Trabajo práctico #1: conjunto de instrucciones MIPS

Santiago Fernandez, *Padrón Nro. 94.489*
fernandezsantid@gmail.com

Francisco Landino, *Padrón Nro. 94.475*
landinofrancisco@gmail.com

Matias Duarte, *Padrón Nro. 92.186*
duarte.mati@gmail.com

2do. Cuatrimestre de 2014
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Este trabajo práctico trata de una versión en lenguaje C, de un programa que lee desde un archivo de texto o desde STDIN, un texto que contiene tags, posiblemente anidados, y verifica que la estructura sea correcta. Además, se hizo un version en Assembly, de una función, para familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI.

Índice

1. Introducción

Como objetivo de este trabajo practico se trabajo tanto usando el conjunto de instrucciones de MIPS32 como en el lenguaje C, para codificar un programa que recibe desde STDIN o desde un archivo de texto, un texto en el cual se encuentran tags, de la forma *<Tag>... </Tag>*, posiblemente anidados. El programa se encarga de verificar que estén correctamente relacionados, lo que significa que el texto que abre un tag sea igual al texto que lo cierra y que estén correctamente anidados.

También se trabajo usando el ABI presentado por la cátedra, para guardar los argumentos en el stack entre las llamadas de función a función.

2. Desarrollo

Para desarrollar este trabajo primero se realizo un programa en lenguaje C, el cual contempla las siguientes condiciones.

Los *tags* tienen que cumplir con la siguiente estructura:

- Todos los *tags* abiertos se cierran,
- Todos los *tags* que se cierran fueron abiertos.
- Todos los *tags* abiertos dentro de otro *tag* se cierran antes que ?este.
- Si se detecta una violación de alguna de las reglas anteriores, se especifica el error (*tag* sin cerrar, *tag* sin abrir, *tag* mal anidado) y el numero de línea (empezando por 0) por **stderr**. En caso de que la estructura del archivo sea valida, el programa debe devolver 0, caso contrario devolver 1, ademas de la salida por **stderr**.

Un *tag* esta compuesto por una secuencia de apertura, de la forma "<**nombre-tag**>", un contenido que puede ser texto u otros *tags*, y una secuencia de cierre de la forma "</**nombre-tag**>".

Se considera que no pueden haber caracteres del tipo *'<'* o *'>'* sin que implique una apertura o cierre de un *tag*.

Luego a partir de esta implementacion en C, se desarrollaron las funciones *validate* y *analizarTag* en Assembly MIPS32. Para dicha implementación se uso el ABI que enseño la cátedra durante las clases. Al principio de cada función se crea un stack, donde se guardan los registros del Return Address, Frame Pointer y Global Pointer en las posiciones mas grandes del stack. Luego se guardan los atributos de la función y finalmente los argumentos.

2.1. Recursos y Portabilidad

Uno de los objetivos del trabajo práctico es poder probar la portabilidad del programa en diferentes entornos. En el enunciado se pedía que el programa se pudiera ejecutar en NetBSD[4]/pmax (usando el simulador GXemul[5]) y en la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386. En particular, se lo probó en Ubuntu 14.04. En GXemul

se corrió una máquina de arquitectura MIPS cuyo sistema operativo era una versión reciente de NetBSD/pmax. La transferencia de archivos entre la máquina host y la guest se hizo mediante *SSH*. Se procedió de la siguiente manera:

Para trabajar con el GXemul se procedió primero creando una nueva interfaz de red (debe crearse cada vez que se inicia el *host* y con permisos de administrador):

hostOS$ sudo ifconfig lo:0 172.20.0.1

Luego se ejecutó el GXemul en modo X:

hostOS$ ./xgxemul -e 3max -d netbsd-pmax.img -x

Una vez ya ingresado con el usuario y la contraseña en la máquina simulada, se creó un túnel reverso para saltear las limitaciones propias del GXemul:

guestOS$ ssh -R 2222:127.0.0.1:22 usuario@172.20.0.1

A partir de ese momento y dejando lo anterior en segundo plano, ya se puedo trabajar mediante SSH de manera más cómoda:

hostOS$ ssh -p 2222 root@127.0.0.1

2.2. Implementación

El programa en lenguaje C, cuenta con dos archivos: *main.c* y *validate.c*.

En el archivo *main.c* se encuentran tres funciones:

void cargarArchivoAMemoria(FILE archivoEntrada, char* text)*

Se encarga de pasar el archivo de entrada a memoria. Por *archivoEntrada* se pasa el puntero al archivo y por *text* se obtiene el puntero a la cadena de texto en memoria.

void printManual()

Imprime el manual sobre como usar el programa.

void checkFile(FILE file)*

Verifica si el archivo del cual se quiere levantar es correcto.

Luego en el main el programa se encarga de procesar todas las opciones, que se le introducen por argumento, y en el caso de que se pueda levantar el texto correctamente, llama a la función *int validate(char *text, char **errmsg)* la cual se encarga de validar el archivo. Si validate devuelve 0, el programa imprime un 0 y termina su ejecución, pero si validate devuelve un 1, imprime un 1, junto

con el mensaje de error en errmsg.

En el archivo *validate.c* se encuentran dos funciones:

int analizarTag(char text, char* tagEncontrado, int pos, int *contadorLineas)*

Analiza que el tag que le pasan por argumento sea igual al próximo tag que se cierra. Si encuentra que se abre otro tag antes de encontrar que se cierra un tag, levanta el tag anidado y se llama recursivamente para que lo verifique.

text es el puntero al texto que se esta analizando.

tagEncontrado es el puntero a la cadena de texto que contiene el tag que se levanto anteriormente y que hay que comparar con el siguiente tag que se cierre.

pos es la posición a partir de la cual hay que ir analizando el texto.

contadorLineas es el puntero al contador que lleva el numero de linea.

