FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1: Conjunto de Instrucciones MIPS

6620 - Organización de Computadoras
Primer cuatrimestre de 2020

Alumno	Padrón	Email
Maximiliano Yacobucci	93321	myacobucci@fi.uba.ar
Federico del Mazo	100029	fdelmazo@fi.uba.ar
Ariel Vergara	97010	avergara@fi.uba.ar

Resumen

En este trabajo práctico se realizará una versión del juego de la vida de Conway con el objetivo de comprender de una mejor manera el concepto de ABI y el conjunto de instrucciones MIPS32.

1. Introducción

En el siguiente trabajo práctico se realizará una versión del juego de la vida de Conway en su mayor parte en lenguaje C excepto una función, la cual se efectuará en assembler MIPS32. Para simular el entorno de desarrollo de una máquina MIPS utilizaremos el programa `qemu`.

El juego de la vida de Conway se basa en una grilla la cual contiene celdas que pueden estar vivas o muertas (encendidas o apagadas). Esta matriz evoluciona con el correr de los estados siguiendo estas reglas:

- Si una celda apagada tiene exactamente tres vecinos encendidos, su estado siguiente será encendida
- Si una celda prendida tiene menos de dos o más de tres vecinos encendidos, su estado siguiente será apagada
- En cualquier otro caso, la celda mantendrá su estado

Si bien el juego de la vida originalmente ideado por John Conway estaba pensado con una matriz infinita (lo cual hace que sea Turing completo), esto es irrealizable en la computadora (ya que la memoria de una computadora es finita). Es por eso que hay que superar este obstaculo, usando una matriz finita.

Al no ser una matriz infinita, se deben tener en cuenta los extremos de la matriz, qué reglas establecer para dichas celdas. Para ello se seguirá la hipótesis del mundo toroidal que indica que la fila 0 tendrá como vecino superior a la fila M-1, mientras que la columna 0 tendrá como vecino de la izquierda a la columna N-1, y viceversa.

El programa será capaz de ejecutar tantos estados del juego de la vida como el usuario lo indique, partiendo un estado inicial el cual también se ingresará mediante un archivo. También se deberá indicar el tamaño de la matriz, indicando la cantidad de filas y de columnas al llamar al programa.

En caso de éxito, el programa devolverá varios archivos .PBM, cada uno correspondiente a cada estado de la matriz. A su vez se brindará un archivo `.gif` donde se observará el correr de los diferentes estados, esto será realizado utilizando el programa `ffmpeg`.

2. Diseño e Implementación

2.1. Análisis de la línea de comandos

Las opciones válidas ingresadas por líneas de comandos serán:

- `./conway -h`
- `./conway -V`
- `./conway i M N inputfile [-o outputprefix] [-p]`

En el primer caso se mostrará un mensaje de ayuda con un ejemplo de cómo ejecutar el programa.

La segunda opción de comando devolverá por pantalla el número de versión del programa que se está ejecutando.

El tercer comando es el que efectivamente corre el programa en sí. Como se puede ver acepta diferentes parámetros, los cuáles se detallan a continuación:

- i: cantidad de iteraciones del juego de la vida a ejecutar
- M: cantidad de filas
- N: cantidad de columnas
- inputfile: nombre del archivo que contiene las celdas encendidas en el estado inicial
- outputprefix: prefijo del nombre de los archivos pbm de salida del programa (este parámetro es opcional, y su valor por defecto es el mismo que el input)
- p: de ser recibido este parámetro, el programa mostrará indefinidamente estados del juego, en vez de guardar archivos pbm

Es importante destacar que el formato del archivo inputfile esperado serán las coordenadas de las celdas encendidas, indicadas una por línea insertando en primer lugar el número de fila y luego la columna dejando un espacio entremedio.

