Trabajo Práctico 1 — Conjunto de instrucciones MIPS

[66.20] Organización de Computadoras
Curso 2
Primer cuatrimestre de 2021

Alumnos	Padrón	Email
ARRACHEA, Tomás	104393	tarrachea@fi.uba.ar

Índice

1. Diseño y detalles de implementación **2**

2. Stack de la función *proximo* **2**

3. Ejemplos de ejecución **2**

4. Conclusiones **5**

5. Apéndice **6**
5.1. autcel.c 6
5.2. proximo.c 9
5.3. proximo.S 10

1. Diseño y detalles de implementación

El programa se implementó en C a partir de una matriz dinámica de chars de tamaño n*n. Se carga la primera fila con el contenido inicial del autómata, proporcionado por el archivo inicial. Luego, se accede a cada posición de la matriz y se asigna el valor correspondiente del autómata, calculado por la función *proximo*, que accede a la posición a partir de un puntero al heap y encapsula la lógica de cálculo del autómata.

En el módulo implementado en MIPS, se decidió no guardar los valores de las variables de los registros temporales en el Stack. De esta manera, se logra un rendimiento más rápido, a costa de perder la posibilidad de hacer un backtracing. Esto impide que al momento de debuggear el programa, se pueda acceder al valor de esas variables. Sin embargo, considero que no es necesario para este programa.

2. Stack de la función *proximo*

Diagrama del manejo del stack de la función *proximo*, necesario para el manejo de la ABI:

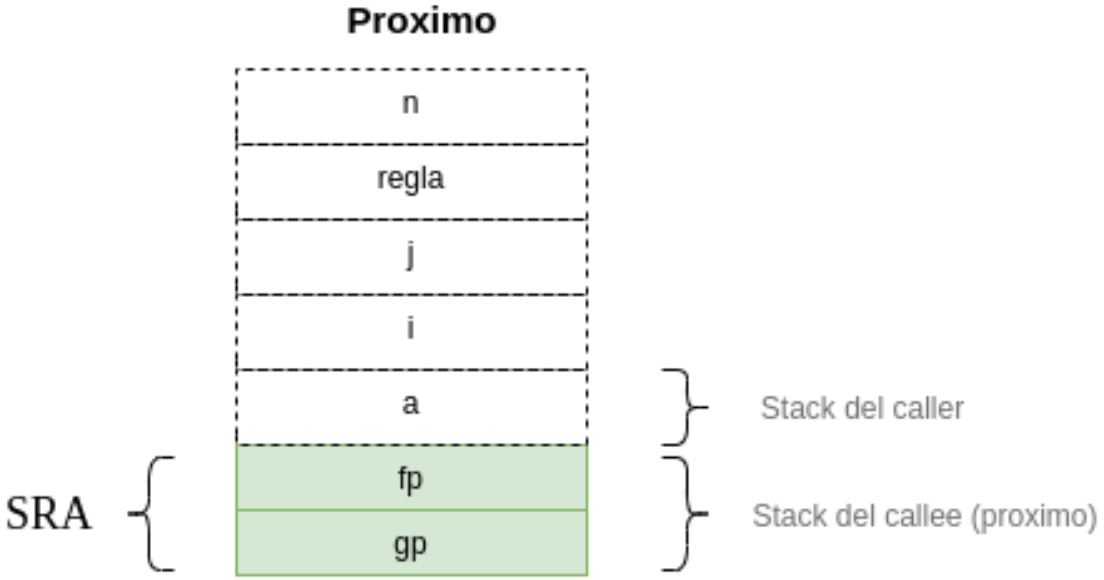

Figura 1: Stack de *proximo*.

3. Ejemplos de ejecución

Se hicieron corridas de prueba para una matriz de lado 80, de las reglas 30, 110 y 126, con una celda ocupada en el centro como estado inicial. Estos fueron los resultados:

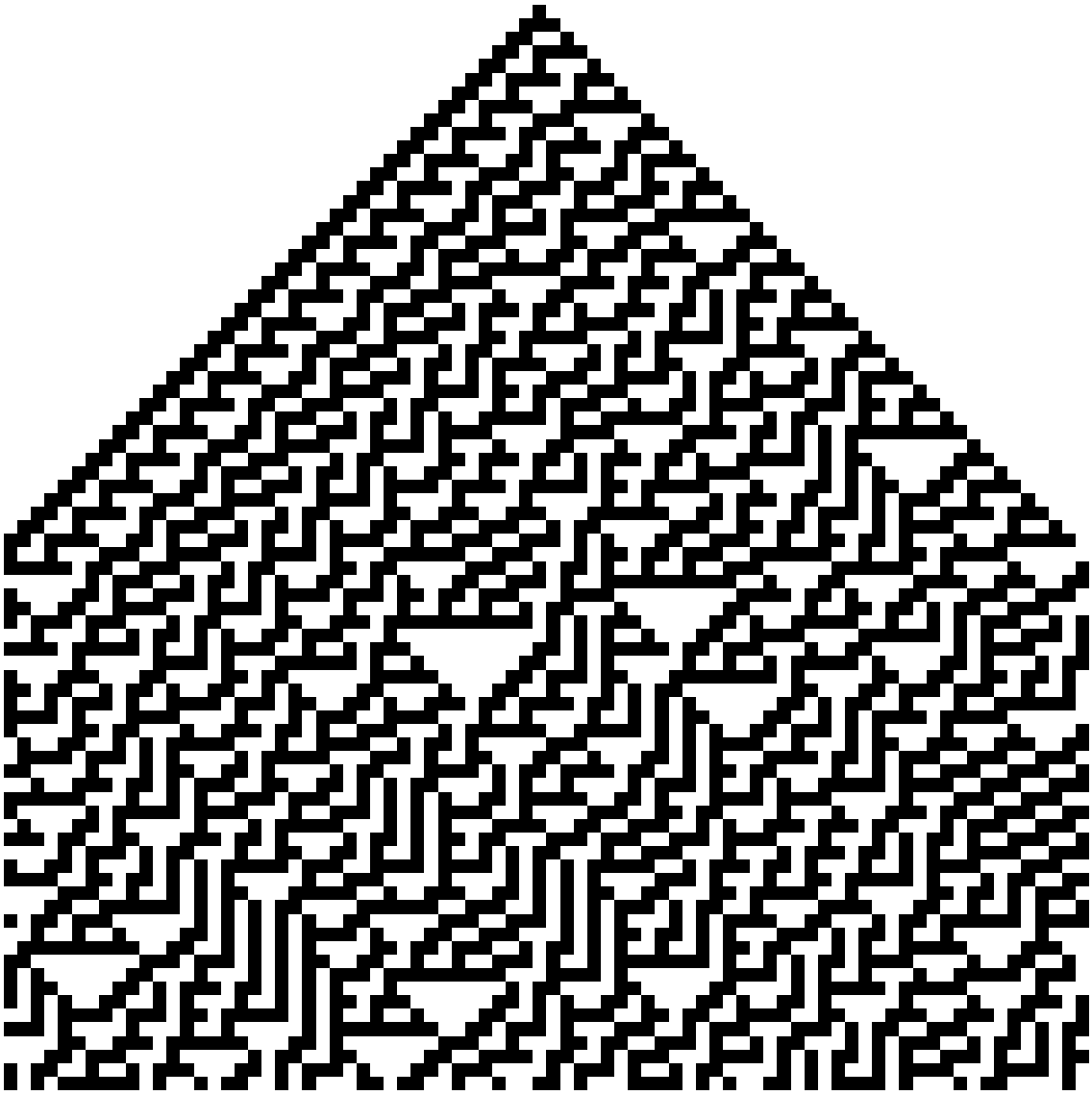

Figura 2: Corrida con regla 30.