La función devuelve:

-1 si es un error donde el ultimo tag abierto no fue cerrado.

-2 si es un error donde el tag cerrado no corresponde con el tag abierto.

Sino devuelve un numero mayor a cero, el cual indica la posición del texto donde quedo después de analizar que se cerro correctamente el tag.

*int validate(char *text, char **errmsg)*

Va recorriendo el texto hasta que encuentre el fin o hasta que encuentre un tag. Cuando encuentra un tag, lo guarda en el atributo *tagALevantar* y luego llama a la función *analizarTag* para que verifique si se cierra correctamente. Si analizar tag devuelve en un numero mayor a cero, continua analizando el texto desde esa posición, sino verifica que tipo de error es y devuelve por errmsg, el mensaje que se va a imprimir.

text es el puntero al texto a verificar.

errmsg es el puntero a la cadena de texto donde se tiene que guardar el mensaje de error en el caso de que haya alguno error en la estructura de los tags en el texto.

La función devuelve un 0 si llego al fin del texto y no se encontró ningún error, con la estructura de los tags, o devuelve un 1 si se encontró algún error.

2.3. Compilación

Para compilar el trabajo práctico, realizamos un Makefile para hacer mas sencilla esta tarea.

Para compilar para C: *make c*

Para compilar para MIPS: *make mips*

Finalmente para limpiar los archivos: *make clean*

2.4. Corrida de Pruebas

Hicimos corridas con diferentes textos, para probar las diferentes combinaciones posibles y además, probamos el funcionamiento para archivos y para stdin, tanto en su version en c como en mips. Luego, los resultados fueron los siguientes.

```
Corrida de Pruebas en C:

-Prueba Archivos

Para entender estas pruebas, si el programa no devuelve nada es que
termino su ejecucion bien.
En caso contrario devolvia en numero de linea y el mensaje de error.

root@:~/orgaTP1# nl pruebaCorrecta.txt
     1  <hola> esto es una prueba </hola>
     2  <tag1> prueba </tag1>
root@:~/orgaTP1# ./validate pruebaCorrecta.txt
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaCorrectaAnidados.txt
     1  <hola> esto es una prueba </hola>
     2  <tag1> prueba </tag1>
     3  <hola> <tag2>  esto es una prueba con 2 tags </tag2> </hola>
     4  <tag5> <tag3> <otroTag> prueba con 3 </otroTag> </tag3> </tag5>
root@:~/orgaTP1# ./validate pruebaCorrectaAnidados.txt
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaIncorrecta.txt
     1  <hola> esto es una prueba </hola>
     2  <tag1> prueba </chau>
root@:~/orgaTP1# ./validate pruebaIncorrecta.txt

Linea: 2. Tag mal anidado, el ultimo tag cerrado, no corresponde con el
ultimo tag abierto.

root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaIncorrectaAnidados.txt
     1  <tag1> prueba </tag1>
     2  <hola> <tag2>  esto es una prueba con 2 tags </tag2> </hola>
     3  <tag5> <tag3> <otroTag> prueba con 3 </tag3> </otroTag> </tag5>
root@:~/orgaTP1# ./validate pruebaIncorrectaAnidados.txt
```

```
Linea: 3. Tag mal anidado, el ultimo tag cerrado, no corresponde con el
ultimo tag abierto.

root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaTagNoCerrado.txt
     1  <tag1> prueba </tag1>
     2  <hola>
root@:~/orgaTP1# ./validate pruebaTagNoCerrado.txt

Linea: 2. El tag abierto, no fue cerrado.

root@:~/orgaTP1#
```

-Prueba de STDIN

```
root@:~/orgaTP1# echo "<hola>" | ./validate -i -

Linea: 1. El tag abierto, no fue cerrado.

root@:~/orgaTP1#
```

```
root@:~/orgaTP1# echo "<hola> </hola>" | ./validate -i -
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# ./validate -i -
<hola> <tag2> esto es una prueba de anidados </tag2></hola>
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# ./validate -i -
<hola> <tag2> esto es una prueba erronea de anidados </hola> <tag3>

Linea: 1. Tag mal anidado, el ultimo tag cerrado, no corresponde con
el ultimo tag abierto.

root@:~/orgaTP1#
```

Corrida de Pruebas en MIPS

-Prueba Archivos

```
Para entender estas pruebas, si el programa no devuelve nada es que
termino su ejecucion bien.
En casa contrario devolvia en numero de linea y el mensaje de error.

root@:~/orgaTP1# nl pruebaCorrectaMips.txt
     1  <tag1> prueba </tag1>
```

```
     2  <hola> esto es una prueba </hola>
root@:~/orgaTP1# ./validate pruebaCorrecta.txt
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaCorrectaAnidadosMips.txt
     1  <hola> esto es una prueba </hola>
     2  <hola> <tag2>  esto es una prueba con 2 tags </tag2> </hola>
     3  <tag1> prueba </tag1>
     4  <tag5> <tag3> <otroTag> prueba con 3 </otroTag> </tag3> </tag5>
root@:~/orgaTP1# ./validate pruebaCorrectaAnidados.txt
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaIncorrectaMips.txt
     1  <hola> esto es una prueba </hola>
     2  <tag1> prueba </chau>
root@:~/orgaTP1# ./validate pruebaIncorrecta.txt

Linea: 2. Tag mal anidado, el ultimo tag cerrado, no corresponde con el
ultimo tag abierto.

root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaIncorrectaAnidadosMips.txt
     1  <tag1> prueba </tag1>
     2  <tag5> <tag3> <otroTag> prueba con 3 </tag3> </otroTag> </tag5>
     3  <hola> <tag2>  esto es una prueba con 2 tags </tag2> </hola>
root@:~/orgaTP1# ./validate pruebaIncorrectaAnidados.txt

Linea: 2. Tag mal anidado, el ultimo tag cerrado, no corresponde con el
ultimo tag abierto.

root@:~/orgaTP1#
```

```
root@:~/orgaTP1# nl pruebaTagNoCerradoMips.txt
     1  <hola>
     2  <tag1> prueba </tag1>
root@:~/orgaTP1# ./validate pruebaTagNoCerrado.txt

Linea: 1. El tag abierto, no fue cerrado.

root@:~/orgaTP1#
```