2.2. Desarrollo del Código Fuente

Como se ha explicado anteriormente, el programa está hecho en lenguaje C con la excepción de una función, la cual está programada en Assembler MIPS. Dicha función se llama "vecinosz es la encargada de calcular cuántos vecinos encendidos posee una celda:

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

El primer parámetro a será un puntero a la posición [0, 0] de la matriz. i y j serán la fila y la columna de la celda de la que queremos saber cuántos vecinos encendidos tiene. M y N serán la cantidad total de filas y columnas que posee la matriz.

Esta función "vecinos"llama internamente a otra función programada en Assembler MIPS, llamada "tiene_vecino", la cual indica si hay o no un vecino en la celda recibida:

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

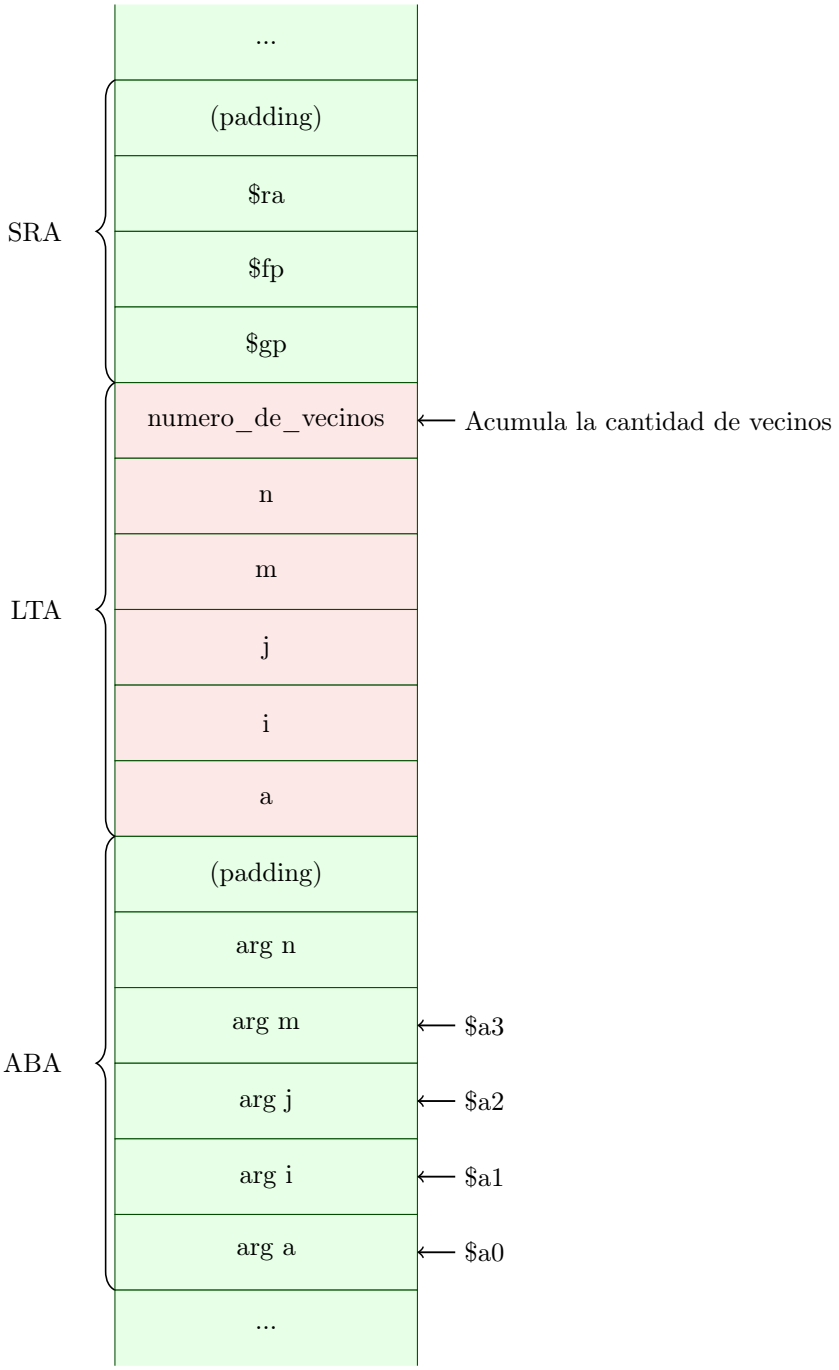
Los parámetros son los mismos que "vecinos", la diferencia radica en que la celda indicada por los parámetros i y j representan una celda vecina a la indicada a la función "vecinos". Esta se llama 8 veces, una por cada celda vecina.

