FACULTAD DE INGENIERIA
Universidad de Buenos Aires

UBA FACULTAD DE INGENIERÍA

66.20 Organización de Computadoras

Trabajo Practico

2^{do} Cuatrimestre 2016

Grupo 1

Integrantes:

Federico Rodriguez	93336
fede.longhi@hotmail.com	
Ezequiel Dufau	91985
fede.longhi@hotmail.com	
Pablo Ascarza	89711
fede.longhi@hotmail.com	

Índice

1. Enunciado

2. Introducción

El presente trabajo tiene como objetivo familiarizarse con el conjunto de instrucciones *MIPS-32* y el concepto de ABI. Para el cumplimiento de este objetivo se desarrolló un programa que simula el *"Juego de la Vida" de Conway* según lo detallado en el enunciado.

La implementación se realizo en el lenguaje de programacion C. Además se desarrollo una porción en assembler *MIPS-32* que luego será detallada.

El programa fue desarrollado para correr sobre una plataforma *NetBSD / MIPS-32* mediante el emulador *GXEmul*.

3. Utilización

El programa fue implementado para que cumpliera con los requisitos pedidos por el tp. En las siguientes secciones se detallarán los diferentes aspectos para la ejecución del programa.

3.1. Compilación y Ejecución

1. Descargar el archivo Conway.zip desde el repositorio[1].

2. Descomprimir en el directorio que desee.

3. Desde el directorio donde se descomprimieron los archivos, ejecutar `make` para generar el código enteramente en C o `make mips` para generar el código con la función vecinos en Mips.

4. Ejecutar el programa con: `./conway i M N input [-o output]`

3.2. Documentación de Parámetros

- `i` es la cantidad de simulaciones que queremos realizar.
- `M` y `N` especifican las dimensiones de la matriz sobre la cual queremos simular.
- `input` es el nombre del archivo que contiene las coordenadas de las celdas vivas e identifica el estado inicial de la matriz.
- `-o` es un parámetro opcional que especifica que se utilizará el nombre `output` como prefijo de los nombres de los archivos pbm generados. En caso de no existir este parámetro tomara como prefijo `input`.

[1] https://github.com/fede29/orga_tp1

- `-V` o `--version` muestra la versión del programa.
- `-h` o `--help` muestra la ayuda.

3.3. Documentación de Errores

A continuación se detallan los errores y su significado:

- **Actions Count must be a positive integer:** El primer parámetro tiene que ser un entero positivo.
- **Number of rows must be a positive integer:** El número de filas tiene que ser un entero positivo.
- **Number of columns must be a positive integer:**El número de columnas tiene que ser un entero positivo.
- **Invalid parameter:** Para el caso de `-V`, `-h` y `-o`. El parámetro no coincide con estos valores (o sus equivalentes).
- **Wrong number of parameters:** Hay parámetros de mas o de menos (se pasó un número de parámetros distinto a 1 o 6).
- **Error while opening input file:** No se pudo abrir el archivo de entrada.
- **Error while opening output file: [nombre_de_archivo]:** No se pudo abrir el archivo de salida con el nombre *nombre_de_archivo*

3.4. Algunas Aclaraciones

- Las imágenes pbm generadas se guardan en la carpeta imágenes.
- Todos los errores se imprimen directamente a `stderr`.

3.5. Ejemplos de Uso

Para ver la documentación:

```
./conway -h
```

Para ver la informacion sobre la version:

```
./conway -V
```

Para generar un tablero de 100×50 a partir del archivo `glider` y realizar 20 iteraciones:

```
./conway 20 100 50 glider
```

Los archivos pbm generados por el comando anterior seran nombrados de la forma: "glider_N.pbm".

Para generar un tablero de 20×30 a partir del archivo `pento`, realizar 10 iteraciones y que los archivos pbm generados tengan como prefijo el nombre `jorge`:

```
./conway 10 20 30 pento -o jorge
```

Los archivos pbm generados por el comando anterior seran nombrados de la forma: "jorge_N.pbm".

