FACULTAD DE INGENIERIA
Universidad de Buenos Aires

[66.20] ORGANIZACIÓN DE COMPUTADORAS

TRABAJO PRÁCTICO 1

1ER 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

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, mostrandose 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` descripta 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 Assembly 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 utiizan 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 descripta 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

```
# Local and Temporary Area (LTA).
#define r 0
#define PADDING_LTA_0 r + 4

# Saved-registers area (SRA).
#define GP PADDING_LTA_0 + 4
#define FP GP + 4

# Caller ABA.
#define x FP + 4
#define m x + 4

#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

```
#define STACK_SIZE FP + 4
```

y toma el valor de 24 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

```
subu    sp,sp,STACK_SIZE
sw      fp,FP(sp)
.cprestore GP # Equivalent to     sw      gp, GP(sp).
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

```
sw      a0, x(fp)
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`:

```
add     v0, t0, a1
# ------------------------------
exit_function:
# Stack frame unwinding.
lw      fp, FP(sp)
lw      gp, GP(sp)
addu    sp, sp, STACK_SIZE
jr      ra
```

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

```
remu    t0,a0,a1
```

2.2. `vecinos()`

La función `vecinos()` es la siguiente

```
unsigned int vecinos(unsigned char *a, unsigned int i,
                     unsigned int j, unsigned int M,
                     unsigned int N)
{
    int contador = 0;
    int c1 = 0;
    int c2 = 0;
    int f = 0;
    int c = 0;

    for (c1 = -1; c1 <= 1; ++c1)
    {
```

```
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
7  # Local and Temporary Area (LTA).
8  #define contador            (ARG3 + 4)
9  #define c1                  (contador + 4)
10 #define c2                  (c1 + 4)
11 #define f                   (c2 + 4)
12 #define c                   (f + 4)
13 #define PADDING_LTA_0       (c + 4)
14
15 # Saved-registers area (SRA).
16 #define GP                  (PADDING_LTA_0 + 4)
17 #define FP                  (GP + 4)
18 #define RA                  (FP + 4)
19 #define PADDING_SRA0        (RA + 4)
20
21 # Caller ABA.
22 #define a                   (PADDING_SRA0 + 4)
23 #define i                   (a + 4)
24 #define j                   (i + 4)
25 #define M                   (j + 4)
```

```
26 #define N                    (M + 4)
27
28 #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()`.

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

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

y toma el valor de 56 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

```
subu    sp,sp,STACK_SIZE
sw      ra, RA(sp)
sw      fp, FP(sp)
.cprestore GP # Alternative: sw      gp, GP(sp)
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:

```
sw      a0, a(fp)
sw      a1, i(fp)
sw      a2, j(fp)
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`:

```
lw      v0, contador(fp)
lw      ra, RA(sp)
lw      fp, FP(sp)
lw      gp, GP(sp)
addu    sp, sp, STACK_SIZE
jr      ra
```

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

```
# f = mod(i + c1, N)
lw      t0, i(fp)
lw      t1, c1(fp)
add     a0, t1, t0
lw      a1, N(fp)
jal     mod
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

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

mientras que en ASM se debe realizar lo siguiente

```
# (a[f * M + c] == 1)
lw      t0, f(fp)
lw      t1, M(fp)
lw      t2, c(fp)
mul     t0, t0, t1
addu    t0, t0, t2
# access a[].
lw      t1, a(fp)
addu    t0, t1, t0
lbu     t0, 0(t0)
sll     t0, t0, 24 # Stay with lower byte.
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:

```
$ make
```

Para el segundo caso, ejecutar el siguiente comando

```
$ 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`

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

Los flags de compilación utilizados para la compilación de código C son

```
CFLAGS = -Wall -O0 -g
```

mientras que para la compilación de código ASM MIPS32 son

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

Por otro lado, se puede compilar el programa para realizar mediciones de tiempos de ejecución promedio de la función `vecinos()`, utilizando el comando

```
$ make measureTimeCFiles
```

si se quiere medir el tiempo utilizando los archivos en lenguaje C, o si no

```
$ make measureTimeSFiles
```

si se quiere medir el tiempo utilizando los archivos en lenguaje ASM MIPS32. En este modo de ejecución, se va a crear un archivo de texto `time_measurements.txt` en el directorio `src` con el promedio de las mediciones tomadas. Ver la sección de pruebas de tiempos de ejecución para una explicación más detallada.

Para eliminar todos los archivos generados, ejecutar

