .
19 PASSED
20     PROGRAM OUTPUT:
21         ERROR: Invalid output stream.
22 Testing: ./conway 10 20 20 glider -o ..
23 PASSED
24     PROGRAM OUTPUT:
25         ERROR: Invalid output stream.
26 Testing: ./conway 10 20 20 glider -o /
27 PASSED
28     PROGRAM OUTPUT:
29         ERROR: Invalid output stream.
30 Testing: ./conway 10 20 20 glider -o //
31 PASSED
32     PROGRAM OUTPUT:
33         ERROR: Invalid output stream.
34 -----
35 TEST21: not enough input arguments.
36 -----
37 Testing: ./conway 10
38 PASSED
39     PROGRAM OUTPUT:
40         ERROR: not enough input arguments.
41 Testing: ./conway 10 20
42 PASSED
43     PROGRAM OUTPUT:
44         ERROR: not enough input arguments.
45 Testing: ./conway 10 20 20
46 PASSED
47     PROGRAM OUTPUT:
```

```
48         ERROR: not enough input arguments.
49 -----
50 TEST3: all options with correct parameters.
51 -----
52 Testing: ./conway 10 20 20 glider
53 PASSED
54     PROGRAM OUTPUT:
55         Leyendo estado inicial...
56 Grabando ../output/glider001.pbm
57 Grabando ../output/glider002.pbm
58 Grabando ../output/glider003.pbm
59 Grabando ../output/glider004.pbm
60 Grabando ../output/glider005.pbm
61 Grabando ../output/glider006.pbm
62 Grabando ../output/glider007.pbm
63 Grabando ../output/glider008.pbm
64 Grabando ../output/glider009.pbm
65 Grabando ../output/glider010.pbm
66 Listo.
67 -----
68 Test suite ended.
69 -----
70     All tests passed.
```

#### 4.4. Pruebas de generación de imágenes

Se prueba el programa con los tres archivos provistos: glider, sapo, y pento, utilizando tablero de 20x20 y produciendo 10 iteraciones del juego. Para evitar tener que mostrar demasiadas imágenes, se realizaron 3 videos con el programa FFMPEG [11]. Los mismos se encuentran en el directorio output.

### 5. Pruebas de tiempo de ejecución

Se presentan a continuación los distintos tiempos de ejecución obtenidos corriendo el programa para los 3 archivos brindados, utilizando primero la versión que utiliza solo código C, y luego la versión con la implementación en ASM MIPS32 de la función `vecinos()`.

Se utilizó un script Bash para correr el mismo comando 10 veces y así obtener un promedio de tiempo de corrida. Para todos los archivos, se utilizaron corridas de 10 iteraciones en grillas de 20x20 como se requiere en el enunciado del trabajo.

#### 5.1. Glider



Figura 5.1: Estado Inicial Glider.

Imp.vecinos	real	user	sys
C	4,284	4,049	0,227
Mips	4,791	4,539	0,228

#### 5.2. Sapo

Imp.vecinos	real	user	sys
C	4,682	4,442	0,236
Mips	4,746	4,491	0,249



Figura 5.2: Estado Inicial Sapo.



Figura 5.3: Estado Inicial Pento.

### 5.3. Pento

Imp.vecinos	real	user	sys
C	4,308	4,090	0,192
Mips	4,476	4,215	0.220

## 6. Herramientas de hardware y software utilizadas

La computadora utilizada para realizar el desarrollo y las pruebas tiene las siguientes especificaciones:

- Procesador: Intel i3-6100.
- Memoria: 16GB RAM DDR4.
- Almacenamiento: Disco HDD SATA 1TB de 7200RPM - Disco V-NAND SSD NVME 500 GB.

El entorno de desarrollo utilizado fue una máquina virtual con sistema operativo Linux Debian 4.9.65-3 (2017-12-03) mips64 GNU/Linux ejecutada a través del programa qemu, donde el sistema operativo host fue Linux Ubuntu, cuyos datos de distribución son

- Distributor ID: Ubuntu
- Description: Ubuntu 18.04.4 LTS
- Release: 18.04
- Codename: bionic

Además, se utilizaron las siguientes herramientas:

- Compilador del proyecto: gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516 [7].
- Control del proceso de compilación: GNU Make 4.1 [6].
- Compilador del presente informe: pdfTeX 3.14159265-2.6-1.40.18 (TeX Live 2017/Debian) [8].
- Edición de código fuente: VIM - Vi IMproved 8.0 (2016 Sep 12, compiled Mar 18 2020 18:29:15) [9].
- Depuración del programa: GNU gdb (Debian 7.12-6) 7.12.0.20161007-git [10].
- Creación de videos: ffmpeg version 3.4.6-0ubuntu0.18.04.1 [11].

## 7. Conclusiones

Viendo los tiempos entre la implementación que utiliza Assembler y la que usa puramente C, podemos ver que con todos los archivos iniciales fue la versión que hace uso de vecinos.S la que más tardó.

En una primer interpretación de los tiempos encontrados, se podría concluir que, aún dando las corridas los mismo tiempos, es innecesario crear la función vecinos en S, ya que no reporta beneficio y se necesitó de mayor trabajo para implementarla.

Por otra parte, se considera que la función realizada en assembler, pese a ser de fácil comprensión en términos de qué realiza esta, puede por los mismos motivos no ser una versión óptima. Por ende se considera que se podría programar una versión de vecinos.S tal que mejore los tiempos del mismo.

Las pruebas, por otra parte, se pidieron realizar únicamente para 10 iteraciones en una grilla de 20x20. En una grilla pequeña y pocas iteraciones es difícil apreciar las diferencias de tiempos entre dos programas muy similares. Es por este motivo que, si el programa fue efectivamente realizado para manejar estos número, no se encuentra razón por la que crear una función en .S para este.

Finalmente, de manejar el programa con valores en la cantidad de iteraciones en varios ordenes de magnitud mayor, y con grillas de mayores dimensiones, consideraríamos optimizar el programa reemplazando funciones en C por las mismas en assembler. Siempre buscando la manera de implementar este lo más óptima posible, ya que utilizar una implementación en assembler, como pudimos ver previamente, no garantiza que mejoren los tiempos.

## Referencias

- [1] Hennessy, J. L. - Patterson, D. A. - *Computer Architecture: A Quantitative Approach* - 3<sup>rd</sup> edition - Morgan Kaufmann - 2002.
- [2] Patterson, D. A. - Hennessy, J. L. - *Computer Organization and Design: The Hardware/Software Interface* - 3<sup>rd</sup> edition - Morgan Kaufmann - 2004.