4. Implementación

En esta sección se presentar porciones del programa. Para ver el código completo dirigirse al apéndice A.

La implementación se hizo completamente en C. Luego se programo en MIPS la función vecinos. A continuación se detallan las dos implementaciones.

4.1. Implementación en C

Para la implementación se diseño una función en C según la documentación del enunciado que corresponde a:

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

Tenemos que considerar un detalle sobre el tratamiento de la matriz para entender el algoritmo. En la figura siguiente se muestra la conversión de la matriz a array que utilizamos.

$$\begin{bmatrix} 0,0 & 0,1 & 0,2 \\ 1,0 & 1,1 & 1,2 \\ 2,0 & 2,1 & 2,2 \end{bmatrix} \rightarrow \begin{bmatrix} \underset{0}{0,0} & \underset{1}{0,1} & \underset{2}{0,2} & \underset{3}{1,0} & \underset{4}{1,1} & \underset{5}{1,2} & \underset{6}{2,0} & \underset{7}{2,1} & \underset{8}{2,2} \end{bmatrix}$$

A continuación se muestra el algoritmo implementado en C:

Código 1: Código de la funcion vecinos

```
1 unsigned int vecinos(unsigned char *a, unsigned int i, unsigned
      int j, unsigned int M, unsigned int N){
2   unsigned int vecinos = 0;
```

```
  int x,y;
  for (x = -1; x<2; x++){
    for (y = -1; y<2; y++){
      if (x+i>=0 && x+i<N && y+j>=0 && y+j<M){
        if ((!(x==0&&y==0)) && a[N*(x+i) + (y+j)] == '1'){
          vecinos++;
        }
      }
    }
  }
  return vecinos;
}
```

4.2. Implementación en MIPS

Para ver el código fuente dirigirse a la sección A.2.

4.2.1. Diagrama del Stack de Vecinos

5. Corridas de Prueba

5.1. Funcionamiento

Realizamos varias corridas de prueba con los archivos proporcionados (pento, glider y sapo) para verificar el correcto funcionamiento del programa. Mostramos a continuación algunas salidas de las corridas:

`./conway 10 10 10 glider -s`

```
fedelonghi@fedelonghi-dell $ ./conway 5 10 10 glider -s
starting action
Simulation number: 0

..........
..........
..........
....O.....
.....O....
...OOO....
..........
..........
..........
..........
Simulation number: 1

..........
..........
..........
..........
...O.O....
....OO....
....O.....
```

```
..........
..........
..........
Simulation number: 2

..........
..........
..........
..........
.....O....
...O.O....
....OO....
..........
..........
..........
Simulation number: 3

..........
..........
..........
..........
....O.....
.....OO...
....OO....
..........
..........
..........
Simulation number: 4

..........
..........
..........
..........
.....O....
......O...
....OOO...
..........
..........
..........
```

./test 10 10 10 glider -o archivo_salida

```
fedelonghi@fedelonghi-dell $ ./test 10 10 10 glider -o
    archivo_salida
starting action
```

Los archivos .pbm fueron generados en el directorio imagenes.

./test 3 10 10 sapo -s

```
fedelonghi@fedelonghi-dell $ ./test 3 10 10 sapo -s
starting action
Simulation number: 0

..........
```

```
..........
..........
..........
....OOO...
...OOO....
..........
..........
..........
..........
Simulation number: 1

..........
..........
..........
.....O....
...O..O...
...O..O...
....O.....
..........
..........
..........
Simulation number: 2

..........
..........
..........
..........
....OOO...
...OOO....
..........
..........
..........
..........
```

5.2. Resultados

Además medimos los tiempos de ejecucion para el programa compilado enteramente en c y el compilado con la función vecinos en mips. A continuación se muestran los resultados[2]:

[2]Todos los tiempos están medidos en segundos.