```
$ 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

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

y el resultado obtenido es

```
==1837==
==1837== HEAP SUMMARY:
==1837==     in use at exit: 0 bytes in 0 blocks
==1837==   total heap usage: 45 allocs, 45 frees, 56,728 bytes allocated
==1837==
==1837== All heap blocks were freed -- no leaks are possible
==1837==
==1837== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==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 primer 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:

```
------------------------------------------------------
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:

```
-----------------------------------------------------
TEST1: inexistent 'input' stream.
-----------------------------------------------------
Testing: ./conway 10 20 20 invalid
PASSED
  PROGRAM OUTPUT:
        ERROR: Can't open input stream.
-----------------------------------------------------
TEST11: no 'output' option arg.
-----------------------------------------------------
Testing: ./conway -o
PASSED
  PROGRAM OUTPUT:
        ./conway: option requires an argument -- 'o'
-----------------------------------------------------
TEST2: invalid 'out' stream.
-----------------------------------------------------
Testing: ./conway 10 20 20 glider -o .
PASSED
  PROGRAM OUTPUT:
        ERROR: Invalid output stream.
Testing: ./conway 10 20 20 glider -o ..
PASSED
  PROGRAM OUTPUT:
        ERROR: Invalid output stream.
Testing: ./conway 10 20 20 glider -o /
PASSED
  PROGRAM OUTPUT:
        ERROR: Invalid output stream.
Testing: ./conway 10 20 20 glider -o //
PASSED
  PROGRAM OUTPUT:
        ERROR: Invalid output stream.
-----------------------------------------------------
TEST21: not enough input arguments.
-----------------------------------------------------
Testing: ./conway 10
PASSED
  PROGRAM OUTPUT:
        ERROR: not enough input arguments.
Testing: ./conway 10 20
PASSED
  PROGRAM OUTPUT:
        ERROR: not enough input arguments.
Testing: ./conway 10 20 20
PASSED
  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 `videos`.

5. Pruebas de tiempo de ejecución

Como se mencionó en la sección sobre compilación y portabilidad, se puede compilar el programa para medir los tiempos de ejecución promedio de la función `vecinos()`. Para computar dicho tiempo se utiliza la función de biblioteca C `clock()` que retorna la cantidad de ticks de clock de procesador consumido por la sección del programa a probar. En particular, se utiliza solamente en la llamada a la función `vecinos()` dentro de la función `tablero_modificar()`.

En la tabla 3 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()`. Los comandos utilizados para ejecutar los programas fueron los siguientes