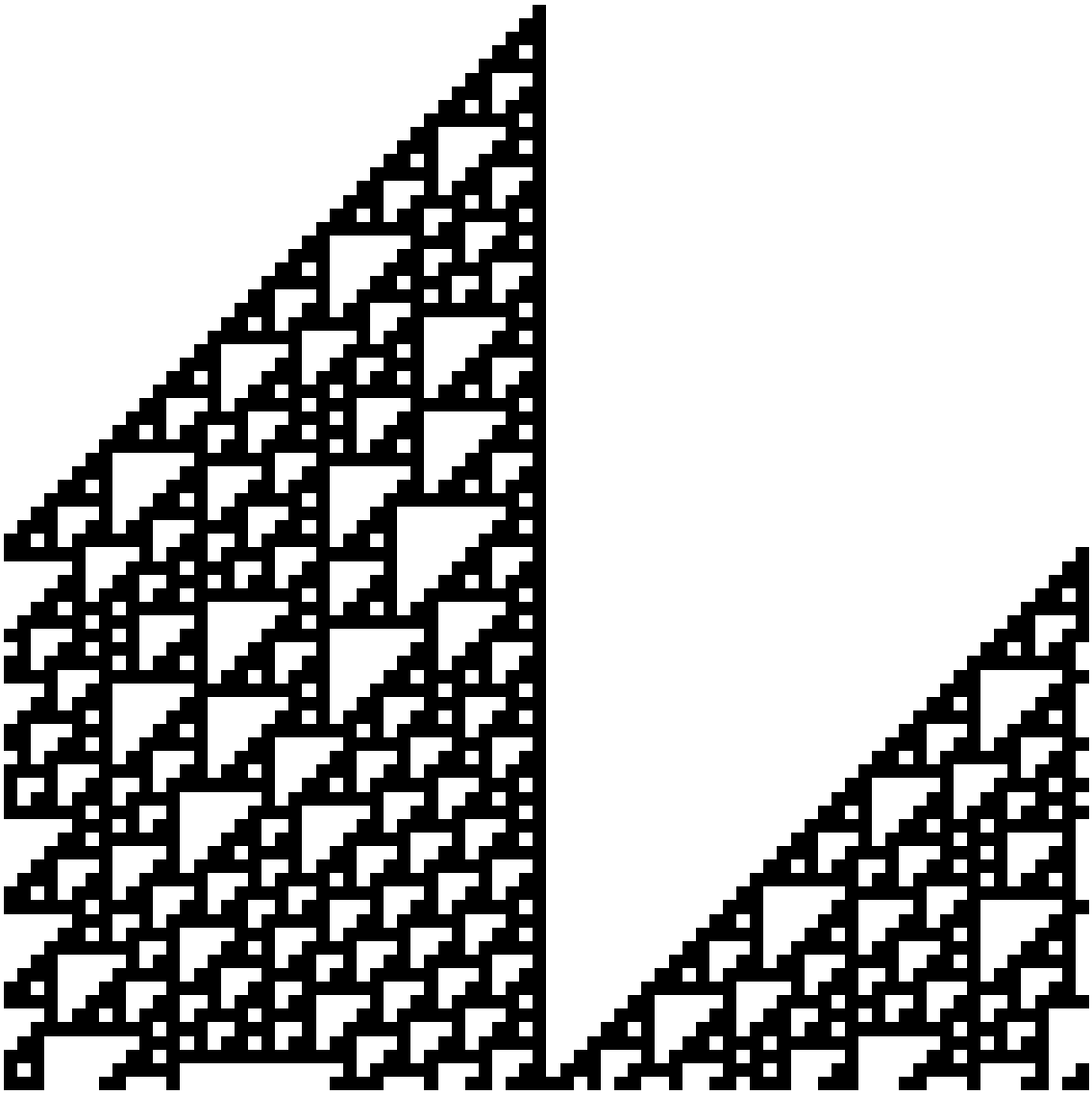

Figura 3: Corrida con regla 110.