-Prueba de STDIN

```
root@:~/orgaTP1# echo "<hola> </hola>" | ./validate -i -
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# ./validate -i -
```

```
<hola> <tag2> esto es una prueba de anidados </tag2></hola>
root@:~/orgaTP1#
```

Hasta este momento, la corrida de pruebas venia bien pero nos encontramos con este error.

```
root@:~/orgaTP1# echo "<hola>" | ./validate -i -

Linea: 1. El tag abierto, no fue cerrado.

Segmentation fault (core dumped)
root@:~/orgaTP1#
```

```
root@:~/orgaTP1# ./validate -i -
<hola> <tag2> esto es una prueba erronea de anidados </hola> <tag3>

Linea: 0. Tag mal anidado, el ultimo tag cerrado, no correspondecon el
ultimo tag abierto.

Segmentation fault (core dumped)
root@:~/orgaTP1#
```

Despues de buscar un rato y no encontrar solucion en el codigo, decidimos usar gdb.

```
root@:~/orgaTP1# gdb validate validate.core
GNU gdb 5.3nb1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "mipsel--netbsd"...
Core was generated by `validate'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /usr/libexec/ld.elf_so...done.
Loaded symbols for /usr/libexec/ld.elf_so
Reading symbols from /usr/lib/libc.so.12...done.
Loaded symbols for /usr/lib/libc.so.12
#0  0x7dfb4508 in _rtld_exit () from /usr/libexec/ld.elf_so
(gdb)
```

Y nos encontramos como que el error estaba en el final de la ejecucion del programa, o sea se imprime el mensaje de error pero en el medio entre eso y el return 1 de la funcion ocurria el segmentation fault. Luego estuvimos buscando un largo rato a ver si encontrabamos el problema pero al no

encontrar solucion lo dejamos asi.

Como se puede ver tanto el programa con archivos como por stdin funcionan, vuelven normalmente y o retornar el mensaje por stderr en caso de error, tanto en sus versiones C como Mips.

Una unica optimizacion posible seria poder encontrar el error que esta causando que solo por stdin cuando termina de imprimir el error hace que falle el programa.

2.5. Código

2.5.1. Código en C

Listing : validate.h

```
#ifndef VALIDATE_H_
#define VALIDATE_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <getopt.h>
#include "mymalloc.h"

extern int validate(char* text, char** errmsg);
extern int analizarTag(char* text, char* tagEncontrado, int pos, int *
    contadorLineas);


#endif /* VALIDATE_H_ */
```

Listing : validate.c

```
#include "validate.h"

int analizarTag(char* text, char* tagEncontrado, int pos, int *
    contadorLineas)
{
        while(text[pos] != '\0')
        {
                if(text[pos] == '\n')
                {
                        (*contadorLineas)++;
                }
                if(text[pos] == '<')
                {
                        pos++;
                        // Si entra en cerrar un tag
```

```
if(text[pos] == '/')
{
        pos++;
        int j = 0;
        while((text[pos] != '>') && (tagEncontrado
            [j] == text[pos]))
        {
                j++;    pos++;
        }
        if((tagEncontrado[j] == '\0') && (text[pos]
            == '>'))
        {
                return pos;
        } else {
                return -2;
        }
}  else {
        int contadorTag = pos;
        while(text[contadorTag] != '>')
        {
                contadorTag++;
        }
        contadorTag = contadorTag - pos;
        contadorTag = contadorTag + 1; //Para el
            '\0' final
        char* tagALevantar = (char*)malloc(sizeof
            (char)*contadorTag);
        int k = 0;
        while(text[pos] != '>')
        {
                tagALevantar[k] = text[pos];
                k++; pos++;
        }
        tagALevantar[k] = '\0';
        pos++;
        pos = analizarTag(text, tagALevantar, pos,
            contadorLineas);
        free(tagALevantar);
        switch(pos)
        {
        case -1:
                return -1;
                break;
        case -2:
                return -2;
                break;

        default:
                break;
```

```
                                }
                        }
                }
                pos++;
        }
        return -1;
}

int validate(char *text, char **errmsg){

        char* buffer;
        int i = 0;
        int contadorLineas = 0;
        while(text[i] != '\0')
        {
                if(text[i] == '\n')
                {
                        contadorLineas++;
                }

                if(text[i] == '<')
                {
                        i++;
                        int contadorTag = i;
                        while(text[contadorTag] != '>')
                        {
                                contadorTag++;
                        }
                        contadorTag = contadorTag - i;
                        contadorTag = contadorTag + 1; //Para el '\0' final
                        char* tagALevantar = (char*)malloc(sizeof(char)*
                            contadorTag);
                        int j = 0;
                        while(text[i] != '>')
                        {
                                tagALevantar[j] = text[i];
                                j++; i++;
                        }
                        tagALevantar[j] = '\0';
                        i = analizarTag(text, tagALevantar, i, &
                            contadorLineas);
                        free(tagALevantar);
                        switch(i)
                        {
                        case -1:
                                contadorLineas = contadorLineas + 1;
                                sprintf(buffer, "Linea: %d. El tag abierto,
                                    no fue cerrado.\n", contadorLineas);
                                *errmsg = buffer;
                                return 1;
```

```
                                break;
                        case -2:
                                contadorLineas = contadorLineas + 1;
                                sprintf(buffer, "Linea: %d. Tag mal
                                    anidado, el ultimo tag cerrado, no
                                    corresponde con el ultimo tag abierto.\
                                    n", contadorLineas);
                                *errmsg = buffer;
                                return 1;
                                break;

                        default:
                                break;

                        }
                }
                i++;
        }
        return 0;
}
```