input	it	M	N	time 1	time 2	time 3
pento	10	10	10	0,098	0,117	0,102
	100	10	10	0,848	0,836	0,816
	10	20	20	0,832	0,781	0,750
glider	10	10	10	0,102	0,102	0,102
	100	10	10	0,836	0,836	0,785
	10	20	20	0,766	0,77	0,746
sapo	10	10	10	0,102	0,117	0,102
	100	10	10	0,848	0,836	0,816
	10	20	20	0,73	0,766	0,750

Tabla 1: Tiempo de ejecución del programa compilado enteramente C

input	it	M	N	time 1	time 2	time 3
pento	10	10	10	0,098	0,098	0,102
	100	10	10	0,797	0,801	0,77
	10	20	20	0,746	0,77	0,801
glider	10	10	10	0,102	0,098	0,082
	100	10	10	0,816	0,801	0,781
	10	20	20	0,77	0,73	0,75
sapo	10	10	10	0,098	0,102	0,098
	100	10	10	0,801	0,77	0,77
	10	20	20	0,785	0,766	0,77

Tabla 2: Tiempo de ejecución del programa compilado con la función vecinos en mips

Para tener una mejor noción sobre los tiempos a continuación se muestra el promedio. El promedio se calculo sin considerar el tipo de archivo de input, es decir considerando los todos los tiempos pertenecientes a las ejecuciones con los mismos parámetros i M y N.

Parámetros	Tiempo (C)	Tiempo (Mips)
10 10x10	0,1048	0,0975
100 10x10	0,8285	0,7896
10 20x20	0,7656	0,7653

Tabla 3: Promedio de los tiempos de ejecución

5.3. Análisis de Resultados

Observando las tablas podemos ver claramente que hay una tendencia a disminuir el tiempo de ejecucion utilizando el programa compilado en

mips. Esto se ve especialmente analizando la tabla 3. Donde claramente el promedio en todos los casos es menor en mips que en C.

6. Conclusiones

Además de familiarizarnos con las herramientas y el set de instrucciones de Mips pudimos observar como programando una función en mips podemos reducir el costo de tiempo en la ejecucion de un programa.

A la vista de los resultados, la ganancia es muy pequeña. Podemos deducir que se trata debido a que el programa en sí consta de otras partes que tienen mucha injerencia en el costo de tiempo del programa (como puede ser la creación de archivos pbm).

Vale aclarar también que la complejidad de programar una función en Mips es mucho mayor que la de programar una función en C. Mas allá del hecho de que estamos aprendiendo a programar en Mips, podemos notar la mayor cantidad de lineas que tiene el código de la función vecinos.s. Esta claro que para lograr un mismo resultado (con funcionalidades no tan pequeñas) en código assembler que en C es necesario programar mucho más.

Apéndice

A. Código Fuente

A.1. Programa Principal en C

Código 2: cornway.c

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

void writePBM(unsigned char** board, unsigned int dimx,
    unsigned int dimy, const char* fileName, unsigned int
    actionNumber)
{
int i, j;
int actionNumberStringLenght = 1;
if (actionNumber >= 10){
actionNumberStringLenght = 2;
}
char str[actionNumberStringLenght];
sprintf(str, "%d", actionNumber);
char newFileName[50];
newFileName[0]='\0';
strcat(newFileName,"imagenes/");
```

