Organización de Computadoras - TP 1

Martín Cohen, Padron: 100812 martincohen98@gmail.com

1er Cuatrimestre 2020

Resumen

En este trabajo practico se programaron dos versiones del juego de la Vida, una de ellas escrita en C y otra con una función en Assembly, con el objetivo de familiarizarse con la forma de escribir en el Assembly de MIPS, y las herramientas y lógica de escribir en Assembly.

1 Introducción

El trabajo practico tiene como objetivo lograr familiarizarse con la programación en el lenguaje Assembly de MIPS32. Esto incluye el uso y sintaxis de sus instrucciones y conceptos como el la Call Convention y su ABI.

Para esto es necesario hacer una implementación de el Juego de la Vida de Conway. Se pide hacer una implementación totalmente portable en C, y otra con una de las funciones claves del proyecto implementada en Assembly de MIPS32. Este juego modela una progresión en las que una matriz de celdas con organismos vivos se van ocupando y desocupando dependiendo de su entorno. A esto se le agrega la dificultad de que si se llega al borde de la matriz se considera adyacente el bloque del borde opuesto que está a la misma altura.

2 Diseño e Implementación

2.1 Análisis de la Linea de Comandos

En la linea de comandos los parametros deben ser enviados de la siguiente manera:

```
./conway i M N inputfile
```

Siendo i la cantidad de iteraciones, M y N el tamaño de la matriz MxN a utilizarse, e inputfile el archivo donde se detallan las cordenadas de los puntos iniciales del programa.

Los flags que pueden utilizarse son:

```
-h, --help Imprime este mensaje.
-V, --version Da la versión del programa.
-o Prefijo de los archivos de salida.
```

2.2 Desarrollo del Código Fuente

La implementación portable y la implementación de MIPS32 comparten una gran parte de su código. De hecho, todo menos la función vecinos, su declaración y el mensaje del flag -v son exactamente iguales. Debido a esto hay **dos** archivos conway.c, uno para la version C y uno para la version MIPS32, y un archivo vecinos.S.

Cualquier operación exitosa es notificada al usuario a través del standard output. Esto tiene el propósito de que en caso de que haya un error y se notifique por el standard error se pueda confirmar que no hay otros problemas con los otros parámetros.

Asumo que los nombres de los archivos dados por el flag -o no van a ser extremadamente altos, ya que uso un buffer de tamaño fijo (100 caracteres) para el nombre.

Ambas implementaciones tienen una visualización por consola al cargar la matriz del archivo y luego de cada iteración. Esto sirve para que no sea necesario abrir los archivos para poder ver el estado del juego.

En la implementación de la función vecinos podría haber optado en usar un loop doble que recorriera los casilleros rodeando al cual se la llama la función. Sin embargo, tomando en cuenta la especificación de los bordes, una solución más simple es inicialmente fijar las filas y columnas adyacentes al casillero y recorrerlos de manera estructurada en vez de iterativa. Esto resulta en código más largo pero menos complejo.

En la versión de Assembly de esa misma función, opte por usar una variable en memoria menos. Como en la versión de C solo la usaba para acumular la cantidad de vecinos utilicé un registro temporal en vez de alocar memoria. Esto mejora el rendimiento ya que se necesitan menos accesos a memoria sin aumentar la complejidad del código.

2.2.1 ABI

El ABI del lenguaje Assembly de MIPS32 está dividido en tres secciones. Estas son:

- El SRA (Saved Register Area) se usa para guardar registros cuyo valor debe ser devuelto al original al retornar de la función.
- El LTA (Local and Temporary Area) se utiliza para guardar variables locales que se utilizan en la función.
- El ABA (Argument Building Area) se utiliza para guardar los argumentos de la función. Si hay un argumento, es necesario dejar espacio suficiente como para que haya cuatro argumentos. Se agrega más espacio de ser necesario.

Estas tres secciones deben tener un tamaño múltiplo de 8 bytes. De lo contrario se debe agregar padding para compensar.

Dicho esto, el stack frame de la función vecinos en Assembly Esta formado de la siguiente mantera:

SRA	gp fp
LTA	columnaSiguiente columnaAnterior filaSiguiente columnaSiguiente
ABA	padding ancho alto j i matriz

Notar como en el ABA se dejan 4 bytes de padding como se mencionó anteriormente.