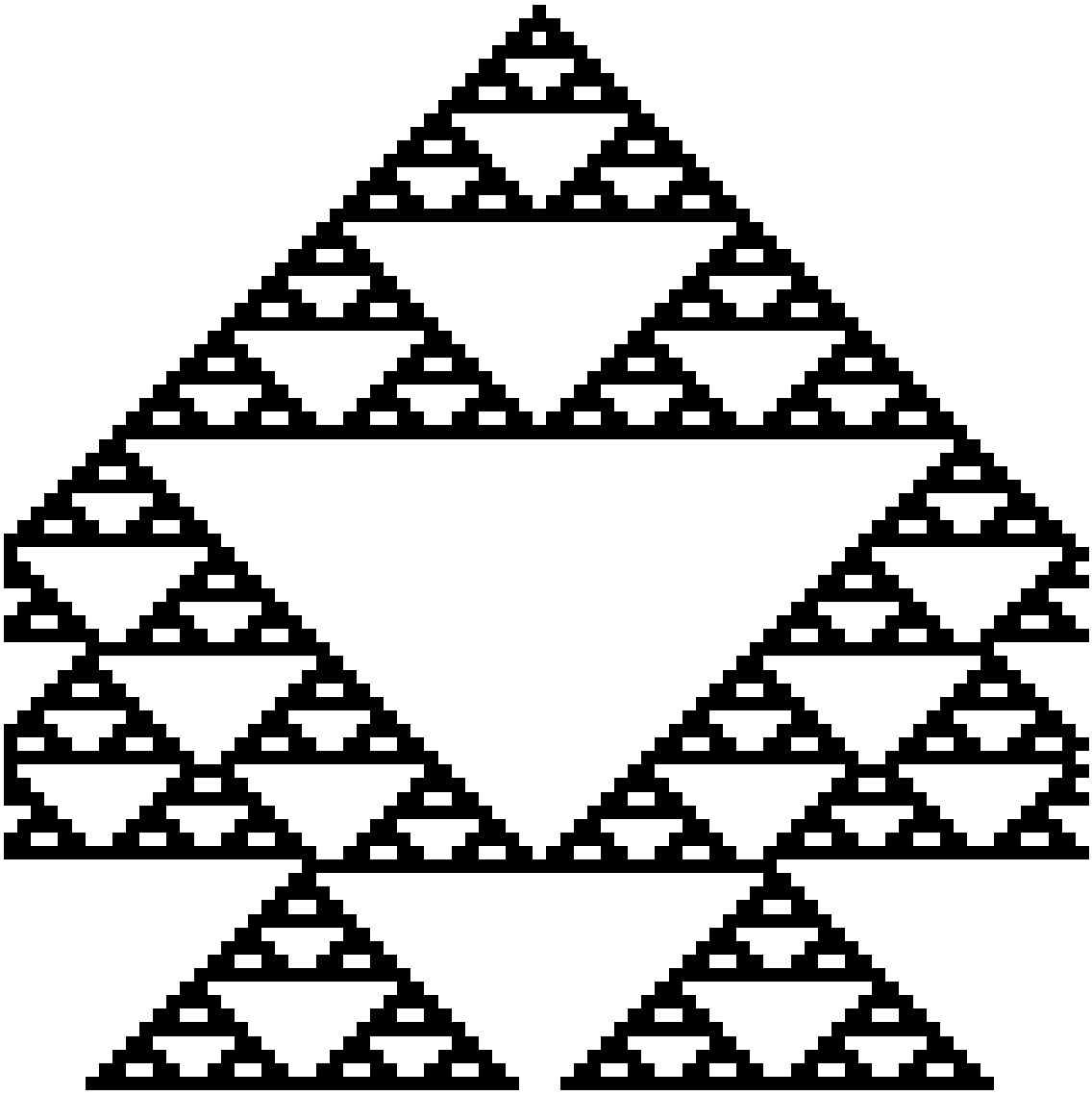

Figura 4: Corrida con regla 126.

4. Conclusiones

Tras haber realizado todo el trabajo práctico, pude afianzar mis conocimientos sobre la arquitectura MIPS y programación en Assembly. Pude implementar la función en lenguaje C y en MIPS, viendo las diferencias entre ambos métodos.

Al implementar la función en MIPS, se trabajó más a fondo para entender como funciona la interfaz de comunicación para la invocación de funciones, el manejo de los parámetros en memoria y en registros.

Por último, el manejo de imagenes y el formato pmb fue interesante. Se pudo ver de forma interactiva el funcionamiento de los autómatas celulares.

5. Apéndice

Código fuente.

5.1. autcel.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>

const char* VERSION_ACTUAL = "1.2";
const int MULTIPLICADOR_PIXELS = 4;

extern unsigned char proximo(unsigned char* a, unsigned int i,
    unsigned int j, unsigned char regla, unsigned int n);
void imprimir_ayuda();
void imprimir_version();
int cargar_inicio(char* nombre_inicial, unsigned char** estados, int
    n);
void guardar_fila(FILE* salida, unsigned char* estados, int n);
FILE* inicializar_pbm(char* nombre, int n);
void generar_automata(FILE* salida, char* nombre_inicial, char regla,
    int n);

int main(int argc, char* argv[]) {
    FILE* salida = NULL;
    int c;
    char* nombre_salida = "evolucion.pbm";