```
./conway 10 20 20 glider
./conway 10 20 20 pento
./conway 10 20 20 sapo
```

donde se utilizan 10 iteraciones del juego, en un tablero de 20x20.

t [us]	C	ASM
Glider	2.14	2.33
Pento	2.37	2.40
Sapo	2.30	2.20

Tabla 3: Tiempos promedio de ejecución de la función `vecinos()` para la versión en lenguaje C y lenguaje ASM MIPS32. El promedio se realiza sobre 4000 llamados a la función.

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

Se implementó el juego de la vida de Conway en lenguaje C y se programó la función de cálculo de vecinos en ASM MIPS32, como así también la función que calcula el módulo para contemplar un tablero toroidal. Se describió el stack creado por dichas funciones, y se desarrollaron las secciones de código más importantes. Por otro lado, se realizaron varios tests para probar el correcto funcionamiento del programa.

En particular fue de interés el test de tiempos de ejecución promedio. Se comprobó que no existió una mejora significativa entre utilizar la versión programada en C o en ASM MIPS32. Solamente en el caso de utilizar el caso “Sapo” fue donde hubo una leve mejora al utilizar la versión en ASM, posiblemente porque siempre se generan las 2 mismas imágenes como se puede ver en el video de muestra. Por lo tanto, en principio, y sin considerar otros aspectos, se puede concluir que a pesar de que ambas versiones producen tiempos de ejecución similares, no es necesario implementar funciones en lenguaje Assembly, ya que no reporta un beneficio en el tiempo de ejecución promedio, y además se necesitó de mayor trabajo para implementarla.

Por otra parte, se considera que la función realizada en Assembly, 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()` 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 no se puede apreciar diferencias significativas de tiempos promedio de ejecución entre las dos versiones implementadas. Es por este motivo que si el programa fue efectivamente realizado para funcionar con entradas de dichos órdenes de magnitud, no se encuentra razón por la que desarrollar funciones en Assembly.

Por último, si se utilizase el programa con valores de 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 Assembly, seguramente teniendo en cuenta la arquitectura de la CPU, y sobretodo el sistema de memoria cache. De todas formas como se comprobó empíricamente, utilizar lenguaje Assembly no garantiza que mejoren los tiempos de ejecución promedio.

Referencias

[1] Hennessy, J. L. - Patterson, D. A. - *Computer Architecture: A Quantitative Approach* - 3rd edition - Morgan Kaufmann - 2002.

[2] Patterson, D. A. - Hennessy, J. L. - *Computer Organization and Design: The Hardware/Software Interface* - 3rd edition - Morgan Kaufmann - 2004.

[3] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd 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* - 3nd edition - 1996.

[6] *GNU Make* - https://www.gnu.org/software/make/

[7] *GNU Gcc* - https://gcc.gnu.org/

[8] LaTeX- 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[1], escribiendo un programa portable que resuelva el problema descripto 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 TEX / 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].

[1] 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.

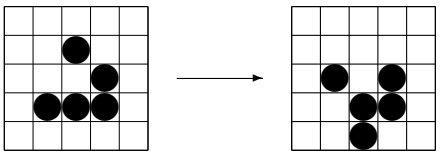

Figura 1: Ejemplo de transición entre estados

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 MxN, y escribirá i archivos .PBM [8] [2] representando i estados consecutivas del "Juego de la Vida" en una matriz de MxN, 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

[2] 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.

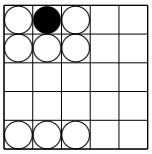

Figura 2: Ejemplo de mundo toroidal: la celda (0,1) y sus vecinos.

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 $10x10$, 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 [3].

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

[3] 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 [4]:

- 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 $20x20$, 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.

[4] 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.

[6] Juego de la Vida de Conway, 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

```
# ------------------------------------------------------------
#
# The source files must have .c extension.
# The object code must have .o extension.
# The header files must have .h extension.
#
# To compile, execute 'make'.
# To clean all the compilation files, issue the command
# 'make clean'.
#
# ------------------------------------------------------------
# List all the header and object files separated by a blank
# space.
_DEPS = params_t.h messages.h
_SRC1_c = conway.c tablero.c cmd_line_parser.c vecinos.c mod.c
_SRC1_S = conway.c tablero.c cmd_line_parser.c vecinos.S mod.S
# ------------------------------------------------------------
# Configuration.
CC = gcc
CFLAGS = -Wall -O0 -g
OUTPUT1 = conway
# ------------------------------------------------------------
all: $(OUTPUT1)

$(OUTPUT1): $(_SRC1_c)
    $(CC) $(CFLAGS) -o $@ $(_SRC1_c)

use_S_files: $(_SRC1_s)
    $(CC) $(CFLAGS) -o $(OUTPUT1) $(_SRC1_S) -mips32 -mlong32

measureTimeCFiles: $(_SRC1_c)
    $(CC) $(CFLAGS) -o $(OUTPUT1) $(_SRC1_c) -D ↩
       MEASURE_EXEC_TIMES

measureTimeSFiles: $(_SRC1_s)
    $(CC) $(CFLAGS) -o $(OUTPUT1) $(_SRC1_S) -D ↩
       MEASURE_EXEC_TIMES

.PHONY: clean use_S_files

clean:
    rm -f ./*.o ./*~ ./*.core ./*~
```

```
41      rm -f $(OUTPUT1)
```

C. Tests

C.0.1. run_tests.sh

```
#!/usr/bin/env bash
# ------------------------------------------------------------
#
# Script to test errors in the program arguments.
#
# To remove newlines from a textfile, use
# printf %s "$(cat file)" > file
#
# To print contents of a file, including control characters, do:
# oc -c file
#
# ------------------------------------------------------------