- [3] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2<sup>nd</sup> edition - Prentice Hall - 1988.
- [4] *Apuntes del curso 66.20 Organización de Computadoras* - Cátedra Hamkalo - Facultad de Ingeniería de la Universidad de Buenos Aires.
- [5] *System V Application Binary Interface - MIPS/RISC Processor Supplement* - 3<sup>rd</sup> edition - 1996.
- [6] *GNU Make* - <https://www.gnu.org/software/make/>
- [7] *GNU Gcc* - <https://gcc.gnu.org/>
- [8] *L<sup>A</sup>T<sub>E</sub>X* - <https://www.latex-project.org/>
- [9] *VIM* - <https://vim.sourceforge.io/>
- [10] *GNU gdb* - <https://www.gnu.org/software/gdb/>
- [11] *FFMPEG* - <https://ffmpeg.org/>

## A. Enunciado del trabajo práctico

### 66:20 Organización de Computadoras Trabajo práctico 1: conjunto de instrucciones MIPS

#### 1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI<sup>1</sup>, escribiendo un programa portable que resuelva el problema descrito en la sección 6.

#### 2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

#### 3. Requisitos

El trabajo deberá ser entregado en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 9), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta  $\text{\TeX}$  /  $\text{\LaTeX}$ .

#### 4. Recursos

Usaremos el programa `qemu` [2] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo Debian [3].

---

<sup>1</sup>Application Binary Interface



## 5. Introducción

El “Juego de la Vida” de Conway es un autómata celular, diseñado por el matemático británico John Conway [5] en 1970 [6], y si bien su funcionamiento es simple, computacionalmente es equivalente a una máquina de Turing [7]. Se trata básicamente de una grilla en principio infinita, en cada una de cuyas celdas puede haber un organismo vivo (se dice que la celda está viva, o encendida) o no, en cuyo caso se dice que está muerta o apagada. Se llama “vecinos” de una celda a las ocho celdas adyacentes a ésta. Esta matriz evoluciona en el tiempo en pasos discretos, o estados, y las transiciones de las celdas se realizan de la siguiente manera:

- Si una celda tiene menos de dos o más de tres vecinos encendidos, su siguiente estado es apagado.
- Si una celda encendida tiene dos o tres vecinos encendidos, su siguiente estado es encendido.
- Si una celda apagada tiene exactamente tres vecinos encendidos, su siguiente estado es encendido.
- Todas las celdas se actualizan simultáneamente.

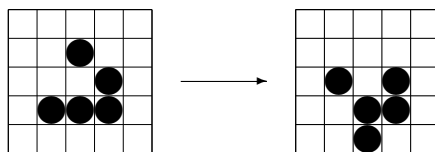


Figura 1: Ejemplo de transición entre estados

## 6. Programa

Se trata de una versión en lenguaje C del “Juego de la Vida”. El programa recibirá por como argumentos tres números naturales,  $i$ ,  $M$  y  $N$ , y el nombre de un archivo de texto con coordenadas en una matriz de  $M \times N$ , y escribirá  $i$  archivos .PBM [8]<sup>2</sup> representando  $i$  estados consecutivos del “Juego de la Vida” en una matriz de  $M \times N$ , tomando como celdas ‘vivas’ iniciales las que están en las coordenadas del archivo de entrada (el primer estado a representar es el inicial). De haber errores, los mensajes de error deberán salir exclusivamente por `stderr`. Si la corrida fue exitosa, usar el

<sup>2</sup>Los nombres de los archivos deberán ser del formato `[outputprefix]_NNN.pbm`, con NNN representando números de orden con ceros a la izquierda: `pref_001.pbm`, `pref_002.pbm`, etc

programa `ffmpeg`[9] para hacer un video estilo “stop motion”[10] (con el paso 3 alcanza).

### 6.1. Condiciones de contorno

Dados los problemas que acarrearía tratar con una matriz infinita, se ha optado por darle un tamaño limitado. En este caso, para las filas y columnas de los ‘bordes’ de la matriz, hay dos maneras básicas de calcular los ‘vecinos’:

1. **Hipótesis del mundo rectangular:** Las posiciones que caerían fuera de la matriz se asumen apagadas. Sencillamente no se computan.
2. **Hipótesis del mundo toroidal:** La fila  $M - 1$  pasa a ser vecina de la fila 0, y la columna  $N - 1$  pasa a ser vecina de la columna 0. Entonces, el vecino superior de la celda  $[0, j]$  es el  $[M - 1, j]$ , el vecino izquierdo de la celda  $[i, 0]$  es el  $[i, N - 1]$ , y viceversa; de esta manera, nunca nos salimos de la matriz. Esta es la opción más interesante, y la que debe usar el programa.

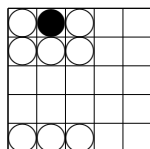


Figura 2: Ejemplo de mundo toroidal: la celda  $(0,1)$  y sus vecinos.

### 6.2. Comportamiento deseado

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ conway -h
Uso:
  conway -h
  conway -V
  conway i M N inputfile [-o outputprefix]
Opciones:
  -h, --help      Imprime este mensaje.
  -V, --version   Da la versión del programa.
  -o Prefijo de los archivos de salida.
Ejemplos:
conway 10 20 20 glider -o estado
Representa 10 iteraciones del Juego de la Vida en una matriz de 20x20,
con un estado inicial tomado del archivo ‘glider’.
Los archivos de salida se llamarán estado_n.pbm.
Si no se da un prefijo para los archivos de salida,
```

el prefijo será el nombre del archivo de entrada.

Ahora usaremos el programa para generar una secuencia de estados del Juego de la Vida.

```
$ conway 5 10 10 glider -o estado
Leyendo estado inicial...
Grabando estado_1.pbm
Grabando estado_2.pbm
Grabando estado_3.pbm
Grabando estado_4.pbm
Grabando estado_5.pbm
Listo
```

El formato del archivo de entrada es de texto, con los números de fila y columna separados por espacios, a una celda ocupada por línea. Ejemplo: si el archivo `glider` representa un planeador en el centro de una grilla de  $10 \times 10$ , se verá de la siguiente manera:

```
$ cat glider
5 3
5 4
5 5
3 4
4 5
```

El programa deberá retornar un error si las coordenadas de las celdas encendidas están fuera de la matriz ( $[0..M - 1]$ ,  $[0..N - 1]$ ), o si el archivo no cumple con el formato.