```
18 strcat(newFileName,fileName);
19 strcat(newFileName,"_");
20 strcat(newFileName,str);
21 strcat(newFileName,".pbm");
22
23 FILE *fp = fopen(newFileName, "wb");
24 if (fp==NULL){
25 fprintf(stderr, "Error while opening output file: %s\n",
       newFileName);
26 exit(1);
27 }
28
29 unsigned int amp = 1;
30 (void) fprintf(fp, "P4\n%d %d\n", dimy*amp, dimx*amp);
31 for (j = 0; j < dimy; ++j){
32 for (i = 0; i < dimx; ++i)
33 {
34 unsigned int x,y;
35 unsigned char writeValue = 0;
36 if ((board[i][j]) == '1'){
37 writeValue = 1;
38 }
39 for (x = i*amp; x < amp*(i+1); x++){
40 for (y = j*amp; y < amp*(j+1); y++){
41 fprintf(fp, "%c", writeValue);
42 }
43 }
44 }
45 }
46 (void) fclose(fp);
47 return;
48 }
49
50 unsigned char ** init_board(unsigned int rows, unsigned int
       cols){
51 unsigned char** board = (unsigned char**)malloc(cols*sizeof(
       char*));
52 unsigned int i,j;
53 for (i = 0; i<cols; i++){
54 board[i] = (unsigned char *) malloc(rows*sizeof(char*));
55 for (j = 0; j<rows; j++){
56 board[i][j]='0';
57 }
58 }
59 return board;
60 }
61
62 void load_board(unsigned char **board, unsigned int rows,
       unsigned int cols, FILE *file){
63 int x,y;
64 while (fscanf (file, "%i %i", &x, &y)!=EOF){
65 if (x<cols||y<rows){
66 board[x][y]='1';
67 }
```

```
68 }
69 }
70
71 int free_board (unsigned char **board, unsigned int rows,
       unsigned int cols){
72 unsigned int i;
73 for (i = 0; i<cols; i++){
74 free(board[i]);
75 }
76 free(board);
77 return 0;
78 }
79
80 void print_board(unsigned char **board, unsigned int rows,
       unsigned int cols){
81 unsigned int i,j;
82 for (i = 0; i<cols; i++){
83 for (j = 0; j<rows; j++){
84 if (board[i][j] == '0'){
85 printf(".");
86 }else{
87 printf("0");
88 }
89 }
90 printf("\n");
91 }
92 }
93
94 void print_board_file(unsigned char **board, unsigned int rows,
        unsigned int cols,int n,const char *filename){
95 char newFilename [50];
96 int numLength = 1;
97 if (n >= 10 && n <= 99){
98 numLength = 2;
99 }else if (n >= 100 && n <= 999){
100 numLength = 3;
101 }
102 char str[numLength];
103 sprintf(str, "%d", n);
104 newFilename[0] = '\0';
105 strcat(newFilename, "output/");
106 strcat(newFilename, filename);
107 strcat(newFilename, "_");
108 strcat(newFilename, str);
109 FILE *file = fopen(newFilename,"w+");
110 fprintf(file, "Simulacion N: %i\n",n);
111 unsigned int i,j;
112 for (i = 0; i<cols; i++){
113 for (j = 0; j<rows; j++){
114 if (board[i][j] == '0'){
115 fprintf(file,".");
116 }else{
117 fprintf(file,"0");
118 }
```

```
119 }
120 fprintf(file,"\n");
121 }
122 fclose(file);
123 }
124
125 unsigned char* board_to_array(unsigned char ** board, unsigned
        int rows, unsigned int cols){
126 unsigned char* array = (unsigned char*)malloc(sizeof(char*)*
        rows*cols);
127 int x,y;
128 for (x=0; x<cols; x++){
129 for (y=0; y<rows; y++){
130 unsigned int pos = cols*x+y;
131 array[pos]=board[x][y];
132 }
133 }
134 return array;
135 }
136
137 void copy_board(unsigned char **from, unsigned char **to,
        unsigned int rows, unsigned int cols){
138 unsigned int i,j;
139 for (i = 0; i<cols; i++){
140 for (j = 0; j<rows; j++){
141 to[i][j] = from[i][j];
142 }
143 }
144 }
145
146 void process_board(unsigned char **board, unsigned int rows,
        unsigned int cols, unsigned int actionCount,const char*
        fileName,bool screen){
147 printf ("starting action\n");
148 unsigned char **thisBoard = init_board(rows, cols);
149 copy_board(board,thisBoard,rows, cols);
150 unsigned int i,x,y;
151 for (i = 0; i < actionCount; i++){
152 unsigned char **nextBoard = init_board(rows, cols);
153 if (screen){
154 printf("Simulation number: %i\n\n", i);
155 print_board(thisBoard,rows,cols);
156 print_board_file(thisBoard,rows,cols,i,fileName);
157 }else{
158 writePBM(thisBoard, rows, cols, fileName, i);
159 }
160 for (x = 0; x < cols; x++){
161 for (y = 0; y < rows; y++){
162 unsigned char * array = board_to_array(thisBoard, rows, cols);
163 unsigned int neighbours = vecinos(array, x, y, rows, cols);
164 free(array);
165 if (thisBoard[x][y]=='1'){
166 if (neighbours == 3 || neighbours == 2){nextBoard[x][y] = '1';}
167 }else{
```