# Program name to test.
PROGRAM_NAME='./conway'

# Failed tests counter.
failedTests=0;

# Colors to be used.
RED="\e[31m";
GREEN="\e[32m";
CYAN="\e[96m";
YELLOW="\e[93m";
BOLD="\033[1m";
DEFAULT="\e[0m";

# Helper and formatting functions definitions.
function header() {
  echo -e "$CYAN------------------------------------------------------------$DEFAULT";
  echo -e "$CYAN$1$DEFAULT";
  echo -e "$CYAN------------------------------------------------------------$DEFAULT";
}

function msg_true() {
  echo -e "$GREEN\0PASSED \n $DEFAULT PROGRAM OUTPUT:\n\t$1";
}
```

```
38
39 function msg_false() {
40   echo -e "$RED\0FAILED \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
108     done
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" == "↩
           $PROGRAM_OUTPUT" ]]; then
131       msg_true "$PROGRAM_OUTPUT"
132     else
133       msg_false "$PROGRAM_OUTPUT"
134       failedTests=$(($failedTests+1));
135     fi
136
137     done
138 }
139
140 EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT=("ERROR: not enough input ↩
      arguments.")
141
142 function test21_parameter_output_no_argument(){
143   header "TEST21: not enough input arguments."
144
145     commands=("10"
146   "10 20"
147   "10 20 20")
148
149     for i in "${commands[@]}"
150     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" == "↩
           $PROGRAM_OUTPUT" ]]; then
157       msg_true "$PROGRAM_OUTPUT"
158     else
159       msg_false "$PROGRAM_OUTPUT"
160       failedTests=$(($failedTests+1));
161     fi
162
163     done
164 }
165
166 EXPECTED_OUTPUT_VALID_PARAMETERS=(
167 "Leyendo estado inicial...
168 Grabando ../output/glider001.pbm
169 Grabando ../output/glider002.pbm
170 Grabando ../output/glider003.pbm
```

```
Grabando ../output/glider004.pbm
Grabando ../output/glider005.pbm
Grabando ../output/glider006.pbm
Grabando ../output/glider007.pbm
Grabando ../output/glider008.pbm
Grabando ../output/glider009.pbm
Grabando ../output/glider010.pbm
Listo."
)

function test3_valid_parameters(){
  header "TEST3: all options with correct parameters."

    commands=(
  "10 20 20 glider")

    for i in "${commands[@]}"
    do

        msg_testing "$PROGRAM_NAME $i"

        PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)


    if [[ "$EXPECTED_OUTPUT_VALID_PARAMETERS" == "↩
       $PROGRAM_OUTPUT" ]]; then
      msg_true "$PROGRAM_OUTPUT"
    else
      msg_false "$PROGRAM_OUTPUT"
      failedTests=$(($failedTests+1));
    fi

    done
}
# -----------------------------------------------------------
# Run the tests.
# -----------------------------------------------------------
test1_parameter_input_inexistent_stream
test11_parameter_output_no_argument
test2_parameter_output_invalid_stream
test21_parameter_output_no_argument
test3_valid_parameters

header "Test suite ended."

if [[ $failedTests -eq $zero ]]; then
  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

```c
#ifndef CMD_LINE_PARSER__H
#define CMD_LINE_PARSER__H

#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "messages.h"
#include "params_t.h"

#ifndef STD_STREAM_TOKEN
#define STD_STREAM_TOKEN "-"
#endif

#ifndef VERSION
#define VERSION "1.0.0"
#endif

typedef enum output_states_
{
    outOK,
    outERROR
} output_state;

void option_iterations(const char *args, params_t *params);
void option_rows_size(const char *args, params_t *params);
void option_columns_size(const char *args,
                         params_t *params);
output_state validate_stream_name(char *streamName);
output_state option_input_file(char *arg, params_t *params);
output_state option_output_prefix(char *arg,
                                  params_t *params);
void option_version(void);
void option_help(char *arg);
output_state parse_cmd_line(int argc, char **argv,
                            params_t *params);
void imprimir_uso();
void manejar_comandos(char *opt);

#endif
```

D.0.2. mod.h

```
#ifndef MOD__H
#define MOD__H

extern int mod(int x, int m);

#endif
```

D.0.3. params_t.h

```
#ifndef PARAMS_T__H
#define PARAMS_T__H

#define MAX_SIZE_PREFIX 1024

typedef struct params_t
{
    int i;
    int N;
    int M;
    FILE *inputStream;
    FILE *outputStream;
    char *prefix;
} params_t;

#endif
```