## 7. Implementación

El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación.

### 7.1. Portabilidad

Pese a contener fragmentos en assembler MIPS32, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad.

Para satisfacer esto, el programa deberá proveer dos versiones de `conway()`, incluyendo la versión MIPS32, pero también una versión C, pensada para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

## 7.2. API

Gran parte del programa estará implementada en lenguaje C. Sin embargo, la función `vecinos()` estará implementada en assembler MIPS32, para proveer soporte específico en nuestra plataforma principal de desarrollo, MIPS32.

El programa en C deberá interpretar los argumentos de entrada, pedir la memoria necesaria para la matriz de estado *A*, popular la matriz con los contenidos del archivo de entrada, y computar el siguiente estado para cada celda valiéndose de la siguiente función:

```
unsigned int vecinos(unsigned char *a,
                    unsigned int i, unsigned int j,
                    unsigned int M, unsigned int N);
```

Donde *a* es un puntero a la posición  $[0, 0]$  de la matriz, *i* y *j* son la fila y la columna respectivamente del elemento cuyos vecinos queremos calcular, y *M* y *N* son las cantidades de filas y columnas de la matriz *A*. El valor de retorno de la función `vecinos` es la cantidad de celdas vecinas que están encendidas en el estado actual. Los elementos de *A* pueden representar una celda cada uno, aunque para reducir el uso de memoria podrían contener hasta ocho cada uno. Después de computar el siguiente estado para la matriz *A*, el programa deberá escribir un archivo en formato PBM [8] representando las celdas encendidas con color blanco y las apagadas con color negro <sup>3</sup>.

## 7.3. ABI

El pasaje de parámetros entre el código C (`main()`, etc) y la rutina `vecinos()`, en assembler, deberá hacerse usando la ABI explicada en clase: los argumentos correspondientes a los registros `$a0-$a3` serán almacenados por el *callee*, siempre, en los 16 bytes dedicados de la sección “function call argument area” [4].

## 7.4. Algoritmo

El algoritmo a implementar es el algoritmo del Juego de la Vida de Conway[6], explicado en clase.

## 8. Proceso de Compilación

En este trabajo, el desarrollo se hará parte en C y parte en lenguaje Assembler. Los programas escritos serán compilados o ensamblados según el caso, y posteriormente enlazados, utilizando las herramientas de GNU

---

<sup>3</sup>Si se utiliza sólo un pixel por celda, no se podrá apreciar el resultado a simple vista. Pruebe haciendo que una celda sea representada por grupos de por ejemplo 16x16 pixels.

disponibles en el sistema NetBSD utilizado. Como resultado del enlace, se genera la aplicación ejecutable.

## 9. Informe

El informe deberá incluir <sup>4</sup>:

- Este enunciado;
- Documentación relevante al diseño e implementación del programa, incluyendo un diagrama del stack de la función `vecinos`;
- Corridas de prueba para diez iteraciones, en una matriz de  $20 \times 20$ , de los archivos de entrada `glider`, `pento` y `sapo`, con los comentarios pertinentes;
- El código fuente completo, en formato digital.

## 10. Mejoras opcionales

- Una versión de terminal, que permita ver en tiempo real la evolución del sistema (y suprima los archivos de salida).
- Un editor de pantalla, de modo texto, a una celda por caracter. Esto permite experimentar con el programa, particularmente combinado con la versión de tiempo real.

## 11. Fecha de entrega

Primera entrega: Semana del 21 de Mayo.

Revisión: Semana del 28 de Mayo.

Última fecha de entrega: Semana del 4 de Junio.

## Referencias

- [1] GXemul, <http://gavare.se/gxemul/>.
- [2] QEMU, <https://www.qemu.org/>
- [3] Debian, the Universal Operating System, <https://www.debian.org/>.
- [4] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

---

<sup>4</sup>no incluir el enunciado en el `.tex` del informe, con agregar el PDF a la entrega alcanza

- [5] John Horton Conway, [1937-2020], [https://en.wikipedia.org/wiki/John\\_Horton\\_Conway](https://en.wikipedia.org/wiki/John_Horton_Conway).
- [6] Juego de la Vida de Conway, [http://es.wikipedia.org/wiki/Juego\\_de\\_la\\_vida](http://es.wikipedia.org/wiki/Juego_de_la_vida).
- [7] Máquina de Turing, implementada en el Juego de la Vida de Conway, <http://rendell-attic.org/gol/tm.htm>.
- [8] <http://netpbm.sourceforge.net/doc/pbm.html>
- [9] <https://www.ffmpeg.org/>
- [10] <https://lukecyca.com/2013/stop-motion-with-ffmpeg.html>

## B. Makefile

### B.0.1. makefile

```
1 # -----
2 #
3 # The source files must have .c extension.
4 # The object code must have .o extension.
5 # The header files must have .h extension.
6 #
7 # To compile, execute 'make'.
8 # To clean all the compilation files, issue the command
9 # 'make clean'.
10 #
11 # -----
12 # List all the header and object files separated by a blank
13 # space.
14 _DEPS = params_t.h messages.h
15 _SRC1_c = conway.c tablero.c cmd_line_parser.c vecinos.c mod.c
16 _SRC1_S = conway.c tablero.c cmd_line_parser.c vecinos.S mod.S
17 # -----
18 # Configuration.
19 CC = gcc
20 CFLAGS = -Wall -O0 -g -mips32 -mlong32
21 OUTPUT1 = conway
22 # -----
23 all: $(OUTPUT1)
24
25 $(OUTPUT1): $(_SRC1_c)
26     $(CC) $(CFLAGS) -o $@ $(_SRC1_c)
27
28 use_S_files: $(_SRC1_s)
29     $(CC) $(CFLAGS) -o $(OUTPUT1) $(_SRC1_S)
30
31 .PHONY: clean use_S_files
32
33 clean:
34     rm -f ./*.o ./*~ ./*.core ./*~
35     rm -f $(OUTPUT1)
```