3 Proceso de Compilación

El proceso de compilación es dentro de todo simple. Para la versión en C, lo único que se debe hacer es llamar a gcc con el archivo conway.c de la version C de la siguiente manera:

```
gcc conway.c -o conway
```

Con esto ya se tiene el ejecutable listo para correr el programa.

La versión de MIPS32 es un poco mas compleja en principio, pero esencialmente es similar. Al tener una función programada en Assembly especifico de MIPS32 no se puede simplemente compilar en un linux corriendo en una computadora con arquitectura diferente. Para esto se necesita correr una maquina virtual corriendo una distribucion de linux de MIPS32 y compilarlo en la misma.

Una vez que ya tenemos esto la compilación vuelve a ser trivial. Se realiza de la siguiente manera, utilizando la versión MIPS32 de conway.c.

```
gcc conway.c vecinos.S -o conway
```

Es importante notar que para ser debuggeado el programa debe ser compilado de una manera distinta. Se le deben agregar los flags siguientes a ambas versiones.

```
gcc conway.c vecinos.S -g -O0 -o conway
```

Estos flags mantienen la información de debuggeo al compilar y me aseguran que no se va a optimizar el código de ninguna manera respectivamente.

4 Casos de Prueba

En esta sección muestro screenshots de las ejecuciones pedidas en el enunciado y ejecuciones importantes que mostrar, como la ejecución del flag -help:

```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./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.
```

Ejecuciones de -v:

```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway -v
Conway-1.0.1-C
```

```
root@deblan-stretch-mips:/home# ./conway -v
Conway-1.0.1-MIPS32
```

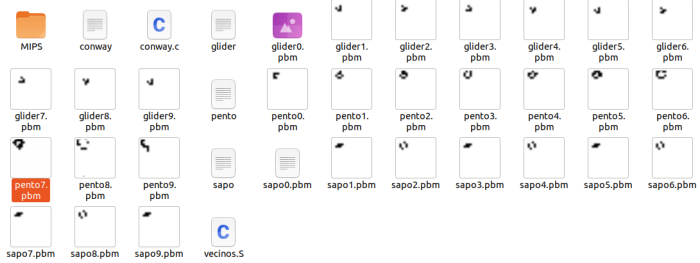
Y las ejecuciones de 20x20 pedidas en el enunciado:

```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 10 20 20
 sapo -o sapo
Generando matriz...
Matriz de 20x20 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 5, 3
Punto encontrado en 5, 4
Punto encontrado en 5, 5
Punto encontrado en 4, 4
Punto encontrado en 4, 5
Punto encontrado en 4, 6
Matriz cargada

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

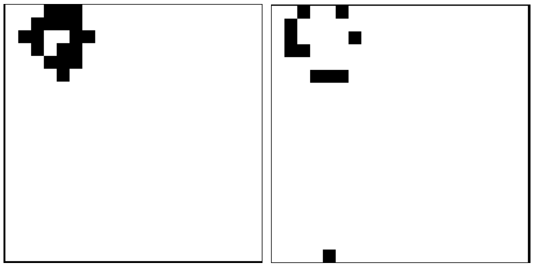
```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 10 20 20
 glider -o glider
Generando matriz...
Matriz de 20x20 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 5, 3
Punto encontrado en 5, 4
Punto encontrado en 5, 5
Punto encontrado en 3, 4
Punto encontrado en 4, 5
Matriz cargada

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

```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 10 20 20
 pento -o pento
Generando matriz...
Matriz de 20x20 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 3, 5
Punto encontrado en 3, 6
Punto encontrado en 4, 4
Punto encontrado en 4, 5
Punto encontrado en 5, 5
Matriz cargada

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

Luego de estas ejecuciones quedan guardadas las imágenes .pbm en la carpeta de la siguiente manera:



Se puede notar que la condición de borde se cumple en las siguientes fotos que son dos pasos consecutivos de la ejecución de pento:



A continuación muestro la ejecución de la versión MIPS de pento, mostrando que también se cumple la condición de borde con un screenshot del segundo paso mostrado anteriormente visualizado en consola (notar el 1 en el borde inferior):