    while ((c = getopt(argc, argv, "hVpo:")) != -1)
        switch (c) {
            case 'h':
                imprimir_ayuda();
                return 0;
            case 'V':
                imprimir_version();
                return 0;
            case 'p':
                salida = stdout;
                break;
            case 'o':
                nombre_salida = optarg;
                break;
            case '?':
                if (optopt == 'o')
                    fprintf (stderr, "La opci n -%c requiere un
                        argumento.\n", optopt);
                else if (isprint (optopt))
                    fprintf (stderr, "Opci n desconocida `-%c'.\n",
                        optopt);
                else
                    fprintf (stderr,"Caracter desconocido
                        `\\x%x'.\n", optopt);
```

```
                return 1;
            default:
                abort ();
    }

    int index = optind;

    if (argc < 4) {
        fprintf(stderr, "Error: los argumentos son inv lidos.");
        return 1;
    }

    int regla = atoi(argv[index++]);
    int n = atoi(argv[index++]);
    char* nombre_inicial = argv[index++];

    if (salida == NULL) {
        salida = inicializar_pbm(nombre_salida, n);
    }

    if (regla > 255 || regla <= 0 || n <= 0) {
        fprintf(stderr, "Error: los argumentos son inv lidos.");
        fclose(salida);
        return 1;
    }

    generar_automata(salida, nombre_inicial, (char)regla, n);

    fclose(salida);
    return 0;
}

FILE* inicializar_pbm(char* nombre, int n) {
    FILE* file = fopen(nombre, "w");
    if (!file){
        fprintf(stderr, "Error al crear el archivo de salida.");
        return 0;
    }

    fprintf(file, "P1\n# feep.pbm\n%i %i\n", n*MULTIPLICADOR_PIXELS,
       n*MULTIPLICADOR_PIXELS);

    return file;
}

void generar_automata(FILE* salida, char* nombre_inicial, char regla,
   int n) {
    unsigned char* estados[n];
    for (int i = 0; i < n; i++)
        estados[i] = malloc(n*sizeof(char));

    if (cargar_inicio(nombre_inicial, estados, n) != 0) {
        return;
    }
```

```
    guardar_fila(salida, estados[0], n);
    for (unsigned int i = 0; i < n-1; i++) {
        for (unsigned int j = 0; j < n; j++) {
            estados[i+1][j] = proximo((unsigned char*)estados, i, j,
                regla, n);
        }
        guardar_fila(salida, estados[i+1], n);
    }

    for (int j = 0; j < n; j++)
        free(estados[j]);
}

void imprimir_ayuda(){
    printf(
    "   Uso:\n"
    "        autcel -h\n"
    "        autcel -V\n"
    "        autcel R N inputfile -s\n"
    "        autcel R N inputfile [-o outputprefix]\n"
    "\n Opciones:\n"
    "        -h  Imprime este mensaje.\n"
    "        -V  Da la versi n del programa.\n"
    "        -s  Imprime por consola la salida del programa, en lugar
       de guardarlo en el archivo de salida.\n"
    "        -o  Prefijo de los archivos de salida.\n"
    "\n Ejemplos:\n"
    "   -> autcel 30 80 inicial -o evolucion\n"
    "        Calcula la evoluci n del aut mata 'Regla 30', en un
       mundo unidimensional de 80 celdas, por 80 iteraciones.\n"
    "        El archivo de salida se llamar  evolucion.pbm.\n"
    "        Si no se da un prefijo para los archivos de salida, el
       prefijo ser  el nombre del archivo de entrada\n");
}

void imprimir_version(){
    printf("Versi n actual: %s\n", VERSION_ACTUAL);
}

int cargar_inicio(char* nombre_inicial, unsigned char** estados, int
   n) {
    FILE* archivo_inicial = fopen(nombre_inicial, "r");
    if (!archivo_inicial){
        fprintf(stderr, "Error al abrir el archivo inicial.");
        return 2;
    }

    char numero;
    for (int j = 0; j < n; j++){
        numero = getc(archivo_inicial) - '0';
        if (numero == EOF){
            fclose(archivo_inicial);
```

```
            fprintf(stderr, "Error: el archivo inicial tiene celdas de menos.");
            return -1;