## C. Tests

### C.0.1. run\_tests.sh

```
1  #!/usr/bin/env bash
2  # -----
3  #
4  # Script to test errors in the program arguments.
5  #
6  # To remove newlines from a textfile, use
7  # printf %s "$(cat file)" > file
8  #
9  # To print contents of a file, including control characters, ↵
   do:
10 # oc -c file
11 #
12 # -----
13
14 # Program name to test.
15 PROGRAM_NAME='./conway'
16
17 # Failed tests counter.
18 failedTests=0;
19
20 # Colors to be used.
21 RED="\e[31m";
22 GREEN="\e[32m";
23 CYAN="\e[96m";
24 YELLOW="\e[93m";
25 BOLD="\033[1m";
26 DEFAULT="\e[0m";
27
28 # Helper and formatting functions definitions.
29 function header() {
30     echo -e "$CYAN↵
   -----↵
   $DEFAULT";
31     echo -e "$CYAN$1$DEFAULT";
32     echo -e "$CYAN↵
   -----↵
   $DEFAULT";
33 }
34
35 function msg_true() {
36     echo -e "$GREEN\OPASSED \n $DEFAULT PROGRAM OUTPUT:\n\t$1";
37 }
```



```

38
39 function msg_false() {
40     echo -e "$RED\OFAILED \n $DEFAULT PROGRAM OUTPUT:\n\t$1";
41 }
42
43 function msg_testing() {
44     echo -e "Testing: $BOLD$1$DEFAULT";
45 }
46
47 function success_msg() {
48     echo -e "  $GREEN$1$DEFAULT";
49 }
50
51 function error_msg() {
52     echo -e "  $RED$1$DEFAULT";
53 }
54
55 # -----
56 # Input parameters tests.
57 # -----
58
59 # Define the expected outputs of each of the test cases with ↵
    its associated
60 # test functions.
61 EXPECTED_OUTPUT_INEXISTENT_INPUT_STREAM=("ERROR: Can't open ↵
    input stream.")
62
63 function test1_parameter_input_inexistent_stream(){
64     header "TEST1: inexistent 'input' stream."
65
66     commands=(
67         "10 20 20 invalid"
68     )
69
70     for i in "${commands[@]}"
71     do
72
73         msg_testing "$PROGRAM_NAME $i"
74
75         PROGRAM_OUTPUT=$(($PROGRAM_NAME $i 2>&1))
76
77         if [[ "$EXPECTED_OUTPUT_INEXISTENT_INPUT_STREAM" == "↵
            $PROGRAM_OUTPUT" ]]; then
78             msg_true "$PROGRAM_OUTPUT"
79         else
80             msg_false "$PROGRAM_OUTPUT"
81             failedTests=$((failedTests+1));

```

```

82     fi
83
84     done
85 }
86
87 function test11_parameter_output_no_argument(){
88     header "TEST11: no 'output' option arg."
89
90     commands=("-o ")
91
92     for i in "${commands[@]}"
93     do
94
95         msg_testing "$PROGRAM_NAME $i"
96
97         PROGRAM_OUTPUT=$(($PROGRAM_NAME $i 2>&1)
98
99         if [[ "./conway: option requires an argument -- 'o'" == "$←
PROGRAM_OUTPUT" ]]; then
100             msg_true "$PROGRAM_OUTPUT"
101         else
102             msg_false "$PROGRAM_OUTPUT"
103             msg_false "EXPECTED:"
104             msg_false "$EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT"
105             failedTests=$((failedTests+1));
106         fi
107     done
108 }
109
110
111 EXPECTED_OUTPUT_OUTPUT_INVALID_STREAM=("ERROR: Invalid output ←
stream.")
112
113 function test2_parameter_output_invalid_stream(){
114     header "TEST2: invalid 'out' stream."
115
116     commands=(
117 "10 20 20 glider -o ."
118 "10 20 20 glider -o .."
119 "10 20 20 glider -o /"
120 "10 20 20 glider -o //"
121 )
122
123     for i in "${commands[@]}"
124     do
125
126         msg_testing "$PROGRAM_NAME $i"

```

```

127
128     PROGRAM_OUTPUT=$(($PROGRAM_NAME $i 2>&1)
129
130     if [[ "$EXPECTED_OUTPUT_OUTPUT_INVALID_STREAM" == "↵
131         $PROGRAM_OUTPUT" ]]; then
132         msg_true "$PROGRAM_OUTPUT"
133     else
134         msg_false "$PROGRAM_OUTPUT"
135         failedTests=$((failedTests+1));
136     fi
137 done
138 }
139
140 EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT=("ERROR: not enough input ↵
141     arguments.")
142
143 function test21_parameter_output_no_argument(){
144     header "TEST21: not enough input arguments."
145
146     commands=("10 "
147         "10 20 "
148         "10 20 20 ")
149
150     for i in "${commands[@]}"; do
151
152         msg_testing "$PROGRAM_NAME $i"
153
154         PROGRAM_OUTPUT=$(($PROGRAM_NAME $i 2>&1)
155
156         if [[ "$EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT" == "↵
157             $PROGRAM_OUTPUT" ]]; then
158             msg_true "$PROGRAM_OUTPUT"
159         else
160             msg_false "$PROGRAM_OUTPUT"
161             failedTests=$((failedTests+1));
162         fi
163     done
164 }
165
166 EXPECTED_OUTPUT_VALID_PARAMETERS=(
167     "Leyendo estado inicial..."
168     "Grabando ../output/gliders001.pbm"
169     "Grabando ../output/gliders002.pbm"
170     "Grabando ../output/gliders003.pbm"

```

```
171 Grabando ../output/gliders004.pbm
172 Grabando ../output/gliders005.pbm
173 Grabando ../output/gliders006.pbm
174 Grabando ../output/gliders007.pbm
175 Grabando ../output/gliders008.pbm
176 Grabando ../output/gliders009.pbm
177 Grabando ../output/gliders010.pbm
178 Listo."
179 )
180
181 function test3_valid_parameters(){
182     header "TEST3: all options with correct parameters."
183
184     commands=(
185         "10 20 20 glider")
186
187     for i in "${commands[@]}"
188     do
189
190         msg_testing "$PROGRAM_NAME $i"
191
192         PROGRAM_OUTPUT=$(($PROGRAM_NAME $i 2>&1)
193
194
195         if [[ "$EXPECTED_OUTPUT_VALID_PARAMETERS" == "$PROGRAM_OUTPUT" ]]; then
196             msg_true "$PROGRAM_OUTPUT"
197         else
198             msg_false "$PROGRAM_OUTPUT"
199             failedTests=$((failedTests+1));
200         fi
201
202     done
203 }
204 # -----
205 # Run the tests.
206 # -----
207 test1_parameter_input_inexistent_stream
208 test11_parameter_output_no_argument
209 test2_parameter_output_invalid_stream
210 test21_parameter_output_no_argument
211 test3_valid_parameters
212
213 header "Test suite ended."
214
215 if [[ $failedTests -eq $zero ]]; then
216     success_msg "All tests passed.";
```

```
217 else
218     error_msg "Failed tests: $failedTests";
219 fi
```

