Trabajo Práctico 1
Conjunto de instrucciones MIPS

Tomás Lubertino, *Padrón 95140*
tomas.lubertino@gmail.com

Maximiliano Suppes, *Padrón 96938*
maxisuppes@gmail.com

Zuretti Agustin, *Padrón 95605*
zurettiagustin@gmail.com

1er. Cuatrimestre de 2020
86.37 / 66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

El objetivo de este trabajo práctico consiste en familiarizarse con el conjunto de instrucciones MIPS 32, para lo cual se desarrollo una versión del "Juego de la Vida"de John Conway en lenguaje C, donde la función **vecinos** está codificada en assembly MIPS 32.
Dicho juego consta de una grilla infinita, en donde en cada una de las 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 grilla o matriz evolucionara en el tiempo en pasos o estados discretos, y en cada estado sus celdas variaran segun las siguientes normas:

- 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 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.

El programa sera capaz de ejecutar tantos estados del juego de la vida como el usuario lo indique, partiendo de un estado inicial el cual también se ingresara mediante un archivo. También se deberá indicar el tamaño de la matriz, indicando cantidad de filas y columnas al llamar al programa.
En caso de éxito, el programa devolverá varios archivos de formato **.pbm**, cada uno correspondiente a cada estado de la matriz. A su vez se brindara un archivo **.mov** donde se observara el correr de los diferentes estados, el cual sera generado utilizando el programa **ffmpeg**.

2. Diseño e implementación

Las opciones validas ingresadas por lineas de comandos seran:

- ./conway -h
  Muestra un mensaje de ayuda con un ejemplo.
- ./conway -V
  Muestra el numero de version del programa.
- ./conway i M N inputfile [-o outputprefix]
  Se ejecuta el programa.

Como se puede ver en el ultimo comando, se aceptan diferentes parámetros:

- 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).

3. Desarrollo

A continuación se ilustra el diagrama del stack de la función **vecinos**, respetando la ABI de la cátedra.

3.1. Stack función vecinos (Leaf)

Distancia $sp	Referencia
40	$fp
36	$gp
32	$t7
28	$t6
24	$t5
16	$t4
12	$t3
08	$t2
04	$t1
00	$t0

4. Proceso de compilación y ejecución

El programa posee un archivo Makefile que se encarga de la compilación y ejecución desde un alto nivel. Las opciones son:

- make conway: compila los archivos y genera el ejecutable.
- make: ejecuta el programa con datos predefinidos.
- make clean: elimina todos los archivos generados por el make.

La primera opción de compilación va a intentar compilar como si fuese una arquitectura MIPS y en caso de no lograrlo, lo compilara como cualquier otro programa en C.

Otras opciones de compilación, dependiendo de la arquitectura, son:

```
$ gcc -Wall -g -o conway main.c
```

```
$ gcc -Wall -g -o conway main.c neighbors.S
```

Para correr el programa junto con un archivo de prueba, se debe utilizar el siguiente comando:

```
$ ./conway i m n glider|sapo|penta
```

- i es la cantidad de iteraciones
- m es la cantidad de filas de la matriz
- n es la cantidad de filas de la matriz
- glider, sapo o pento segun la grilla de inicio predilecta.

```
./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.
```

5. Portabilidad

Como el programa no utiliza librerías o funciones particulares de un sistema operativo, es posible ejecutarlo en cualquier sistema. Mas allá de esto, el archivo Makefile posee instrucciones propias de UNIX por lo que solo se podrá ejecutar en dicho sistema. Para este ultimo caso, se han realizado pruebas en un entorno con sistema operativo NetBSD utilizando el emulador GXemul.

6. Código fuente en MIPS

Una vez definida la ABI, se procedió a la codificación de dicha función en assembly MIPS32. Para su compilación y ejecución utilizamos el Debian MIPS (QEMU), y el gdb para debuggear la aplicación con el fin de verificar su funcionalidad y corregir posibles errores.