Listing : main.c

```
#include "validate.h"

void cargarArchivoAMemoria(FILE* archivoEntrada, char* text)
{
        int actual = 0;
        while(!feof(archivoEntrada)){
                (text)[actual] = fgetc(archivoEntrada);
                actual++;
        }
        (text)[actual - 1] = '\0';
}

//Funcion que imprime el manual del TP1
void printManual(){
        printf("Usage:\n validate -h\n");
        printf(" validate -V\n");
        printf(" validate [options] file \n");
        printf("Options:\n");
        printf(" -h, --help     Prints usage information.\n");
        printf(" -V, --version  Prints version information.\n");
        printf(" -i, --input    Path to input file (-i - for stdin)\n");
        printf("Examples:\n");
        printf(" validate -i -\n");
        printf(" validate myfile.tagged\n");
        printf(" validate -i myfile.tagged\n");
}
```

```
//Funcion para comprobar que los archivos se abrieron bien.
void checkFile(FILE* file){
    if ( file == NULL){
        //Si hay error se escribe por stderr.
        fprintf(stderr, "Error, nombre de archivo inexistente, el programa
            se cerrara.\n");
        exit(1); // Se cierra el programa y se devuelve 1 por error.
    }
}

//Funcion principal del TP1
int main(int argc, char* argv[]){

        int next_option;

        const char* const short_options = "i:hV";

        const struct option long_options[] = {

                { "input",      1, NULL, 'i' },
                { "help",       0, NULL, 'h' },
                { "version",    0, NULL, 'V' },
                { NULL,         0, NULL, 0 } /* Necesario al final del
                    array. */
        };

        //Aca por defecto se establecen algunos parametros.
        //Luego depende las opciones elegidas se van cambiando.
        FILE* archivoEntrada = stdin;
        const char* nombreArchivo;
        int ejecutar = 0;
        int flag;
        char* errmsg;
        int tamanio = 2048;
        char text[tamanio];

        //Procesamiento de los parametros de entrada.
        do {
                next_option = getopt_long(argc, argv, short_options,
                    long_options, NULL);

                switch (next_option){

                case 'i':    /* -i, --input */
                /* Indica el archivo de entrada a utilizar (-i - for stdin)
                    */
                if (strcmp(optarg, "-") != 0){
                        nombreArchivo = optarg;
                        archivoEntrada = fopen(nombreArchivo, "r");
                        checkFile(archivoEntrada);
```

```
                cargarArchivoAMemoria(archivoEntrada, text);
                fclose(archivoEntrada);
        }else {
                cargarArchivoAMemoria(archivoEntrada, text);
        }
        ejecutar = 1;
        break;

        case 'h':    /* -h, --help */
        /* Imprime el menu de ayuda */
        printManual();
        return 0;
        break;

        case 'V':    /* -V, --version */
        /*  Prints version information.  */
        printf("␣Version␣1.0␣del␣TP1\n");
        exit(0);
        break;

        case -1:    /* Se terminaron las opciones */
        break;

        default:    /* Opcion incorrecta */
        fprintf(stderr, "Error,␣opcion␣incorrecta,␣el␣programa␣se␣
            cerrara.\n");
        printManual(); //Se imprime el manual para que se vean las
            opciones correctas.
        exit(1);          // Se cierra el programa y se devuelve 1 por
            error.
        }
} while (next_option != -1);

if (argc == (optind + 1)){
        nombreArchivo = argv[optind];
        archivoEntrada = fopen(nombreArchivo, "r");
        checkFile(archivoEntrada);
        cargarArchivoAMemoria(archivoEntrada, text);
        fclose(archivoEntrada);
        ejecutar = 1;
}

//Arranca la ejecucion del programa.
if (ejecutar){

        //Se llama a la funcion validate.
        // *text es un puntero al texto contenido en el archivo.
        // **errmsg es un puntero a un array de caracteres, a llenar
            por la funcion validate en caso de error.
```

```
                //Se utiliza la variable flag para ver si hubo error o no en
                    la validacion.
                //La funcion debe retornar 0 en caso de que la validacion
                    sea correcta, o 1 en caso de que no.
                flag = validate(text, &errmsg);
                if (flag>0){
                        fprintf(stderr, "\n%s\n", errmsg);
                        return 1;
                }

        } else {
                fprintf(stderr, "Error, accion invalida, el programa se
                    cerrara.\n");
                printManual(); //Se imprime el manual para que se vean las
                    opciones correctas.
                return 1;       // Se cierra el programa y se devuelve 1 por
                    error.
        }
    return 0;
}
```