D.0.4. tablero.h

```
#ifndef TABLERO__H
#define TABLERO__H

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef MEASURE_EXEC_TIMES
#include <time.h>
#endif

#include "messages.h"
#include "params_t.h"
#include "vecinos.h"

/* Mantener un número múltiplo de 10 */
#define ZOOM_ARCHIVO 30

typedef struct tablero
{
    unsigned char *tabla;
    int l;
    int h;
} tablero_t;

void tablero_eliminar(tablero_t *self);
tablero_t *tablero_crear(int length, int height);
int tablero_cargar_tablero(tablero_t *self,
                           params_t *params);
void tablero_modificar_estado_campo(tablero_t *self,
                                    int fila, int col);
void tablero_imprimir(tablero_t *self, int iteracion,
                      params_t *params);
int mod(int x, int m);
unsigned int vecinos_c(unsigned char *a, unsigned int i,
                       unsigned int j, unsigned int M,
                       unsigned int N);
unsigned int vecinos_mips(unsigned char *a, unsigned int i,
                          unsigned int j, unsigned int M,
                          unsigned int N);
tablero_t *tablero_modificar(tablero_t *self);
void tablero_eliminar(tablero_t *self);

```

```
45 #endif
```

D.0.5. vecinos.h

```
#ifndef VECINOS__H
#define VECINOS__H

#include "mod.h"

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

#endif
```

E. Código fuente

E.0.1. cmd_line_parser.c

```c
#include "cmd_line_parser.h"

void option_iterations(const char *args, params_t *params)
{
    int iterations;

    if (sscanf(args, "%d", &iterations) != 1 ||
        iterations <= 0)
    {
        fprintf(stderr, ERROR_INVALID_ITERATIONS);
        exit(EXIT_FAILURE);
    }

    params->i = iterations;
}

void option_rows_size(const char *args, params_t *params)
{
    int rows;

    if (sscanf(args, "%d", &rows) != 1 || rows <= 0)
    {
        fprintf(stderr, ERROR_INVALID_ROWS);
        exit(EXIT_FAILURE);
    }

    params->N = rows;
}

void option_columns_size(const char *args, params_t *params)
{
    int cols;

    if (sscanf(args, "%d", &cols) != 1 || cols <= 0)
    {
        fprintf(stderr, ERROR_INVALID_COLUMNS);
        exit(EXIT_FAILURE);
    }

    params->M = cols;
}

```

```
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 
```

```c
output_state option_output_prefix(char *arg,
                                  params_t *params)
{
    if (validate_stream_name(arg) == outERROR)
    {
        fprintf(stderr, ERROR_INVALID_OUTPUT_STREAM);
        exit(EXIT_FAILURE);
    }

    if (strcmp(arg, STD_STREAM_TOKEN) == 0)
    {
        params->outputStream = stdout;
        params->prefix = NULL;
    }
    else
    {
        params->outputStream = NULL;
        params->prefix = arg;
    }

    return outOK;
}

void option_version(void)
{
    fprintf(stderr, "%s\n", VERSION);

    exit(EXIT_SUCCESS);
}

void option_help(char *arg)
{
    fprintf(stderr, "Uso:\n");
    fprintf(stderr, " %s -h\n", arg);
    fprintf(stderr, " %s -V\n", arg);
    fprintf(stderr, " %s i M N inputfile [-o outputprefix]\n",
            arg);

    fprintf(stderr, "Opciones:\n");
    fprintf(stderr, " -h, --help\tImprime este mensaje.\n");
    fprintf(stderr,
            " -V, --version\tDa la versión del programa.\n");
    fprintf(stderr,
            " -o Prefijo de los archivos de salida.\n");

    fprintf(stderr, "Ejemplos:\n");
    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

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "cmd_line_parser.h"
#include "messages.h"
#include "params_t.h"
#include "tablero.h"

/*
For memory leaks checking, precede the program call with the
following commands:

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

*/

/*
Parámetros recibidos:
i -> cantidad de archivos a escribir
M -> coordenada x
N -> coordenada y
path -> nombre archivo */

#ifdef MEASURE_EXEC_TIMES
size_t times_called;
double mean_time;
#endif

int main(int argc, char *argv[])
{
    params_t params;

    if (parse_cmd_line(argc, argv, &params) == outERROR)
    {
        exit(EXIT_FAILURE);
    }

#ifdef MEASURE_EXEC_TIMES
#include "shared.h"
    times_called = 0;
    mean_time = 0;
    FILE *time_measurements_fp =
```

```
45          fopen("time_measurements.txt", "w");
46      if (time_measurements_fp == NULL)
47      {
48          fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);
49      }
50  #endif
51
52      tablero_t *tablero = tablero_crear(params.M, params.N);
53      tablero_cargar_tablero(tablero, &params);
54
55      int n = 0;
56      for (n = 1; n <= params.i; ++n)
57      {
58          tablero_imprimir(tablero, n, &params);
59          tablero = tablero_modificar(tablero);
60      }
61
62      tablero_eliminar(tablero);
63      fprintf(stdout, MSG_READY);
64
65  #ifdef MEASURE_EXEC_TIMES
66      fprintf(stdout, "Mean time: %.2f [us] for %d iterations.\n↩
            ", mean_time,times_called);
67      fprintf(time_measurements_fp, "Mean time: %.2f [us] for %d↩
            iterations.\n", mean_time,times_called);
68      if (time_measurements_fp != NULL)
69      {
70          fclose(time_measurements_fp);
71      }
72  #endif
73
74      return EXIT_SUCCESS;
75  }
```