```
168 if (neighbours == 3){nextBoard[x][y] = '1';}
169 }
170 }
171 }
172 copy_board(nextBoard,thisBoard,rows, cols);
173 free_board(nextBoard, rows, cols);
174 }
175 free_board(thisBoard, rows, cols);
176 }
177
178 void print_help(){
179 printf("Uso:\n");
180 printf("  conway -h\n  conway -V\n  conway i M N inputfile [-o
        outputprefix]\n");
181 printf("Opciones:\n");
182 printf("  -h, --help    Imprime este mensaje\n");
183 printf("  -V, --version Da la version del programa\n");
184 printf("  -o            Prefijo de los archivos de salida\n");
185 printf("Ejemplos: \n");
186 printf("  conway 10 20 20 glider -o estado\n");
187 printf("  Representa 10 iteraciones del Juego de la Vida en una
         Matriz\n");
188 printf("  de 20x20, con un estado inicial tomado del archivo ''
        glider''.\n");
189 printf("  Los archivos de salida se llamarán estado_n.pbm,\n");
190 printf("  si no se da un prefijo para el archivo de salida, \n"
        );
191 printf("  el prefijo será el nombre del archivo de entrada.\n")
        ;
192 }
193
194 void print_version (){
195 printf("Conway -Game of Life- version: 1.0\n");
196 }
197
198 void validate_actionsCount(int actionsCount){
199 if (actionsCount <= 0){
200 fprintf(stderr, "Actions Count must be a positive integer!\n");
201 exit(1);
202 }
203 }
204
205 void validate_rows(int rows){
206 if (rows <= 0){
207 fprintf(stderr, "Number of rows must be a positive integer!\n")
        ;
208 exit(1);
209 }
210 }
211
212 void validate_cols(int cols){
213 if (cols <= 0){
214 fprintf(stderr, "Number of columns must be a positive integer!\
        n");
```

```
exit(1);
}
}

int main(int argc, char* argv[])
{
char const *fileName;
char const *outputFileName;
unsigned int actionsCount;
unsigned int rows;
unsigned int cols;
unsigned char **board;
bool screen = 0;

if (argc == 2){
char* arg = argv[1];
if (strcmp(arg,"-h") == 0 || strcmp(arg,"--help") == 0){
print_help();
return 0;
}
else if (strcmp(arg,"-v") == 0 || strcmp(arg,"--version") == 0)
    {
print_version();
return 0;
}
else{
fprintf(stderr,"Invalid parameter!\n");
exit(1);
}
}else if (argc != 5 && argc != 7 && argc != 6){ //ver
fprintf(stderr,"Wrong number of parameters!\n");
exit(1);
}else{
fileName = argv[4];
actionsCount = (int) atoi(argv[1]);
validate_actionsCount(actionsCount);
rows = (int) atoi(argv[2]);
validate_rows(rows);
cols = (int) atoi(argv[3]);
validate_cols(cols);
outputFileName = fileName;
if (argc == 7){
if (strcmp(argv[5],"-o") == 0){
outputFileName = argv[6];
}else{
fprintf(stderr,"Invalid parameter!\n");
exit(1);
}
}
if (argc == 6){
if (strcmp(argv[5],"-s") == 0){
screen = 1;
}else{
fprintf(stderr,"Invalid parameter!\n");
```