6.1. Función vecinos

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

// SRA
#define STACK_SIZE 40
#define SRA_GP 36
#define SRA_FP 32

// LTA
#define LTA_a 0
#define LTA_i 4
#define LTA_j 8
#define LTA_M 12
#define LTA_N 16

// Caller ABA
#define ABA_a 40
#define ABA_i 44
#define ABA_j 48
#define ABA_M 52
#define ABA_N 56

.text
.align 2
.globl neighbors
.ent neighbors

neighbors:
// To be position independent
.frame fp, STACK_SIZE, ra
.set noreorder
.cpload t9
.set reorder

// Creating stack
subu sp, sp, STACK_SIZE
sw gp, SRA_GP(sp)
sw fp, SRA_FP(sp)
move fp, sp

// Loading parameters
sw a0, ABA_a(fp) # matrix pointer
sw a1, ABA_i(fp) # i
sw a2, ABA_j(fp) # j
sw a3, ABA_M(fp) # M
lw t0, ABA_N(fp) # N

// Storing in LTA
sw t0, LTA_N(fp)

lw t1, ABA_a(fp)
sw t1, LTA_a(fp)
```

```
lw t2, ABA_i(fp)
sw t2, LTA_i(fp)

lw t3, ABA_j(fp)
sw t3, LTA_j(fp)

lw t4, ABA_M(fp)
sw t4, LTA_M(fp)

// Defining variables
li t0, 0                # living_neighbors = t0 = 0

addi t1, t2, -1         # t1 = i - 1
div t1, t4              # HO = (i-1) % M
mfhi t1                 # upPosition = t1 = (i-1) % M

lw t6, LTA_i(fp)        # t6 = i
addi t2, t6, 1          # t2 = i + 1
div t2, t4              # HO = (i+1) % M
mfhi t2                 # downPosition = t2 = (i+1) % M

lw t6, LTA_j(fp)        # t6 = j
lw t7, LTA_N(fp)        # t7 = N
addi t3, t6, -1         # t3 = j - 1
div t3, t7              # HO = (j-1) % N
mfhi t3                 # leftPosition = t3 = (j-1) % N

addi t4, t6, 1          # t4 = j + 1
div t4, t7              # HO = (j+1) % N
mfhi t4                 # rightPosition = t4 = (j+1) % N

columns_edge_case:
bne t6, 0, rows_edge_case  # if (j != 0) go to ciclo
addi t3, t7, -1         # leftPosition = N - 1

rows_edge_case:
lw t6, LTA_i(fp)        # t6 = i
lw t7, LTA_M(fp)        # t7 = M
bne t6, 0, up_and_left  # if (i != 0) go to ciclo
addi t1, t7, -1         # upPosition = M - 1

up_and_left:
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
lw t7, LTA_N(fp)        # t7 = N
mul t5, t1, t7          # index = t5 = upPosition * N
add t5, t5, t3          # index = t5 = upPosition * N + leftPosition
addu t5, t5, t6         # index = t5 = puntero a (upPosition, leftPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[upPosition, leftPosition]
bne t5, t7, up
addi t0, t0, 1          # living_neighbors += 1

up:
lw t6, LTA_j(fp)        # t6 = j
lw t7, LTA_N(fp)        # t7 = N
mul t5, t1, t7          # index = t5 = upPosition * N
add t5, t5, t6          # index = t5 = upPosition * N + j
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (upPosition, j)
li t7, '-'              # t7 = '-'
```

```
lbu t5, (t5)            # t5 = matrix[upPosition, j]
bne t5, t7, up_and_right
addi t0, t0, 1          # living_neighbors += 1

up_and_right:
lw t7, LTA_N(fp)        # t7 = N
mul t5, t1, t7          # index = t5 = upPosition * N
add t5, t5, t4          # index = t5 = upPosition * N + rightPosition
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (upPosition, rightPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[upPosition, rightPosition]
bne t5, t7, left
addi t0, t0, 1          # living_neighbors += 1

left:
lw t6, LTA_i(fp)        # t6 = i
lw t7, LTA_N(fp)        # t7 = N
mul t5, t6, t7          # index = t5 = i * N
add t5, t5, t3          # index = t5 = i * N + leftPosition
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (i, leftPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[i, leftPosition]
bne t5, t7, right
addi t0, t0, 1          # living_neighbors += 1

right:
lw t6, LTA_i(fp)        # t6 = i
lw t7, LTA_N(fp)        # t7 = N
mul t5, t6, t7          # index = t5 = i * N
add t5, t5, t4          # index = t5 = i * N + rightPosition
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (i, rightPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[i, rightPosition]
bne t5, t7, down_and_left
addi t0, t0, 1          # living_neighbors += 1

down_and_left:
lw t7, LTA_N(fp)        # t7 = N
mul t5, t2, t7          # index = t5 = downPosition * N
add t5, t5, t3          # index = t5 = downPosition * N + leftPosition
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (downPosition, leftPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[downPosition, leftPosition]
bne t5, t7, down
addi t0, t0, 1          # living_neighbors += 1

down:
lw t6, LTA_j(fp)        # t6 = j
lw t7, LTA_N(fp)        # t7 = N
mul t5, t2, t7          # index = t5 = downPosition * N
add t5, t5, t6          # index = t5 = downPosition * N + j
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (downPosition, j)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[downPosition, j]
bne t5, t7, down_and_right
addi t0, t0, 1          # living_neighbors += 1
```