```
root@debian-stretch-mips:/home# ./conway 10 20 20 pento
Generando matriz...
Matriz de 20x20 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 3, 5
Punto encontrado en 3, 6
Punto encontrado en 4, 4
Punto encontrado en 4, 5
Punto encontrado en 5, 5
Matriz cargada

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

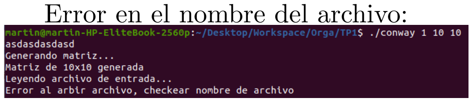
```
0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0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

Iterando...
```

4.1 Manejo de Errores

En esta sección muestro screenshots de como el programa maneja varios tipos de errores. Como un error en los argumentos:

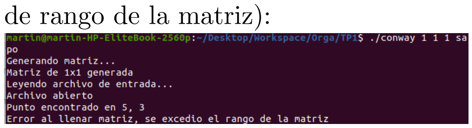
```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 1 10
Error en entrada de parametros, ejecutar ./conway -h para recibir ayuda
```

Error en el nombre del archivo:



```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 1 10 10
asdasdasdasd
Generando matriz...
Matriz de 10x10 generada
Leyendo archivo de entrada...
Error al arbir archivo, checkear nombre de archivo
```

Error en el tamaño de la matriz (el archivo tiene un punto que queda fuera de rango de la matriz):



```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 1 1 1 sa
po
Generando matriz...
Matriz de 1x1 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 5, 3
Error al llenar matriz, se excedio el rango de la matriz
```

Error en el formato del archivo, con el archivo utilizado:

```
martin@martin-HP-EliteBook-2560p:~/Desktop/Workspace/Orga/TP1$ ./conway 1 10 10
errorFormato
Generando matriz...
Matriz de 10x10 generada
Leyendo archivo de entrada...
Archivo abierto
Punto encontrado en 3, 3
Punto encontrado en 4, 1
Punto encontrado en 6, 6
Error al llenar matriz, formato de archivo incorrecto
```

```
3 3
4 1
6 6
4
```

5 Conclusiones

El trabajo practico tuvo más efecto que solo familiarizarse con el Assembly de MIPS. También tuvo una función de acercarse más a la programación de

bajo nivel y la fuerte dependencia de esta al hardware. Se puede notar que el lenguaje C tiene casi una traducción directa a Assembly, con la diferencia que el lenguaje C se puede escribir de la misma manera sin importar la arquitectura de la computadora en la que va a ser ejecutado.

El trabajo me llevo a descubrir que gcc en una computadora con una arquitectura no solo traduce el código C de manera distinta que en una computadora con otra arquitectura (cosa que es trivial si se sabe el el código Assembly es completamente distinto), sino que no **puede** compilar el código para otra arquitectura. Esto significa que ambas versiones de gcc tienen código de traducción que la otra no tiene. Esto significa que son programas diferentes con el mismo nombre y propósito.

Una característica del ABI de MIPS que me pareció extraña es que las funciones deben guardar sus propios argumentos en memoria cuando se prepara el stack frame. No encuentro el propósito ya que los argumentos solo son útiles para la función misma y otras funciones no los necesitan. La única razón que se me ocurre es a la hora de hacer un backtrace durante un debuggeo o un crash-dump se pueda ver los parámetros con los que se llamo la función problemática.

6 Codigo

Codigo de la implementacion en C:

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

int ejecutarConway(int argc, char** argv);
int llenarMatriz(unsigned char* matriz, FILE* archivo,
unsigned int alto, unsigned int ancho);
int avanzarTiempo(unsigned char* matriz, unsigned int alto,
unsigned int ancho, char* nombre);
unsigned int vecinos(unsigned char* matriz, unsigned int i,
unsigned int j, unsigned int alto, unsigned int ancho);
unsigned char realizarCambio(unsigned char estado,
unsigned char vecinos);

const size_t BUFLEN = 100;

int main(int argc, char** argv) {
bool argsOk = false;
int retorno = 0;
    if((argc == 2) && ((strcmp(argv[1], "-h") == 0) ||
     (strcmp(argv[1], "--help") == 0))) {
     printf("Uso:\n"
```