2.5.2. Código en Mips

Listing : validate.S

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

#Argumentos de la funcion
#define VALIDATE_ARG1     60
#define VALIDATE_ARG0     56

#Stack Size
#define VALIDATE_SS               56

#SRA
#define VALIDATE_RA       48
#define VALIDATE_FP               44
#define VALIDATE_GP               40

#LTA
#define VALIDATE_RT               36
#define VALIDATE_BF               32
#define VALIDATE_TAGL     28
#define VALIDATE_J                24
#define VALIDATE_CL               20
#define VALIDATE_I                16

#ABA
```

```
#define ABA_3                   12
#define ABA_2                   8
#define ABA_1                   4
#define ABA_0                   0

#Constantes
#define FIN_TEXTO               0
#define SALTO_DE_LINEA  10
#define MENOR                   60
#define MAYOR                   62

        .data
        .align  2
ER_TAG_MAL_CERRADO:
                .ascii  "Linea: %d. El tag abierto, no fue cerrado.\n\0"

        .align  2
ER_TAG_MAL_ANIDADO:
                .ascii  "Linea: %d. Tag mal anidado, el ultimo tag cerrado,
                    no corresponde"
                .ascii  "con el ultimo tag abierto.\n\0"

        .text
        .align  2
        .globl  validate
        .ent            validate

validate:
        .frame  $fp, VALIDATE_SS, ra
        .set            noreorder
        .cpload t9
        .set            reorder
        #Creo el stack frame
        subu            sp,     sp,     VALIDATE_SS
        .cprestore VALIDATE_GP
        sw              ra,     VALIDATE_RA(sp)
        sw              $fp,VALIDATE_FP(sp)
        sw              gp, VALIDATE_GP(sp)
        move    $fp,sp
        sw              a0, VALIDATE_ARG0($fp)
        sw              a1, VALIDATE_ARG1($fp)



        sw      zero, VALIDATE_I($fp)       # i = 0
        sw              zero, VALIDATE_CL($fp)      # contadorLineas =
            0

whileFinLinea:
        lw              t1, VALIDATE_ARG0($fp)      # t1 = *text
        lw              t0, VALIDATE_I($fp)                 # t0 = i
```

```
        addu    t6, t0, t1                      # Cargo direc te
            text[i] en t6
        lbu             t6, 0(t6)                       # t6
            = text[i]
        li              t2, FIN_TEXTO
        bne             t6, t2, ifFinLinea              # Si text[i]
            != \0 voy a ifFinLinea
        b               return_0

ifFinLinea:
        lw              t1, VALIDATE_ARG0($fp)  # t1 = *text
        lw              t0, VALIDATE_I($fp)             # t0 = i
        addu    t6, t0, t1                      # Cargo direc te
            text[i] en t6
        lbu             t6, 0(t6)                       # t6
            = text[i]
        li              t2, SALTO_DE_LINEA              # t2 = \n
        bne             t6,t2, ifMenor                  # (if text[i
            ] != \n)
        lw              t3,VALIDATE_CL($fp)     # t3 -> contador
        addu    t3, t3, 1                       #
            contadorLinea++
        sw              t3,VALIDATE_CL($fp)     #Almaceno el
            contadorLinea modificado en el stack

ifMenor:
        lw              t1, VALIDATE_ARG0($fp)  # t1 = *text
        lw              t0, VALIDATE_I($fp)             # t0 = i
        addu    t6, t0, t1                      # Cargo direc te
            text[i] en t6
        lbu             t6, 0(t6)                       # t6
            = text[i]
        li              t2, MENOR                       # t2
            = <
        bne             t6, t2, incrementarPos  #Si no son iguales
            incremento la posicion
        lw              t0, VALIDATE_I($fp)     #Cargo el valor de i
             almacenado en el stack
        addu    t0, t0,1                                # i++
        sw              t0, VALIDATE_I($fp)



        lw              t0, VALIDATE_I($fp)             #Cargo la i
        sw              t0, VALIDATE_TAGL($fp)  #contadorTag = i
contadorTag:
        lw              t1, VALIDATE_ARG0($fp)  # t1 = *text
        lw              t0, VALIDATE_TAGL($fp)  # t0 = contadorTag
        addu    t6, t0, t1                      # Cargo direc te
            text[i] en t6
```

```
        lb              t6, 0(t6)                                   # t6
            = text[i]
        li              t2, MAYOR
        bne             t6, t2, aumentarConTag      #Distino de fin de
            texto
        lw              t2, VALIDATE_TAGL($fp)      #t2 = contadorTag
        lw              t0, VALIDATE_I($fp)             #t0 = i
        subu    t7, t2, t0                              #
            contadorTag = contadorTag - i
        addu    t7, t7, 1                               #
            contadorTag = contadorTag + 1 para el '\0'
        sw              t7, VALIDATE_TAGL($fp)
        lw              a0, VALIDATE_TAGL($fp)      #Cargo el
            argunmento de la funcion
        la              t9, mymalloc                    #Cargo en
            t9 la direccion de la funcion
        jal             t9
                    #llamo a la funcion malloc
        sw              v0, VALIDATE_TAGL($fp)      #guardo la posicion
            de memoria que reserve