E.0.3. mod.c

```c
#include "mod.h"

int mod(int x, int m)
{
    int r = x % m;
    return r < 0 ? r + m : r;
}
```

E.0.4. mod.S

```
#include <sys/regdef.h>

# Leaf function.

# Local and Temporary Area (LTA).
#define r 0
#define PADDING_LTA_0 r + 4

# Saved-registers area (SRA).
#define GP PADDING_LTA_0 + 4
#define FP GP + 4

# Caller ABA.
#define x FP + 4
#define m x + 4

#define STACK_SIZE FP + 4

.text
.align  2
.globl  mod
.ent    mod

.set    noreorder
.cpload t9
.set    reorder

mod:
    # Allocate memory for the stack.
    subu    sp,sp,STACK_SIZE
    # Save the callee-saved registers used by the caller in ↩
       the SRA.
    sw      fp,FP(sp)
.cprestore GP # Equivalent to     sw      gp, GP(sp).
    # We adopt the convention of using the frame pointer
    # as our index in the stack.
    move    fp,sp
    # Now we save the arguments that were loaded by the caller
    # in the area reserved by the callee.
    sw      a0, x(fp)
    sw      a1, m(fp)
    # ------------------------------
    ##############################
    # 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

```c
#include "tablero.h"

/* Para facilitar la tarea de mips, la grilla es realmente
una lista que es luego tratada como un grilla */
tablero_t *tablero_crear(int length, int height)
{
    /* TODO: check malloc and calloc errors. */
    tablero_t *tablero =
        (tablero_t *)malloc(sizeof(tablero_t));
    if (tablero == NULL)
    {
        fprintf(stderr, ERROR_MEMORY_REQUEST);
        return NULL;
    }
    tablero->tabla = (unsigned char *)calloc(
        height * length, sizeof(unsigned char));

    if (tablero->tabla == NULL)
    {
        fprintf(stderr, ERROR_MEMORY_REQUEST);
        return NULL;
    }
    tablero->l = length;
    tablero->h = height;
    return tablero;
}

/* Carga el tablero a partir del nombre de archivo pasado
por parámetro. Devuelve 0 si se cargó correctamente, otro en
caso de error.
*/
int tablero_cargar_tablero(tablero_t *self,
                           params_t *params)
{
    if (params->inputStream == NULL)
    {
        fprintf(stderr, ERROR_INVALID_INPUT_STREAM);
        exit(EXIT_FAILURE);
    }

    fprintf(stdout, MSG_READING_INIT_STATE);
    char buffer[4];
    while (fgets(buffer, 100, params->inputStream) != NULL)
    {
```