```
exit(1);
}
}
}

FILE* file = fopen(fileName, "r");
if (file==NULL){
fprintf(stderr, "Error while opening input file\n");
exit(1);
}

board = init_board(rows, cols);
load_board(board, rows, cols, file);
fclose(file);
process_board(board, rows, cols, actionsCount, outputFileName,
    screen);
free_board(board, rows, cols);

return 0;
}
```

A.2. Función Vecinos en Mips

Código 3: vecinos.s

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

##
    ############################################################################

# Empieza el codigo...
# los nombres de los registros estan mal es para ubicarlos
    facil y despues cambiarlos
# la cosa es asi las 8 comparaciones i +/- j +/- si pasa entra
    a la funcion donde se accede
# a i j de la matriz y verifica que sea igual a 1 devuelve 1
    sino 0, falta allocar bien los stack
# de la principal y de la funcion
##
    ############################################################################

.text
.align 2
.globl vecinos
.ent vecinos
vecinos:
subu  sp, sp, 76 #creo stack
sw  ra, 40(sp) #guardo el ra en stack
sw  $fp, 36(sp) #guardo el fp en stack
sw  gp, 32(sp) #guardo el gp en stack
sw  s0, 44(sp) #guardo el s0 en stack
```

```
21 sw  s1, 48(sp) #guardo el s1 en stack
22 sw  s2, 52(sp) #guardo el s2 en stack
23 sw  s3, 56(sp) #guardo el s3 en stack
24 sw  s4, 60(sp) #guardo el s4 en stack
25 sw  s5, 64(sp) #guardo el s5 en stack
26 sw  s6, 68(sp) #guardo el s6 en stack
27 sw  s7, 72(sp) #guardo el s7 en stack
28 sw  a0, 0(sp) #guardo el 1er parametro N en el stack  en stack
29 sw  a1, 4(sp) #guardo el 2do parametro N en el stack  en stack
30 sw  a2, 8(sp) #guardo el 3er parametro N en el stack  en stack
31 sw  a3, 12(sp) #guardo el 4to parametro N en el stack  en stack
32 lw  t0, 92(sp) #cargo el 5to parametro N en el stack  en stack
33 sw  t0, 16(sp) #guardo el 5to parametro N en el stack  en stack
34 move  $fp, sp #muevo el fp al inicio del stack
35 move  t1, a0 #cargo el 1er parametro a* en t1
36 move  t2, a1 #cargo el 2do parametro i en t2
37 move  t3, a2 #cargo el 3er parametro j en t3
38 move  t4, a3 #cargo el 4tp parametro M en t4
39 move  s1, t2 #i
40 move  s2, t3 #j
41 addi  s3, s1, 1 #imasuno
42 addi  s4, s2, 1 #jmasuno
43 addi  s5, s1, -1  #imenosuno
44 addi  s6, s2, -1 #jmenosuno
45 move  s0, t4 #m
46 move  s7, t0 #n
47 move  t8, t1 #matriz
48 li  t6, 0 #en t6 se va a ir acumulando el valor
49 blt s3, s0, imas #compara si i + 1 < m
50 comp2:
51 blt s4, s7, jmas #compara si j + 1 < n
52 comp3:
53 bge s5, zero, imenos #compara si i - 1 >= 0
54 comp4:
55 bge s6, zero, jmenos #compara si j - 1 >= 0
56 comp5:
57 bge s3, s0, comp6 #compara si i + 1 >= m
58 blt s4, s7, imasjmas #compara si j + 1 < n
59 comp6:
60 bge s3, s0, comp7 #compara si i + 1 >= m
61 bge s6, zero, imasjmenos #compara si j - 1 >= 0
62 comp7:
63 blt s5, zero, comp8 #compara si i + 1 < 0
64 blt s4, s7, imenosjmas #compara si j + 1 < n
65 comp8:
66 blt s5, zero, fin #compara si i + 1 < 0
67 bge s6, zero, imenosjmenos #compara si j - 1 >= 0
68 #falta desapilar stack
69 fin:
70 #move v0, s7