        sw              zero,VALIDATE_J($fp)        # j = 0
whileNotFinTag:
        lw              t1, VALIDATE_ARG0($fp)      # t1 = *text
        lw              t0, VALIDATE_I($fp)             #Cargo el
            valor de i almacenado en el stack
        addu    t6, t1, t0                              #Muevo el
            texto a la nueva i
        lbu             t6, 0(t6)                               #
            text[i]
        li              t2, MAYOR                               #
            Cargo en t2 >
        bne             t6, t2, tagALevantar        # text[i] != '>'
        lw              t4, VALIDATE_TAGL($fp)      #Almaceno memoria
             para tagALevantar
        lw              t5, VALIDATE_J($fp)             #Almacento
             en t5 j
        addu    t8, t4, t5                              #Cargo en
            a0 la direccion de tagALevantar[j]
        li              t2, FIN_TEXTO                   #Cargo en
            t2 \0
        sb              t2, 0(t8)                               #
            tagALevantar[j] = '\0'
        b               switchValidate

        aumentarConTag:
        lw              t0,VALIDATE_TAGL($fp)
        addu            t0,t0,1                                 #
            contadorTag++
```

```
        sw              t0,VALIDATE_TAGL($fp)
        b               contadorTag

tagALevantar:
        #Para cargar tagALevantar[j]
        lw              t4, VALIDATE_TAGL($fp)    #Cargo tag a
            levantar del stack
        lw              t5, VALIDATE_J($fp)       #Cargo en t5 j, j =
            0
        addu            t8, t4,t5                                 #
            Almacento en t8 la direc de de memoria de tagALevantar[j]
        #Para cargar text[i]
        lw              t1, VALIDATE_ARG0($fp)    # t1 = *text
        lw              t0, VALIDATE_I($fp)       #Cargo el valor de i
             almacenado en el stack
        addu            t6, t1,t0                                 #
            Almaceno la dir de memoria de text[i]
        # tagALevantar[j] = text[i];
        lb              t6, 0(t6)                                 # t6
             = text[i]
        sb              t6, 0(t8)                                 #
            tagALevantar[j] = text[i];

        #j++
        lw              t5, VALIDATE_J($fp)       #Cargo en t5 j, j =
            0
        addiu   t5, t5,1                                    #j++
        sw              t5, VALIDATE_J($fp)       #Almaceno el valor
            de j en el stack
        #i++
        lw              t0, VALIDATE_I($fp)             #Cargo en
            t0 i
        addiu   t0, t0,1                                    #i++
        sw              t0, VALIDATE_I($fp)       #Almaceno el valor
            de i en el stack
        b               whileNotFinTag


switchValidate:
        lw              a0, VALIDATE_ARG0($fp)    #Cargo en a0 *text
        lw              a1, VALIDATE_TAGL($fp)    #Cargo en a1
            tagALevantar
        lw              a2, VALIDATE_I($fp)       #Cargo en a2 la
            posicion
        la              a3, VALIDATE_CL($fp)      #Cargo en a3 el
            contadorLinea
        la              t9, analizarTag                 #Cargo la
            direccion de analizarTag en t9
        jal             t9
```

```
        sw              v0, VALIDATE_I($fp)         #Almaceno en el
            stack lo que me devuelve analizarTag
        #Cargo argumento - FREE
        lw              a0, VALIDATE_TAGL($fp)      #Cargo en a1 la
            seccion a liberar
        la              t9, myfree
        jal             t9

        #Sigo con el fin del programa
        lw              t0, VALIDATE_I($fp)                 #Cargo en
            t0 i
        #Arranca el switch
        li              t6, -2                                      #
            Cargo en t6 -2
        beq             t0, t6, errorAnidado        # Si i = -2 voy a
            errorAnidado
        li              t6, -1                                      #
            Cargo en t6 -1
        beq             t0, t6, errorNoCerrado      # si i = -1 voy a
            errorNoCerrado
        b               incrementarPos                      #Va a
            default


errorNoCerrado:
        lw              a0, VALIDATE_BF($fp)        #Cargo en a0 el
            buffer
        la              a1, ER_TAG_MAL_CERRADO   #Cargo en a1 la
             direcc de tag mal cerrado
        lw              a2, VALIDATE_CL($fp)        #Cargo en a2
            contador lineas
        la              t9, sprintf                                 #
            Cargo la direccion de sprintf en t9
        jal             t9

        lw              t0, VALIDATE_ARG1($fp)      #Cargo en t0 *
            errmsg
        lw              t1, VALIDATE_BF($fp)        #Cargo en t1 el *
            buffer
        sw              t1, 0(t0)                                   #*
            errmsg = buffer
        li              v0, 1                                       #v0
             = 1
        sw              v0, VALIDATE_RT($fp)        #return 1
        b               liberarStack


errorAnidado:
        lw              a0, VALIDATE_BF($fp)        #Cargo en a0 el
            buffer
```

```
        la              a1, ER_TAG_MAL_ANIDADO   #Cargo en a1 la
            direc de tag mal anidado
        lw              a2, VALIDATE_CL($fp)     #Cargo en a2
            contador lineas
        la              t9, sprintf                              #
            Cargo la direccion de sprintf en t9
        jal             t9

        lw              t0, VALIDATE_ARG1($fp)   #Cargo en t0 *
            errmsg
        lw              t1, VALIDATE_BF($fp)             #Cargo en
            t1 el *buffer
        sw              t1, 0(t0)                                #*
            errmsg = buffer
        li              v0, 1                                    #v0
             = 1
        sw              v0, VALIDATE_RT($fp)     #return 1
        b               liberarStack

incrementarPos:
        lw              t0, VALIDATE_I($fp)              #Cargo en
            t0 el valor de i
        addu    t0, t0, 1                                # i++
        sw              t0, VALIDATE_I($fp)              #Lo
            almaceno en el stack
        b               whileFinLinea

return_0:
        sw              zero,VALIDATE_RT($fp)

liberarStack:
        #Destruye stack frame
        lw              v0, VALIDATE_RT($fp)
        move    sp, $fp
        lw              ra, VALIDATE_RA(sp)
        lw              $fp,VALIDATE_FP(sp)
        lw              gp, VALIDATE_GP(sp)
        addu    sp, sp,VALIDATE_SS
        j               ra