## D. Header files

### D.0.1. cmd\_line\_parser.h

```
1  #ifndef CMD_LINE_PARSER__H
2  #define CMD_LINE_PARSER__H
3
4  #include <getopt.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <unistd.h>
9
10 #include "messages.h"
11 #include "params_t.h"
12
13 #ifndef STD_STREAM_TOKEN
14 #define STD_STREAM_TOKEN "-"
15 #endif
16
17 #ifndef VERSION
18 #define VERSION "1.0.0"
19 #endif
20
21 typedef enum output_states_
22 {
23     outOK,
24     outERROR
25 } output_state;
26
27 void option_iterations(const char *args, params_t *params);
28 void option_rows_size(const char *args, params_t *params);
29 void option_columns_size(const char *args,
30                          params_t *params);
31 output_state validate_stream_name(char *streamName);
32 output_state option_input_file(char *arg, params_t *params);
33 output_state option_output_prefix(char *arg,
34                                   params_t *params);
35 void option_version(void);
36 void option_help(char *arg);
37 output_state parse_cmd_line(int argc, char **argv,
38                             params_t *params);
39 void imprimir_uso();
40 void manejar_comandos(char *opt);
41
42 #endif
```

**D.0.2. mod.h**

```
1 #ifndef MOD__H
2 #define MOD__H
3
4 extern int mod(int x, int m);
5
6 #endif
```

**D.0.3. params\_t.h**

```
1  #ifndef PARAMS_T__H
2  #define PARAMS_T__H
3
4  #define MAX_SIZE_PREFIX 1024
5
6  typedef struct params_t
7  {
8      int i;
9      int N;
10     int M;
11     FILE *inputStream;
12     FILE *outputStream;
13     char *prefix;
14 } params_t;
15
16 #endif
```



**D.0.4. tablero.h**

```
1  #ifndef TABLERO__H
2  #define TABLERO__H
3
4  #include <stdbool.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  #include "messages.h"
10 #include "params_t.h"
11 #include "vecinos.h"
12
13 /* Mantener un número múltiplo de 10 */
14 #define ZOOM_ARCHIVO 30
15
16 typedef struct tablero
17 {
18     unsigned char *tabla;
19     int l;
20     int h;
21 } tablero_t;
22
23 void tablero_eliminar(tablero_t *self);
24 tablero_t *tablero_crear(int length, int height);
25 int tablero_cargar_tablero(tablero_t *self,
26                             params_t *params);
27 void tablero_modificar_estado_campo(tablero_t *self,
28                                     int fila, int col);
29 void tablero_imprimir(tablero_t *self, int iteracion,
30                       params_t *params);
31 int mod(int x, int m);
32 unsigned int vecinos_c(unsigned char *a, unsigned int i,
33                       unsigned int j, unsigned int M,
34                       unsigned int N);
35 unsigned int vecinos_mips(unsigned char *a, unsigned int i,
36                          unsigned int j, unsigned int M,
37                          unsigned int N);
38 tablero_t *tablero_modificar(tablero_t *self);
39 void tablero_eliminar(tablero_t *self);
40
41 #endif
```

**D.0.5. vecinos.h**

```
1 #ifndef VECINOS__H
2 #define VECINOS__H
3
4 extern unsigned int vecinos(unsigned char *a,
5                             unsigned int i, unsigned int j,
6                             unsigned int M, unsigned int N);
7
8 #endif
```