```
     "\tconway -h\n"
     "\tconway -V\n"
     "\tconway i M N inputfile [-o outputprefix]\n"
     "Opciones:\n"
     "\t-h, --help \tImprime este mensaje.\n"
     "\t-V, --version \tDa la versión del programa.\n"
     "\t-o \tPrefijo de los archivos de salida.\n"
     "Ejemplos:\n"
     "\tconway 10 20 20 glider -o estado\n"
     "\tRepresenta 10 iteraciones del Juego de la Vida
     en una matriz de "
     "20x20,\n\tcon un estado inicial tomado del archivo
     ‘‘glider’’.\n\t"
     "Los archivos de salida se llamarán estado_n.pbm.\n"
     "\tSi no se da un prefijo para los archivos de salida,
     \n\tel prefijo"
     " será el nombre del archivo de entrada.\n");
     argsOk = true;
    }

    if((argc == 2) && ((strcmp(argv[1], "-V") == 0) ||
     (strcmp(argv[1], "-v") == 0) ||
         (strcmp(argv[1], "--version") == 0))) {
     printf("Conway-1.0.1-C\n");
     argsOk = true;
    }

    if((argc == 5) || ((argc == 7) &&
    (strcmp(argv[5], "-o") == 0))) {
     argsOk = true;
     retorno = ejecutarConway(argc, argv);
    }

    if(!argsOk) {
     fprintf(stderr, "Error en entrada de parametros,
     ejecutar ./conway -h "
     "para recibir ayuda\n");
     retorno = -1;
    }
    printf("\n");
    return retorno;
}


int ejecutarConway(int argc, char** argv) {
printf("Generando matriz...\n");
```

```
unsigned int alto = atoi(argv[2]);
unsigned int ancho = atoi(argv[3]);

if ((alto == 0) || (ancho == 0)) {
fprintf(stderr, "Error al generar matriz,
checkear dimensiones\n");
return -2;
}
unsigned char* matriz = malloc(alto * ancho);
if (matriz == NULL) {
fprintf(stderr, "Error al generar matriz, no hay memoria\n");
return -3;
}
printf("Matriz de %ux%u generada\n", alto, ancho);

printf("Leyendo archivo de entrada...\n");
FILE* archivo;

archivo = fopen(argv[4], "r");
if (archivo == NULL) {
fprintf(stderr, "Error al arbir archivo,
checkear nombre de archivo\n");
free(matriz);
return -4;
}
printf("Archivo abierto\n");

int retorno = llenarMatriz(matriz, archivo, alto, ancho);

fclose(archivo);

char prefijo[BUFLEN];
if (argc == 7) {
snprintf(prefijo, BUFLEN, "%s", argv[6]);
}

unsigned int iteraciones = atoi(argv[1]);
if (retorno == 0) {
for (int i = 0; i < iteraciones; i++) {
char terminacion[50];
char nombreArchivo[BUFLEN];
snprintf(terminacion, 50, "%i.pbm", i);
snprintf(nombreArchivo, BUFLEN, "%s", prefijo);
strcat(nombreArchivo, terminacion);
retorno = avanzarTiempo(matriz, alto, ancho, nombreArchivo);
}
```

```
}
free(matriz);
return retorno;
}



int llenarMatriz(unsigned char* matriz, FILE* archivo,
unsigned int alto, unsigned int ancho) {
for (int i = 0; i < (alto * ancho); i++) {
matriz[i] = 0;
}
unsigned int fila, columna;
int resultadoLectura;
resultadoLectura = fscanf(archivo, "%u %u", &fila, &columna);
while (resultadoLectura == 2) {
printf("Punto encontrado en %u, %u\n", fila, columna);
if ((fila > alto) || (columna > ancho)) {
fprintf(stderr, "Error al llenar matriz,
se excedio el rango de la matriz\n");
return -5;
} else {
matriz[((fila - 1) * ancho) + columna - 1] = 1;
}
resultadoLectura = fscanf(archivo, "%u %u", &fila, &columna);
}
if (resultadoLectura != EOF) {
fprintf(stderr, "Error al llenar matriz,
formato de archivo incorrecto\n");
return -6;
} else {
printf("Matriz cargada\n\n");
for (int i = 0; i < alto; i++) {
for (int j = 0; j < ancho; j++) {
printf("%u ", matriz[(i * ancho) + j]);
}
printf("\n");
}
printf("\n");
}
return 0;
}


int avanzarTiempo(unsigned char* matriz, unsigned int alto,
unsigned int ancho, char* nombre) {
```