.end    validate
```

Listing : analizartag.S

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

#Argumentos de la funcion
#define ATAG_ARG3          52
#define ATAG_ARG2          48
```

```
#define ATAG_ARG1           44
#define ATAG_ARG0           40

#Stack Size
#define ATAG_SS             40

#SRA
#define ATAG_RA             32
#define ATAG_FP             28
#define ATAG_GP             24

#LTA
#define TAG_A_LEVANTAR      16
#define VAR_AUX                     20

#ABA
#define ABA_3               12
#define ABA_2               8
#define ABA_1               4
#define ABA_0               0

#Constantes
#define FIN_TEXTO           0
#define SALTO_DE_LINEA 10
#define BARRA               47
#define MENOR               60
#define MAYOR               62

        .text
        .align  2
        .globl  analizarTag
        .ent    analizarTag

analizarTag:
        .frame  $fp, ATAG_SS, ra
        .set            noreorder
        .cpload t9
        .set            reorder
        #Creo el stack frame
        subu    sp,     sp,     ATAG_SS
        .cprestore ATAG_GP
        sw              ra,     ATAG_RA(sp)
        sw              $fp,ATAG_FP(sp)
        sw              gp, ATAG_GP(sp)
        move    $fp, sp
        sw              a0,ATAG_ARG0($fp)
        sw              a1,ATAG_ARG1($fp)
        sw              a2,ATAG_ARG2($fp)
        sw              a3,ATAG_ARG3($fp)
```

```
whileDistintoDeEnd:
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc el texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la pos
        addu            t0, t0,t1
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, FIN_TEXTO                                   #
            Cargo el fin de texto
        bne             t0, t6,verSiEsSaltoDeLinea              #Distino de
            fin de texto
        b               errorNoCerrado                                  #
            LLegue al fin del texto y no cerre el tag

verSiEsSaltoDeLinea:
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc el texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la pos
        addu            t0, t0,t1
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, SALTO_DE_LINEA                              #
            Cargo el salto de linea
        bne             t0, t6,verSiComienzaTag                 #Si no es
            salto de linea, salta a analizar si empieza un tag

        #Si es igual a salto de linea, tengo que sumar uno al contador de
            lineas
        lw              t3, ATAG_ARG3($fp)                              #
            Cargo la direccion de contadorLineas
        lw              t6, ATAG_ARG3($fp)                              #
            Cargo un auxiliar para no perder la referencia
        lw              t6, 0(t6)
                        #Cargo contadorLineas(int)
        addu            t6, t6,1
                        #Le sumo uno a contadorLineas
        sw              t6,     0(t3)
                        #Guardo en la direc de contadorLineas, contadorLineas
            + 1

verSiComienzaTag:
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc el texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la pos
```

```
        addu            t0, t0,t1
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, MENOR
                        #Cargo el MENOR '<'
        beq             t0, t6,comienzaTag                              #Si
            es igual a '<'salto a comienzaTag

        # Si es distinto de '<' tengo que sumar uno a pos y volver al
            principio
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo pos
        addu            t1, t1,1
                        #Si es distinto, le sumo uno a la pos
        sw              t1, ATAG_ARG2($fp)                              #
            Guardo pos++
        b               whileDistintoDeEnd                              #
            Vuelvo al principio

comienzaTag:

        #Antes encontre un '<', tengo que saltearlo sumandole uno a pos
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo pos
        addu            t1, t1,1
                        #Le sumo uno a pos
        sw              t1, ATAG_ARG2($fp)                              #
            Guardo pos++
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc del texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la pos acutal
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lbu             t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, BARRA
                        #Cargo la BARRA '/'
        bne             t0, t6, hayNuevoTag                             #Si
            text[pos] no es igual a la barra empieza un nuevo tag anidado

        #Si habia una barra, tengo que saltearla sumandole uno a pos
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo pos
        addu            t1, t1,1
                        #Le sumo uno a pos
        sw              t1, ATAG_ARG2($fp)                              #
            Guardo pos++
```

```
        sw              zero,20($fp)
                    #j = 0 (Con j itero por el tagLevantado)

        #Despues de saltear la barra analizo si el tag que me pasaron es
            igual al que estoy encontrando
whileDistintoDeCerrarTag:
        lw              t0, ATAG_ARG0($fp)                  #
            Cargo la direc del texto
        lw              t1, ATAG_ARG2($fp)                  #
            Cargo pos
        addu            t0, t1,t0
                    #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                    #Cargo el texto en la pos(cargo un char)
        li              t6, MAYOR
                    #Cargo el MAYOR '>'
        beq             t0, t6, finCerrarTag                #Si
            text[pos] == '>' salto a finCerrarTag
                                                                #
                                                                    llegue
                                                                    al
                                                                    final
                                                                    del
                                                                    tag
                                                                    y
                                                                    voy
                                                                    a
                                                                    analizar
                                                                    si
                                                                    eran
                                                                    iguales

        #Cargo tagEncontrado[j]
        lw              t2, ATAG_ARG1($fp)                  #