En cuanto al código en lenguaje C se ha separado todo lo relacionado exclusivamente a la matriz en el archivo **matrix.c** con su correspondiente **matrix.h**. El programa principal se encuentra en el archivo **conway.c**.

2.3. Diagramas de stack

2.3.1. vecinos

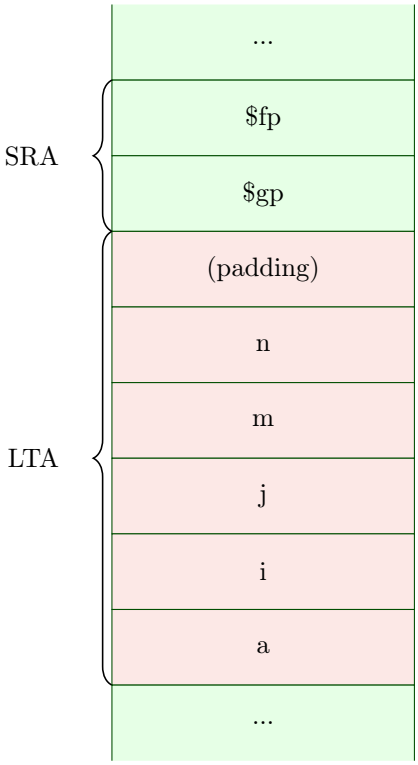
Esta función corresponde a una función no hoja dado que internamente llama a la función "tiene_vecino". Su diagrama de stack es el siguiente:



Cabe destacar que, según lo determinado por la ABI, sólo reserva el espacio para su ABA y carga únicamente el quinto argumento de la función "tiene_vecino.[en] el mismo. Los demás argumentos los pasa a través de los registros $a0 - $a3 y la función "tiene_vecino"se encarga de cargarlos en el ABA mostrado.

2.3.2. tiene_vecino

Esta función es una función hoja, por lo que no guarda el $ra en su SRA y no define un ABA. Su diagrama es el siguiente:



3. Proceso de Compilación

El programa posee un archivo Makefile que se encarga de la compilación y ejecución desde un alto nivel (sin importar la máquina). Sus opciones son:

- **make conway**: compila los archivos fuente y crea el ejecutable
- **make**: ejecutar el programa con datos iniciales predefinidos
- **make clean**: limpia todos los archivos generados por el make

Para lograr esto, la regla de compilación (`make conway`) primero intenta compilar como si fuese una arquitectura MIPS, y de no lograrlo, recae en la regla general para compilar en C.

De querer una compilación mas granulada, se tiene, dependiendo de la arquietctura:

- `gcc conway.c matrix.c vecinos.S tiene_vecino.S -DUSE_MIPS -o conway`
- `gcc conway.c matrix.c -o conway`

4. Portabilidad

El programa efectuado completamente en lenguaje C es posible ejecutarlo en cualquier sistema operativo al no utilizar bibliotecas o funciones particulares de un sistema operativo. Sin embargo, el makefile posee instrucciones de compilación propias de UNIX por lo que sólo se podrá correr en dicho sistema.

En este caso se han realizado pruebas en una arquitectura MIPS emulada. Se ha emulado una máquina con sistema operativo NetBSD utilizando el programa de emulación GXemul.

5. Casos de Prueba

A continuación se mostrarán caso de prueba del programa.