## E. Código fuente

### E.0.1. cmd\_line\_parser.c

```
1  #include "cmd_line_parser.h"
2
3  void option_iterations(const char *args, params_t *params)
4  {
5      int iterations;
6
7      if (sscanf(args, "%d", &iterations) != 1 ||
8          iterations <= 0)
9      {
10         fprintf(stderr, ERROR_INVALID_ITERATIONS);
11         exit(EXIT_FAILURE);
12     }
13
14     params->i = iterations;
15 }
16
17 void option_rows_size(const char *args, params_t *params)
18 {
19     int rows;
20
21     if (sscanf(args, "%d", &rows) != 1 || rows <= 0)
22     {
23         fprintf(stderr, ERROR_INVALID_ROWS);
24         exit(EXIT_FAILURE);
25     }
26
27     params->N = rows;
28 }
29
30 void option_columns_size(const char *args, params_t *params)
31 {
32     int cols;
33
34     if (sscanf(args, "%d", &cols) != 1 || cols <= 0)
35     {
36         fprintf(stderr, ERROR_INVALID_COLUMNS);
37         exit(EXIT_FAILURE);
38     }
39
40     params->M = cols;
41 }
42
```

```
43 output_state validate_stream_name(char *streamName)
44 {
45     if (streamName == NULL)
46     {
47         return outERROR;
48     }
49
50     /* strcmp() returns 0 if identical. */
51     if (!strcmp(streamName, ".") ||
52         !strcmp(streamName, "..") ||
53         !strcmp(streamName, "/") || !strcmp(streamName, "//"))
54     {
55         return outERROR;
56     }
57
58     return outOK;
59 }
60
61 output_state option_input_file(char *arg, params_t *params)
62 {
63     if (validate_stream_name(arg) == outERROR)
64     {
65         fprintf(stderr, ERROR_INVALID_INPUT_STREAM);
66         exit(EXIT_FAILURE);
67     }
68
69     /* We cover the optional enhancement to print the PBM
70      * images to the terminal.
71      */
72     if (strcmp(arg, STD_STREAM_TOKEN) == 0)
73     {
74         params->inputStream = stdin;
75     }
76     else
77     {
78         params->inputStream = fopen(arg, "r");
79     }
80
81     if ((params->inputStream) == NULL)
82     {
83         fprintf(stderr, ERROR_OPENING_INPUT_STREAM);
84         exit(EXIT_FAILURE);
85     }
86
87     return outOK;
88 }
89
```

```
90 output_state option_output_prefix(char *arg,
91                                   params_t *params)
92 {
93     if (validate_stream_name(arg) == outERROR)
94     {
95         fprintf(stderr, ERROR_INVALID_OUTPUT_STREAM);
96         exit(EXIT_FAILURE);
97     }
98
99     if (strcmp(arg, STD_STREAM_TOKEN) == 0)
100     {
101         params->outputStream = stdout;
102         params->prefix = NULL;
103     }
104     else
105     {
106         params->outputStream = NULL;
107         params->prefix = arg;
108     }
109
110     return outOK;
111 }
112
113 void option_version(void)
114 {
115     fprintf(stderr, "%s\n", VERSION);
116
117     exit(EXIT_SUCCESS);
118 }
119
120 void option_help(char *arg)
121 {
122     fprintf(stderr, "Uso:\n");
123     fprintf(stderr, " %s -h\n", arg);
124     fprintf(stderr, " %s -V\n", arg);
125     fprintf(stderr, " %s i M N inputfile [-o outputprefix]\n",
126             arg);
127
128     fprintf(stderr, "Opciones:\n");
129     fprintf(stderr, " -h, --help\tImprime este mensaje.\n");
130     fprintf(stderr,
131             " -V, --version\tDa la versión del programa.\n");
132     fprintf(stderr,
133             " -o Prefijo de los archivos de salida.\n");
134
135     fprintf(stderr, "Ejemplos:\n");
136     fprintf(stderr, " %s 10 20 20 glider -o estado\n", arg);
```

```
137     fprintf(stderr,
138         " Representa 10 iteraciones del Juego de la Vida "
139         "en una matriz de 20x20,\n");
140     fprintf(stderr,
141         " con un estado inicial tomado del archivo "
142         "'glider'.\n");
143     fprintf(stderr,
144         " Los archivos de salida se llamarán "
145         "estado_n.pbm.\n");
146     fprintf(stderr,
147         " Si no se da un prefijo para los archivos de "
148         "salida,\n");
149     fprintf(stderr,
150         " el prefijo será el nombre del archivo de "
151         "entrada.\n");
152
153     exit(EXIT_SUCCESS);
154 }
155
156 output_state parse_cmd_line(int argc, char **argv,
157                             params_t *params)
158 {
159     size_t n_detected_args = 0;
160
161     if ((argc != 2 && argc != 5 && argc != 7))
162     {
163         fprintf(stderr, ERROR_NOT_ENOUGH_ARGS);
164         return outERROR;
165     }
166
167     if ((argc == 5) || (argc == 7))
168     {
169         n_detected_args = argc;
170         /* Process fixed options. */
171         option_iterations(argv[1], params);
172         option_rows_size(argv[2], params);
173         option_columns_size(argv[3], params);
174         option_input_file(argv[4], params);
175         /* Set the default values. */
176         params->outputStream = NULL;
177         params->prefix = argv[4];
178     }
179
180     /* 'version' and 'help' have no arguments. Output has an
181        optional one. The options are distinguished by the ASCII
182        code of the 'char' variables. */
183     struct option cmd_line_options[] = {
```

```
184     {"version", no_argument, NULL, 'V'},
185     {"help", no_argument, NULL, 'h'},
186     {"output", required_argument, NULL, 'o'},
187     {0, 0, 0, 0}};
188
189     char *optstring = "Vho:";
190     int index_ptr = 0;
191     int option_code;
192     output_state option_state = outERROR;
193     char *program_name = argv[0];
194
195     while ((option_code = getopt_long(argc, argv, optstring,
196                                     cmd_line_options,
197                                     &index_ptr)) != -1)
198     {
199         n_detected_args++;
200         switch (option_code)
201         {
202             case 'V':
203                 option_version();
204                 exit(EXIT_SUCCESS);
205                 break;
206             case 'h':
207                 option_help(program_name);
208                 exit(EXIT_SUCCESS);
209                 break;
210             case 'o':
211                 option_state = option_output_prefix(optarg, ←
                params);
212                 break;
213             default:
214                 option_state = outERROR;
215                 break;
216         }
217         if(option_state == outERROR){
218             return outERROR;
219         }
220     }
221
222     if (n_detected_args == 0)
223     {
224         fprintf(stderr, ERROR_NOT_ENOUGH_ARGS);
225         return outERROR;
226     }
227
228     return outOK;
229 }
```

**E.0.2. conway.c**

```
1  #include <stdbool.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #include "cmd_line_parser.h"
6  #include "messages.h"
7  #include "params_t.h"
8  #include "tablero.h"
9
10 /*
11  For memory leaks checking, precede the program call with the
12  following commands:
13
14  valgrind --tool=memcheck --leak-check=full
15  --show-leak-kinds=all -v ./conway 10 20 20 glider -o estado
16
17  */
18
19 /*
20  Parámetros recibidos:
21  i -> cantidad de archivos a escribir
22  M -> coordenada x
23  N -> coordenada y
24  path -> nombre archivo */
25
26 int main(int argc, char *argv[])
27 {
28     params_t params;
29
30     if (parse_cmd_line(argc, argv, &params) == outERROR)
31     {
32         exit(EXIT_FAILURE);
33     }
34
35     tablero_t *tablero = tablero_crear(params.M, params.N);
36     tablero_cargar_tablero(tablero, &params);
37
38     int n = 0;
39     for (n = 1; n <= params.i; ++n)
40     {
41         tablero_imprimir(tablero, n, &params);
42         tablero = tablero_modificar(tablero);
43     }
44 }
```



```
45     tablero_eliminar(tablero);  
46     fprintf(stdout, MSG_READY);  
47  
48     return EXIT_SUCCESS;  
49 }
```

**E.0.3. mod.c**

```
1 #include "mod.h"
2
3 int mod(int x, int m)
4 {
5     int r = x % m;
6     return r < 0 ? r + m : r;
7 }
```