```
down_and_right:
lw t7, LTA_N(fp)        # t7 = N
mul t5, t2, t7          # index = t5 = downPosition * N
add t5, t5, t4          # index = t5 = downPosition * N + rightPosition
lw t6, LTA_a(fp)        # t6 = puntero a (0,0)
addu t5, t5, t6         # index = t5 = puntero a (downPosition, rightPosition)
li t7, '-'              # t7 = '-'
lbu t5, (t5)            # t5 = matrix[downPosition, rightPosition]
bne t5, t7, return
addi t0, t0, 1          # living_neighbors += 1

return:
move v0, t0

// Unwinding stack
lw fp, SRA_FP(sp)
lw gp, SRA_GP(sp)
addu sp, sp, STACK_SIZE
jr ra
.end neighbors
```

7. Pruebas

A continuación se mostraran algunos casos de prueba del programa.

7.1. Consola

Si ejecutamos el programa sin parámetros.

```
./conway
Argumentos invalidos. Usa conway -h para ver ejemplos validos.
```

Si ejecutamos el programa con un solo parámetro.

```
./conway 10
Argumentos invalidos. Usa conway -h para ver ejemplos validos.
```

Si ejecutamos el programa sin el archivo de entrada.

```
./conway 10 20 20
Argumentos invalidos. Usa conway -h para ver ejemplos validos.
```

Si ejecutamos el programa con parámetros de más.

```
./conway 10 20 20 45 54 88 99
Argumentos invalidos. Usa conway -h para ver ejemplos validos.
```

Si ejecutamos el programa con un numero negativo para las iteraciones.

```
./conway -10 20 20 glider
./conway: invalid option -- '1'
./conway: invalid option -- '0'
Los argumentos no son numeros positivos.
```

Si ejecutamos el programa con una matriz de formato invalido.

```
./conway 10 20 -20 glider
./conway: invalid option -- '2'
./conway: invalid option -- '0'
Los argumentos no son numeros positivos.
```

7.2. Funcionamiento de la aplicación

Algunos casos de prueba que se indican en el enunciado:

```
./conway 5 10 10 glider -o estado
Leyendo estado inicial...
Grabando estado_001.pbm
Grabando estado_002.pbm
Grabando estado_003.pbm
Grabando estado_004.pbm
Grabando estado_005.pbm
Listo.
```

```
./conway 5 10 10 glider
Leyendo estado inicial...
Grabando glider_001.pbm
Grabando glider_002.pbm
```

```
Grabando glider_003.pbm
Grabando glider_004.pbm
Grabando glider_005.pbm
Listo.
```

A continuación se muestran Corridas de prueba para diez iteraciones, en una matriz de 20x20, de los archivos de entrada glider, sapo y pento.

```
./conway 10 20 20 glider
```

1 5 10

```
./conway 10 20 20 sapo
```

1 5 10

```
./conway 10 20 20 pento
```

1 5 10

8. Conclusión

Este trabajo, por un lado, nos ayudó a familiarizarnos con la programación en assembly MIPS32, lo que nos parece una herramienta muy útil tanto para programar, como para entender empezar a entender el funcionamiento de los procesadores.
También nos dimos cuenta que el código assembly puede tener mejor o peor performance respecto a las versiones compiladas según como se encare la implementación, y estos análisis nos sirven como una herramienta de medida para saber si nuestro código puede optimizarse aún más.

Referencias

[1] GXemul, http://gavare.se/gxemul/.

[2] QEMU, https://www.qemu.org/.

[3] Debian, the Universal Operating System, https://www.debian.org/.

[4] John Horton Conway, [1937-2020], https://en.wikipedia.org/wiki/John_Horton_Conway.

[5] Juego de la Vida de Conway, http://es.wikipedia.org/wiki/Juego_de_la_vida.

[6] Máquina de Turing, implementada en el Juego de la Vida de Conway, http://rendell-attic.org/gol/tm.htm.

[7] http://netpbm.sourceforge.net/doc/pbm.html.

[8] https://www.ffmpeg.org/.

[9] https://lukecyca.com/2013/stop-motion-with-ffmpeg.html.