5.1. Línea de comandos

Ejecutamos la aplicación sin parámetros.

```
$ ./conway
Invalid arguments. Use conway -h to see valid examples.
```

Ejecutamos el programa solo con un parámetro.

```
$ ./conway 1
Invalid arguments. Use conway -h to see valid examples.
```

Ejecutamos el programa sin indicar el archivo input.

```
$ ./conway 10 10 10
Invalid arguments. Use conway -h to see valid examples.
```

Ejecutamos el programa con más parámetros de lo esperado.

```
$ ./conway 10 10 10 10 20 54 85
Invalid arguments. Use conway -h to see valid examples.
```

Ejecutamos el programa con un número de corridas negativo

```
$ ./conway -1 10 10 glider
./conway: invalid option -- '1'
The arguments are not positive numbers
```

Ejecutamos el programa con una matriz de tamaño inválido

```
$ ./conway 10 10 -10 glider
./conway: invalid option -- '1'
./conway: invalid option -- '0'
The arguments are not positive numbers
```

5.2. Funcionamiento de la aplicación

Acá mostramos el caso de prueba que se indica en el enunciado:

```
conway 5 10 10 glider -o estado
```

```
Reading initial state
Saving state estado_000.pbm
Saving state estado_001.pbm
Saving state estado_002.pbm
Saving state estado_003.pbm
Saving state estado_004.pbm
Done
```

También se lo puede ejecutar sin indicar el nombre de los archivos de estados. En este caso tomará el nombre del archivo de entrada.

```
conway 5 10 10 glider
```

```
Reading initial state
Saving state glider_000.pbm
Saving state glider_001.pbm
Saving state glider_002.pbm
Saving state glider_003.pbm
Saving state glider_004.pbm
Done
```

A su vez el programa cuenta con una versión interactiva indicando la opción `-p`.
`conway 5 10 10 glider -p`

```
Reading initial state
Entering interactive mode:
 Press ENTER to see the next state
 Press ESC followed by ENTER to finish execution


0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Corremos el programa con 10 iteraciones para una matriz de 20x20 para los siguientes archivos (observando los estados 1, 5 y 10):

- **glider**: las celdas encendidas comienzan a descender en diagonal con el correr de los estados

  ```
  conway 10 20 20 glider
  ```

- **pento**: las celdas encendidas inicialmente concentradas una al lado de la otra se van expandiendo formando formas similares a figuras geométricas

  ```
  conway 10 20 20 pento
  ```

- **sapo**: las celdas encendidas van cambiando entre solo dos estados posibles como se puede ver en las imágenes

```
conway 10 20 20 sapo
```

6. Conclusión

En este trabajo práctico pudimos aprender mucho acerca de la integración, la portabilidad y de la historia de diversos conceptos de las ciencias de la computación en general. Por un lado, tenemos el juego de Conway, una idea con un detrás matemático, diseñada en 1970, inspirada en las ideas de John Von Neuman y Alan Turing, 30 años antes. Esta idea pudo ser llevada a cabo en un entorno Unix (1970) en el lenguaje C (1971), con funciones en el lenguaje assembly (1949) de la arquitectura MIPS (1985). Todo esto se logró compilar y empaquetar a un formato netpbm originalmente diseñado para poder envíar archivos por email.

Es irónico pensar que lo más sofisticado de todo este sistema armado es el pre-informe escrito en Markdown, un lenguaje de marcado diseñado hace apenás más de 15 años. Sin contar esto, todo el trabajo práctico podría haberse hecho hace más de 30 años con los mismos resultados.

El trabajo nos enseño la importancia, el cuidado y el detalle con el que se tiene que programar cuando se lidia en assembly, teniendo muy en cuenta el orden de los parametros, la memoria pedida y todos los recursos que hoy por hoy damos por hecho en un lenguaje de alto nivel.

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.

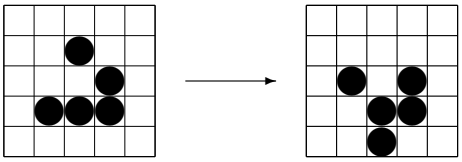


Figura 1: Ejemplo de transición entre estados

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.

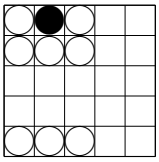


Figura 2: Ejemplo de mundo toroidal: la celda (0,1) y sus vecinos.

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 contenter 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