**E.0.4. mod.S**

```

1  #include <sys/regdef.h>
2
3  # Leaf function.
4
5  # Local and Temporary Area (LTA).
6  #define r 0
7  #define PADDING_LTA_0 r + 4
8
9  # Saved-registers area (SRA).
10 #define GP PADDING_LTA_0 + 4
11 #define FP GP + 4
12
13 # Caller ABA.
14 #define x FP + 4
15 #define m x + 4
16
17 #define STACK_SIZE FP + 4
18
19 .text
20 .align 2
21 .globl mod
22 .ent mod
23
24 .set noreorder
25 .cload t9
26 .set reorder
27
28 mod:
29     # Allocate memory for the stack.
30     subu    sp,sp,STACK_SIZE
31     # Save the callee-saved registers used by the caller in ←
32     # the SRA.
33     sw      fp,FP(sp)
34     # We adopt the convention of using the frame pointer
35     # as our index in the stack.
36     move    fp,sp
37     # Now we save the arguments that were loaded by the caller
38     # in the area reserved by the callee.
39     sw      a0, x(fp)
40     sw      a1, m(fp)
41     # -----
42     #####
43     # body: mod()

```

```
44 #####
45 remu    t0,a0,a1
46 sw      t0,r(fp)
47 blt     t0, zero, r_plus_m
48 lw      v0, r(fp)
49 b       exit_function
50
51 r_plus_m:
52     add    v0, t0, a1
53     # -----
54 exit_function:
55     # Stack frame unwinding.
56     lw     fp, FP(sp)
57     lw     gp, GP(sp)
58     addu   sp, sp, STACK_SIZE
59     jr     ra
60
61 .end     mod
```

**E.0.5. tablero.c**

```
1  #include "tablero.h"
2
3  /* Para facilitar la tarea de mips, la grilla es realmente
4  una lista que es luego tratada como un grilla */
5  tablero_t *tablero_crear(int length, int height)
6  {
7      /* TODO: check malloc and calloc errors. */
8      tablero_t *tablero =
9          (tablero_t *)malloc(sizeof(tablero_t));
10     if (tablero == NULL)
11     {
12         fprintf(stderr, ERROR_MEMORY_REQUEST);
13         return NULL;
14     }
15     tablero->tabla = (unsigned char *)calloc(
16         height * length, sizeof(unsigned char));
17
18     if (tablero->tabla == NULL)
19     {
20         fprintf(stderr, ERROR_MEMORY_REQUEST);
21         return NULL;
22     }
23     tablero->l = length;
24     tablero->h = height;
25     return tablero;
26 }
27
28 /* Carga el tablero a partir del nombre de archivo pasado
29 por parámetro. Devuelve 0 si se cargó correctamente, otro en
30 caso de error.
31 */
32 int tablero_cargar_tablero(tablero_t *self,
33                             params_t *params)
34 {
35     if (params->inputStream == NULL)
36     {
37         fprintf(stderr, ERROR_INVALID_INPUT_STREAM);
38         exit(EXIT_FAILURE);
39     }
40
41     fprintf(stdout, MSG_READING_INIT_STATE);
42     char buffer[4];
43     while (fgets(buffer, 100, params->inputStream) != NULL)
44     {
```

```

45     int fila = buffer[0] - '0';
46     int col = buffer[2] - '0';
47     self->tabla[fila * self->l + col] = 1;
48 }
49 fclose(params->inputStream);
50
51     return 0;
52 }
53
54 /* Llena campo indicado del tablero
55 de no ser posible modificar su valor retorna 1,
56 0 en caso de modificarlo correctamente. */
57 void tablero_modificar_estado_campo(tablero_t *self,
58                                     int fila, int col)
59 {
60     int actual = self->tabla[fila * self->l + col];
61     actual == 0 ? (self->tabla[fila * self->l + col] = 1)
62                 : (self->tabla[fila * self->l + col] = 0);
63 }
64
65 void tablero_imprimir(tablero_t *self, int iteracion,
66                       params_t *params)
67 {
68     if (params->outputStream == stdout)
69     {
70         int i = 0;
71         for (i = 0; i < self->h; i++)
72         {
73             int j = 0;
74             for (j = 0; j < self->l; j++)
75             {
76                 printf("%i ", self->tabla[i * self->l + j]);
77             }
78             printf("\n");
79         }
80         printf("\n");
81     }
82     else
83     {
84         char filename[sizeof("pref100.pbm") + MAX_SIZE_PREFIX];
85         sprintf(filename, "../output/%s%03d.pbm",
86                 params->prefix, iteracion);
87         FILE *fp = fopen(filename, "wb");
88         if (fp == NULL)
89         {
90             fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);

```

```

91     }
92     fprintf(stdout, MSG_RECORDING);
93     printf(" %s\n", filename);
94
95     /* Negro */
96     static unsigned char color_vivo[3] = {0, 0, 0};
97     /* Blanco */
98     static unsigned char color_muerto[3] = {255, 255, 255};
99     (void)fprintf(fp, "P6\n%d %d\n255\n",
100                  self->l * ZOOM_ARCHIVO,
101                  self->h * ZOOM_ARCHIVO);
102
103     int i = 0;
104     for (i = 0; i < (self->h * ZOOM_ARCHIVO); i++)
105     {
106         int j = 0;
107         for (j = 0; j < (self->l * ZOOM_ARCHIVO); j++)
108         {
109             int x = i / ZOOM_ARCHIVO;
110             int y = j / ZOOM_ARCHIVO;
111             int vivo = self->tabla[x * self->l + y];
112             if (vivo)
113             {
114                 (void)fwrite(color_vivo, 1, 3, fp);
115             }
116             else
117             {
118                 (void)fwrite(color_muerto, 1, 3, fp);
119             }
120         }
121     }
122     fclose(fp);
123 }
124 }
125
126 tablero_t *tablero_modificar(tablero_t *self)
127 {
128     tablero_t *tablero_nuevo =
129         tablero_crear(self->l, self->h);
130     if (!tablero_nuevo) return NULL;
131     int i = 0;
132     for (i = 0; i < self->h; i++)
133     {
134         int j = 0;
135         for (j = 0; j < self->l; j++)
136         {

```

```
137         unsigned int v =
138             vecinos(&self->tabla[0], i, j, self->l, self->h);
139         int viva = self->tabla[i * self->l + j];
140         if ((v < 2) || (v > 3))
141         {
142             tablero_nuevo->tabla[i * self->l + j] = 0;
143         }
144         if (viva && (v == 2 || v == 3))
145         {
146             tablero_nuevo->tabla[i * self->l + j] = 1;
147         }
148         if (!viva && v == 3)
149         {
150             tablero_nuevo->tabla[i * self->l + j] = 1;
151         }
152     }
153 }
154 /* Ahora lo elimino */
155 tablero_eliminar(self);
156 return tablero_nuevo;
157 }
158
159 /* Elimina el tablero creado */
160 void tablero_eliminar(tablero_t *self)
161 {
162     free(self->tabla);
163     free(self);
164 }
```