        } else if (numero != 0 && numero != 1){
            fclose(archivo_inicial);
            fprintf(stderr, "Error: el archivo inicial no cumple con el formato.");
            return -1;
        }
        estados[0][j] = numero;
    }

    if (fscanf(archivo_inicial, "%s", &numero) != EOF){
        fprintf(stderr, "Error: el archivo inicial tiene celdas de sobra.");
        return 1;
    }

    fclose(archivo_inicial);

    return 0;
}

void guardar_fila(FILE* salida, unsigned char* estados, int n){
    for (int k = 0; k < MULTIPLICADOR_PIXELS; k++)  {
        for (int j = 0; j < n; j++) {
            for (int i = 0; i < MULTIPLICADOR_PIXELS; i++) {
                fprintf(salida, "%i ", estados[j]);
            }
        }
        fprintf(salida, "\n");
    }
}
```

autcel.c

5.2. proximo.c

```
extern unsigned char proximo(unsigned char** a, unsigned int i, unsigned int j, unsigned char regla, unsigned int n){
    char previo, central, siguiente;
    int posicion;

    if (j == 0) {
        previo = a[i][n-1];
        siguiente = a[i][j+1];
    } else if (j == n-1) {
        previo = a[i][j-1];
        siguiente = a[i][0];
    } else {
        previo = a[i][j-1];
        siguiente = a[i][j+1];
    }
    central = a[i][j];
```

```
    posicion = previo << 2 | central << 1 | siguiente << 0;

    return (1 & (regla >> posicion));
}
```

proximo.c

5.3. proximo.S

```
1       .text
2       .align 2
3       .globl proximo
4       .ent proximo
5
6 proximo:
7       subu $sp, $sp, 8 # 2 words (SRA) + 0 words (LTA) + 0 words
            (ABA) = 8 bytes
8
9       # SRA
10      sw $gp, 4($sp)
11      sw $fp, 8($sp)
12      move $fp, $sp
13
14      #guardo los argumentos en el stack del callee
15            #$a0 = a
16            #$a1 = i
17            #$a2 = j
18            #$a3 = regla
19      sw $a0, 8($fp)
20      sw $a1, 12($fp)
21      sw $a2, 16($fp)
22      sw $a3, 20($fp)
23
24      #$t0 = posicion
25      #$t1 = digito1
26      #$t2 = digito2
27      #$t3 = digito3
28      #t$4 = n
29      lw $t4, 24($fp)         #carga n desde el stack del callee
30
31      sll $a1, $a1, 2         # i * 4, son punteros de 4 bytes
32      add $a0, $a0, $a1       # $a0 = a + i
33      lw $a0, 0($a0)
34      add $a0, $a0, $a2 # $a0 = a[i] + j
35
36      #digito2 = estados[i][j]
37      lb $t2, 0($a0)
38
39 ##if (j == 0)
40      beq $zero, $a2, lim_inferior
41 ##if (j == n-1)
42      addi $t4, $t4, -1
43      beq $t4, $a2, lim_superior
44 ##else
```

```
45        #digito1 = a[i][j-1]
46        #digito3 = a[i][j+1]
47        lb $t1, -1($a0)
48        lb $t3, 1($a0)
49        j exit
50
51 lim_inferior:
52        #digito1 = a[i][n-1]
53        #digito3 = a[i][j+1]
54        lb $t3, 1($a0)
55        addu $a0, $a0, $t4
56        lb $t1, -1($a0)
57        j exit
58
59 lim_superior:
60        #digito1 = a[i][j-1]
61        #digito3 = a[i][0]
62        lb $t1, -1($a0)
63        subu $a0, $a0, $a2
64        lb $t3, ($a0)
65        j exit
66
67 exit:
68
69        #digito1 = digito1 << 2
70        #digito2 = digito2 << 1
71        sll $t1, $t1, 2
72        sll $t2, $t2, 1
73
74        #posicion = digito1 << 2 | digito2 << 1 | digito3 << 0
75        or $t0, $t1, $t2
76        or $t0, $t0, $t3
77
78        #regla >> posicion
79        srlv $a3, $a3, $t0
80
81        #$v0 = regla & 1.
82        andi $v0, $a3, 1
83
84        #liberar Stack
85      lw $fp, 0($sp)
86      lw $gp, 4($sp)
87      addiu $sp, $sp, 8
88
89        jr $ra
90 .end proximo
```

proximo.S