```
printf("Iterando...\n\n");
FILE* archivo = fopen(nombre, "w");
fprintf(archivo, "P1\n# %s\n%u %u\n\n", nombre, ancho, alto);

unsigned char* matrizVecinos = malloc(alto * ancho);
if (matrizVecinos == NULL) {
fprintf(stderr, "Error al generar matriz, no hay memoria\n");
return -3;
}

unsigned int i = 0;
unsigned int j = 0;

for (i = 0; i < alto; i++) {
for (j = 0; j < ancho; j++) {
unsigned char vecinosActuales =
vecinos(matriz, i, j, alto, ancho);
matrizVecinos[(i * ancho) + j] = vecinosActuales;
}
}

for (i = 0; i < alto; i++) {
for (j = 0; j < ancho; j++) {
matriz[(i * ancho) + j] = realizarCambio(matriz[(i * ancho) + j],
matrizVecinos[(i * ancho) + j]);
printf("%u ", matriz[(i * ancho) + j]);
fprintf(archivo, "%u ", matriz[(i * ancho) + j]);
}
printf("\n");
fprintf(archivo, "\n");
}
printf("\n");
free(matrizVecinos);
fclose(archivo);
return 0;
}


unsigned int vecinos(unsigned char* matriz, unsigned int i,
unsigned int j, unsigned int alto, unsigned int ancho) {
unsigned int vecinos = 0;
unsigned int filaAnterior = i - 1;
unsigned int columnaAnterior = j - 1;
unsigned int filaPosterior = i + 1;
unsigned int columnaPosterior = j + 1;
if (i == 0) filaAnterior = (alto - 1);
```

```
if (j == 0) columnaAnterior = (ancho - 1);
if (i == (alto - 1)) filaPosterior = 0;
if (j == (ancho - 1)) columnaPosterior = 0;

vecinos += matriz[(filaAnterior * ancho) + columnaAnterior];
vecinos += matriz[(filaAnterior * ancho) + j];
vecinos += matriz[(filaAnterior * ancho) + columnaPosterior];
vecinos += matriz[(i * ancho) + columnaAnterior];
vecinos += matriz[(i * ancho) + columnaPosterior];
vecinos += matriz[(filaPosterior * ancho) + columnaAnterior];
vecinos += matriz[(filaPosterior * ancho) + j];
vecinos += matriz[(filaPosterior * ancho) + columnaPosterior];
return vecinos;
}


unsigned char realizarCambio(unsigned char estado,
unsigned char vecinos) {
unsigned char nuevoEstado = estado;
if ((vecinos < 2) || (vecinos > 3)) {
nuevoEstado = 0;
}
if (vecinos == 3) {
nuevoEstado = 1;
}
return nuevoEstado;
}
```

Para la versión MIPS32, las únicas diferencias son la declaración de vecinos que se le agrega la keyword extern:

```
extern unsigned int vecinos(unsigned char* matriz,
unsigned int i, unsigned int j, unsigned int alto,
unsigned int ancho);
```

Tambien se cambia el mensaje de versión para MIPS32 de la siguiente manera:

```
printf("Conway-1.0.1-MIPS32\n");
```

Para la implementación de vecinos en Assembly, no pude hacer funcionar macros que acortaban código, así que resulto en una función larga.

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

#define SS 24
#define POS_A4 SS + 16
```

```
#define O_GP 20
#define O_FP 16

#define O_LTA_FA 0
#define O_LTA_FS 4
#define O_LTA_CA 8
#define O_LTA_CS 12

#define O_A0 (SS)
#define O_A1 (O_A0 + 4)
#define O_A2 (O_A1 + 4)
#define O_A3 (O_A2 + 4)
#define O_A4 (O_A3 + 4)
#define O_A5 (O_A4 + 4)


.text
.align 2

/*
unsigned int
vecinos(unsigned char* matriz, unsigned int i, unsigned int j,
unsigned int alto, unsigned int ancho)
*/

.globl vecinos
.ent vecinos

vecinos:

    subu sp, sp, SS

    /* Saved Register Area */
    sw gp, O_GP(sp)
    sw fp, O_FP(sp)
    move fp, sp

    /* Argument Building Area */
    sw a0, O_A0(fp)
    sw a1, O_A1(fp)
    sw a2, O_A2(fp)
    sw a3, O_A3(fp)
    lw t0, POS_A4(sp)
    sw t0, O_A4(fp)
    sw zero, O_A5(fp) # Padding del ABA
```

```
    /* Local and temporary area */
    /* filaAnterior = i - 1 */
    lw t1, 0_A1(fp) # $t1 = i
    sub t0, t1, 1 # $t0 = i - 1
    bne zero, t1, endifFA # si i != 0 saltear
    lw t0, 0_A3(fp) # $t0 = alto
    sub t0, t0, 1 # Resto 1 para que sea la ultima fila
endifFA:
    sw t0, 0_LTA_FA(fp)

/* filaSiguiente = i + 1 */
    addiu t0, t1, 1 # $t0 = i + 1
    lw t2, 0_A3(fp) # $t2 = alto
    sub t2, t2, 1 # Resto 1 para que esea la ultima columna
    bne t1, t2, endifFS # si i != alto saltear el siguiente paso
    addiu t0, zero, 0 # $t0 = 0
endifFS:
    sw t0, 0_LTA_FS(fp)

    /* columnaAnterior = j - 1 */
    lw t1, 0_A2(fp) # $t1 = j
    subu t0, t1, 1 # $t0 = j - 1
    bne zero, t1, endifCA # si j no es 0 saltear
    lw t0, 0_A4(fp) # $t0 = ancho
    sub t0, t0, 1 # Resto 1 para que sea la ultima columna
endifCA:
    sw t0, 0_LTA_CA(fp)

/* columnaSiguiente = j + 1 */
    addiu t0, t1, 1 # $t0 = i + 1
    lw t2, 0_A4(fp) # $t2 = ancho
    sub t2, t2, 1 # Resto 1 para que sea la ultima columna
    bne t1, t2, endifCS # si j != ancho saltear el siguiente paso
    addiu t0, zero, 0 # $t0 = 0
endifCS:
    sw t0, 0_LTA_CS(fp)

/* Calculo de vecinos */
    addiu t1, zero, 0 # Uso $t1 como vecinos, vecinos = 0
    lw t2, 0_A0(fp) # $t2 = matriz
    lw t0, 0_A4(fp) # $t0 = ancho
    lw t3, 0_LTA_FA(fp) # $t3 = filaAnterior
    mul t3, t3, t0 # $t3 = offset en matriz de fila anterior
    lw t4, 0_A1(fp) # $t4 = i
```

```
    mul t4, t4, t0 # $t4 = offset en matriz i
    lw t5, O_LTA_FS(fp) # $t5 = filaSiguiente
    mul t5, t5, t0 # $t5 = offset en matriz de fila siguiente
    lw t6, O_LTA_CA(fp) # $t6 = columnaAnterior
    lw t7, O_A2(fp) # $t7 = j
    lw t8, O_LTA_CS(fp) # $t8 = columnaSiguiente
    addu t9, t2, t3 # $t9 = posicion de la fila anterior
    addu t0, t9, t6 # $t0 = posicion de casillero
    lbu t0, (t0) # Cargo el casillero a $t0
    addu t1, t1, t0 # Casillero tiene 0 si esta vacio y 1 si no
    addu  t0, t9, t7 # Repito proceso en la misma columna...
    lbu t0, (t0)
    addu t1, t1, t0
    addu t0, t9, t8
    lbu t0, (t0)
    addu t1, t1, t0
    addu t9, t2, t4 # Ahora $t9 tiene la posicion  de la fila de i
    addu t0, t9, t6
    lbu t0, (t0)
    addu t1, t1, t0
    addu t0, t9, t8 # Salteo $t7. i, j es el casillero actual
    lbu t0, (t0)
    addu t1, t1, t0
    addu t9, t2, t5 # $t9 esta en posicion de fila siguiente
    addu t0, t9, t6
    lbu t0, (t0)
    addu t1, t1, t0
    addu  t0, t9, t7
    lbu t0, (t0)
    addu t1, t1, t0
    addu t0, t9, t8
    lbu t0, (t0)
    addu t1, t1, t0

    /* return (vecinos) */
    addiu v0, t1, 0

    /* Stack unwinding */
    lw fp, O_FP(sp)
    lw gp, O_GP(sp)
    addiu sp, sp, SS

    jr  ra
.end vecinos
```