**E.0.6. vecinos.c**

```
1  #include "vecinos.h"
2
3  unsigned int vecinos(unsigned char *a, unsigned int i,
4                      unsigned int j, unsigned int M,
5                      unsigned int N)
6  {
7      int contador = 0;
8      int c1 = 0;
9      int c2 = 0;
10     int f = 0;
11     int c = 0;
12
13     for (c1 = -1; c1 <= 1; ++c1)
14     {
15         f = mod(i + c1, N);
16
17         for (c2 = -1; c2 <= 1; ++c2)
18         {
19             if (c1 == 0 && c2 == 0)
20             {
21                 continue;
22             }
23             c = mod(j + c2, M);
24             if (a[f * M + c] == 1)
25             {
26                 contador++;
27             }
28         }
29     }
30     return contador;
31 }
```

**E.0.7. vecinos.S**

```
1  #include <sys/regdef.h>
2
3  # Non-leaf function.
4
5  # Argument building area (ABA).
6  #define ARG0          (0)
7  #define ARG1          (ARG0 + 4)
8  #define ARG2          (ARG1 + 4)
9  #define ARG3          (ARG2 + 4)
10 #define PADDING_ABA0   (ARG3 + 4)
11
12 # Local and Temporary Area (LTA).
13 #define contador       (PADDING_ABA0 + 4)
14 #define c1             (contador + 4)
15 #define c2             (c1 + 4)
16 #define f              (c2 + 4)
17 #define c              (f + 4)
18 #define PADDING_LTA_0  (c + 4)
19
20 # Saved-registers area (SRA).
21 #define GP             (PADDING_LTA_0 + 4)
22 #define FP             (GP + 4)
23 #define RA             (FP + 4)
24 #define PADDING_SRA0   (RA + 4)
25
26 # Caller ABA.
27 #define a              (PADDING_SRA0 + 4)
28 #define i              (a + 4)
29 #define j              (i + 4)
30 #define M              (j + 4)
31 #define N              (M + 4)
32
33 #define STACK_SIZE     (PADDING_SRA0 + 4)
34
35 .text
36 .align 2
37 .globl vecinos
38 .ent vecinos
39 # Debugger metadata.
40 .frame fp,STACK_SIZE,ra
41 .set noreorder
42 .cplod t9
43 .set reorder
44
```

```

45 vecinos:
46     # Allocate memory for the stack.
47     subu    sp,sp,STACK_SIZE
48     # Save the callee-saved registers used by the caller in ←
        the SRA.
49     sw      ra, RA(sp)
50     sw      fp, FP(sp)
51     .cprestore GP # Alternative: sw      gp, GP(sp)
52     # We adopt the convention of using the frame pointer
53     # as our index in the stack.
54     move    fp,sp
55     # Now we save the arguments that were loaded by the caller
56     # in the area reserved by the callee.
57     # In this case, the caller is passing us 5 arguments.
58     # Four of them through a_i registers. The last one through
59     # its ABA (counting 16B onwards, from its stack frame base←
        ).
60     sw      a0, a(fp)
61     sw      a1, i(fp)
62     sw      a2, j(fp)
63     sw      a3, M(fp)
64     # -----
65     #####
66     # body: vecinos() #
67     #####
68     sw      zero, contador(fp)
69     sw      zero, c1(fp)
70     sw      zero, c2(fp)
71     sw      zero, f(fp)
72     sw      zero, c(fp)
73
74     # c1 = -1
75     li      t0, -1
76     sw      t0, c1(fp)
77     # c1 <= 1
78 outer_for:
79     li      t1, 1
80     lw      t0, c1(fp)
81     ble     t0, t1, outer_for_body
82     b       exit_function
83
84 outer_for_body:
85     # f = mod(i + c1, N)
86     lw      t0, i(fp)
87     lw      t1, c1(fp)
88     add     a0, t1, t0
89     lw      a1, N(fp)

```

```
90     jal      mod
91     sw       v0, f(fp)
92
93     # c2 = -1
94     li       t0, -1
95     sw       t0, c2(fp)
96     # c2 <= 1
97 inner_for:
98     li       t0, 1
99     lw       t1, c2(fp)
100    ble      t1, t0, inner_for_body
101
102    # ++c1
103    lw       t0, c1(fp)
104    addi      t0, t0, 1
105    sw       t0, c1(fp)
106    b        outer_for
107
108 inner_for_body:
109    # c1 == 0
110    lw       t0, c1(fp)
111    seq      t1, t0, zero
112    # c2 == 0
113    lw       t0, c2(fp)
114    seq      t2, t0, zero
115    # c1 && c2
116    and      t0, t2, t1
117    bnez     t0, inner_loop_body_continue
118
119    # c = mod(j + c2, M)
120    lw       t0, j(fp)
121    lw       t1, c2(fp)
122    add      a0, t1, t0
123    lw       a1, M(fp)
124    jal      mod
125    sw       v0, c(fp)
126
127    # (a[f * M + c] == 1)
128    lw       t0, f(fp)
129    lw       t1, M(fp)
130    lw       t2, c(fp)
131    mul      t0, t0, t1
132    addu     t0, t0, t2
133    # access a[ ].
134    lw       t1, a(fp)
135    addu     t0, t1, t0
136    lbu      t0, 0(t0)
```

```
137     sll     t0, t0, 24 # Stay with lower byte.
138     sra     t0, t0, 24
139     li      t1, 1
140     seq     t0, t0, t1
141     beqz    t0, inner_loop_body_continue
142     # contador ++
143     lw      t0, contador(fp)
144     addi    t0, t0, 1
145     sw      t0, contador(fp)
146
147 inner_loop_body_continue:
148     # ++c2
149     lw      t0, c2(fp)
150     addi    t0, t0, 1
151     sw      t0, c2(fp)
152     b       inner_for
153
154     # -----
155 exit_function:
156     lw      v0, contador(fp)
157     # Stack frame unwinding.
158     lw      ra, RA(sp)
159     lw      fp, FP(sp)
160     lw      gp, GP(sp)
161     addu    sp, sp, STACK_SIZE
162     jr      ra
163
164 .end     vecinos
```