71 lw  ra, 40(sp) #empiezo a liberar stack, cargo el ra del stack
72 lw  $fp, 36(sp) #cargo el fp del stack
73 sw  s7, 16(sp) #el param 5
74 lw  gp, 32(sp) #cargo el gp del stack
```

```
lw  s0, 44(sp) #cargo el s0 del stack
lw  s1, 48(sp) #cargo el s1 del stack
lw  s2, 52(sp) #cargo el s2 del stack
lw  s3, 56(sp) #cargo el s3 del stack
lw  s4, 60(sp) #cargo el s4 del stack
lw  s5, 64(sp) #cargo el s5 del stack
lw  s6, 68(sp) #cargo el s6 del stack
lw  s7, 72(sp) #cargo el s7 del stack
lw  a0, 0(sp) #cargo el param 1
lw  a1, 4(sp) #cargo el param 2
lw  a2, 8(sp) #cargo el param 3
lw  a3, 12(sp) #cargo el param 4

addiu sp, sp, 76
move  v0, t6
jr  ra

imas: #se llama a a[i+1][j] para ver si esta vivo o no
move  a0, s3
move  a1, s2
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp2
jmas: #se llama a a[i][j+1] para ver si esta vivo o no
move  a0, s1
move  a1, s4
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp3
imenos: #se llama a a[i-1][j] para ver si esta vivo o no
move  a0, s5
move  a1, s2
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp4
jmenos: #se llama a a[i][j-1] para ver si esta vivo o no
move  a0, s1
move  a1, s6
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp5
imasjmas: #se llama a a[i+1][j+1] para ver si esta vivo o no
move  a0, s3
move  a1, s4
move  a2, s7
move  a3, t8
```

```
jal accessijpos
add t6, v0, t6
b comp6
imasjmenos: #se llama a a[i+1][j-1] para ver si esta vivo o no
move  a0, s3
move  a1, s6
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp7
imenosjmas: #se llama a a[i-1][j+1] para ver si esta vivo o no
move  a0, s5
move  a1, s4
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b comp8
imenosjmenos: #se llama a a[i-1][j-1] para ver si esta vivo o
    no
move  a0, s5
move  a1, s6
move  a2, s7
move  a3, t8
jal accessijpos
add t6, v0, t6
b fin

.globl accessijpos
accessijpos:
subu  sp, sp, 32 #creo stack
sw  ra, 24(sp) #guardo el ra en stack
sw  $fp, 20(sp) #guardo el fp en stack
sw  gp, 16(sp) #guardo el gp en stack
move  t2, a0 # pos i
move  t4, a1 # pos j
move  t3, a2 # n de la matriz
move  t0, a3 # pos 0,0 de la matriz
mul t5, t3, t2 # $t5 <-- width * i
add t5, t5, t4 # $t5 <-- width * i + j
add t5, t0, t5 # $t5 <-- base address + (2^2 * (width * i + j))
lbu t1, 0(t5) #carga el '1' o '0' de la matriz
addiu t1, t1,-48 # carga '1' en t7 para comparar
move  v0, t1
lw  ra, 24(sp) #empiezo a liberar stack, cargo el ra del stack
lw  $fp, 20(sp) #cargo el fp del stack
lw  gp, 16(sp) #cargo el gp del stack
addiu sp, sp, 32
jr  ra
.end  vecinos
```

A.3. Función Vecinos en C

Código 4: vecinos.c

```
unsigned int vecinos(unsigned char *a, unsigned int i, unsigned
     int j, unsigned int M, unsigned int N){
  unsigned int vecinos = 0;
  int x,y;
  for (x = -1; x<2; x++){
    for (y = -1; y<2; y++){
      if (x+i>=0 && x+i<N && y+j>=0 && y+j<M){
        if ((!(x==0&&y==0)) && a[N*(x+i) + (y+j)] == '1'){
          vecinos++;
        }
      }
    }
  }
  return vecinos;
}
```

B. Dirección del Repositorio

Todos los archivos correspondientes al trabajo practico se encuentran en este repositorio:

https://github.com/fede29/orga_tp1