```
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 #ifdef MEASURE_EXEC_TIMES
138 #include "shared.h"
139
140             double mean_time_old = mean_time;
141             // Start measuring time
142             clock_t begin = clock();
143
144             unsigned int v =
145                 vecinos(&self->tabla[0], i, j, self->l, self->↩
                        h);
146
147             clock_t end = clock();
148             double elapsed_us =
149                 (double)(end - begin) * 1e6 / CLOCKS_PER_SEC;
150
151             mean_time =
152                 (mean_time_old * times_called + elapsed_us) /
153                 (times_called + 1);
154             times_called++;
155
156 #else
157             unsigned int v =
158                 vecinos(&self->tabla[0], i, j, self->l, self->↩
                        h);
159 #endif
160
161             int viva = self->tabla[i * self->l + j];
162             if ((v < 2) || (v > 3))
163             {
164                 tablero_nuevo->tabla[i * self->l + j] = 0;
165             }
166             if (viva && (v == 2 || v == 3))
167             {
168                 tablero_nuevo->tabla[i * self->l + j] = 1;
169             }
170             if (!viva && v == 3)
171             {
172                 tablero_nuevo->tabla[i * self->l + j] = 1;
173             }
174         }
175     }
176     /* Ahora lo elimino */
177     tablero_eliminar(self);
178     return tablero_nuevo;
179 }
180
181 /* Elimina el talbero creado */
```

```
182 void tablero_eliminar(tablero_t *self)
183 {
184     free(self->tabla);
185     free(self);
186 }
```

E.0.6. vecinos.c

```c
#include "vecinos.h"

unsigned int vecinos(unsigned char *a, unsigned int i,
                     unsigned int j, unsigned int M,
                     unsigned int N)
{
    int contador = 0;
    int c1 = 0;
    int c2 = 0;
    int f = 0;
    int c = 0;

    for (c1 = -1; c1 <= 1; ++c1)
    {
        f = mod(i + c1, N);

        for (c2 = -1; c2 <= 1; ++c2)
        {
            if (c1 == 0 && c2 == 0)
            {
                continue;
            }
            c = mod(j + c2, M);
            if (a[f * M + c] == 1)
            {
                contador++;
            }
        }
    }
    return contador;
}
```

E.0.7. vecinos.S

```
#include <sys/regdef.h>

# Non-leaf function.

# Argument building area (ABA).
#define ARG0                (0)
#define ARG1                (ARG0 + 4)
#define ARG2                (ARG1 + 4)
#define ARG3                (ARG2 + 4)

# Local and Temporary Area (LTA).
#define contador            (ARG3 + 4)
#define c1                  (contador + 4)
#define c2                  (c1 + 4)
#define f                   (c2 + 4)
#define c                   (f + 4)
#define PADDING_LTA_0       (c + 4)

# Saved-registers area (SRA).
#define GP                  (PADDING_LTA_0 + 4)
#define FP                  (GP + 4)
#define RA                  (FP + 4)
#define PADDING_SRA0        (RA + 4)

# Caller ABA.
#define a                   (PADDING_SRA0 + 4)
#define i                   (a + 4)
#define j                   (i + 4)
#define M                   (j + 4)
#define N                   (M + 4)

#define STACK_SIZE          (PADDING_SRA0 + 4)

.text
.align  2
.globl  vecinos
.ent    vecinos
# Debugger metadata.
.frame  fp,STACK_SIZE,ra
.set    noreorder
.cpload t9
.set    reorder

vecinos:
```

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

```
90      sw      v0, f(fp)
91
92      # c2 = -1
93      li      t0, -1
94      sw      t0, c2(fp)
95      # c2 <= 1
96  inner_for:
97      li      t0, 1
98      lw      t1, c2(fp)
99      ble     t1, t0, inner_for_body
100
101     # ++c1
102     lw      t0, c1(fp)
103     addi    t0, t0, 1
104     sw      t0, c1(fp)
105     b       outer_for
106
107 inner_for_body:
108     # c1 == 0
109     lw      t0, c1(fp)
110     seq     t1, t0, zero
111     # c2 == 0
112     lw      t0, c2(fp)
113     seq     t2, t0, zero
114     # c1 && c2
115     and     t0, t2, t1
116     bnez    t0, inner_loop_body_continue
117
118     # c = mod(j + c2, M)
119     lw      t0, j(fp)
120     lw      t1, c2(fp)
121     add     a0, t1, t0
122     lw      a1, M(fp)
123     jal     mod
124     sw      v0, c(fp)
125
126     # (a[f * M + c] == 1)
127     lw      t0, f(fp)
128     lw      t1, M(fp)
129     lw      t2, c(fp)
130     mul     t0, t0, t1
131     addu    t0, t0, t2
132     # access a[].
133     lw      t1, a(fp)
134     addu    t0, t1, t0
135     lbu     t0, 0(t0)
136     sll     t0, t0, 24 # Stay with lower byte.
```

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