            Cargo el tagEncontrado
        lw              t3, VAR_AUX($fp)                    #
            Cargo el valor de j
```

```
        addu            t2, t2, t3
                        #Me paro en tagEncontrado[j]
        lb              t2, 0(t2)
                        #Cargo el caracter de tagEncontrado[j]

        #Cargo text[pos]
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc del texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo pos
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        bne             t0, t2,finCerrarTag                             #Si
            el tagEncontrado[j] es distinto al text[pos] es un error, en
            finCerrarTag lo analizo

        #Si los caracteres eran iguales hago j++ y pos++
        lw              t3, VAR_AUX($fp)                                #
            Cargo el valor de J
        addu    t3, t3,1                                                #j
            ++
        sw              t3, VAR_AUX($fp)                                #
            Guardo j++ en la variable auxiliar
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo pos
        addu    t1, t1,1                                                #
            Sumo uno a pos
        sw              t1, ATAG_ARG2($fp)                              #
            Guardo pos++
        b               whileDistintoDeCerrarTag                #Salto al
            comienzo del while

finCerrarTag:
        #Es un if doble
        lw              t2, ATAG_ARG1($fp)                              #
            Cargo la direccion de tagEncontrado
        lw              t3, VAR_AUX($fp)                                #
            Cargo el valor de j
        addu            t2, t2, t3
                        #Me paro en tagEncontrado[j]
        lb              t2, ATAG_ARG0(t2)                               #
            Cargo el caracter tagEncontrado[j]
        li              t6, FIN_TEXTO                                   #
            Cargo el fin de texto '\0'

        #Revisa si el tag se cerro bien: if(tagEncontrado[j]=='\0')
        bne             t2, t6, errorAnidado                    #Si el
            tagEncontrado no llego al fin, es un error
```

```
        #Ahora reviso si text[pos] == '>'
        lw              t0, ATAG_ARG0($fp)                      #
            Cargo la direc del texto
        lw              t1, ATAG_ARG2($fp)                      #
            Cargo pos
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, MAYOR
                        #Cargo el MAYOR '>'

        #Revisa si text[pos] == '>'
        bne             t0, t6, errorAnidado                    #Si el text[
            pos] no llego a '>' es un error
        b               devolverPosActual                       #Si
            se cumplio lo anterior tengo que devolver la pos

hayNuevoTag:
        lw              t1, ATAG_ARG2($fp)                      #
            Cargo la direc de la pos
        sw              t1, VAR_AUX($fp)                        #
            contadorTag = pos

contadorTag:
        lw              t0, ATAG_ARG0($fp)                      #
            Cargo la direc del texto
        lw              t1, VAR_AUX($fp)                        #
            Cargo pos
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, MAYOR
                        #Cargo el MAYOR '>'
        bne             t0, t6, aumentarConTag                  #Mientras
            sea distinto de MAYOR '>', aumentor el contador
        lw              t3, VAR_AUX($fp)                        #
            Cargo contadortag
        lw              t1, ATAG_ARG2($fp)                      #
            Cargo pos
        subu    t7, t3, t1                                      #
            ContadorTag = contadorTag - pos
        addiu   t7, t7, 1                                       #
            contadorTag = contadorTag + 1 por el '\0'
        sw              t7, VAR_AUX($fp)                        #
            Guardo el nuevo valor de contadorTag
        lw              a0,     VAR_AUX($fp)                    #
            cargo el argunmento de la funcion
```

```
        la              t9, mymalloc                                    #
            Cargo en t9 la direccion de la funcion
        jal             t9
                            #Llamo a la funcion malloc
        sw              v0, TAG_A_LEVANTAR($fp)         #Guardo en
            atag1 la posicion de memoria que reserve
        sw              zero, VAR_AUX($fp)                              #
            int k = 0;

cargarTagALevantar:
        lw              t0, ATAG_ARG0($fp)                              #
            Cargo la direc el texto
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la direc de la pos
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lb              t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        li              t6, MAYOR
                        #Cargo el MAYOR '>'
        bne             t0, t6, cargarCaracter          #Si no
            llegue al MAYOR, cargo el caracter en tagALevantar y aumento
            los contadores

        #Pongo el tagALevantar[k] = '\0' para indicar que llegue al final
        lw              t8, TAG_A_LEVANTAR($fp)         #Cargo el
            tag
        lw              t3, VAR_AUX($fp)                                #
            Cargo k
        addu    t6, t8, t3                                              #
            Me paro en tagALevantar[k]
        li              t7, FIN_TEXTO                                   #
            Cargo en t7 '\0'
        sb              t7, 0(t6)
                        #tagALevantar[k] = '\0'

        #Sigo con la funcion
        lw              t1, ATAG_ARG2($fp)                              #
            Cargo la pos
        addu            t1, t1, 1
                        #pos++
        sw              t1, ATAG_ARG2($fp)                              #
            Guardo la nueva posicion

        #Cargo los argumentos
        lw              a0, ATAG_ARG0($fp)                              #a0
             = texto
        lw              a1, TAG_A_LEVANTAR($fp)         #a1 =
            tagALevantar
```

```
        lw              a2, ATAG_ARG2($fp)                      #a2
            = pos
        lw              a3, ATAG_ARG3($fp)                      #a3
            = contadorlineas
        la              t9, analizarTag
        jal             t9
        sw              v0, ATAG_ARG2($fp)                      #
            Reemplazo la nuevo posicion

        #Cargo argumento
        lw              a0, TAG_A_LEVANTAR($fp)         #Cargo en
            la seccion a liberar el espacio allocado por malloc
        la              t9, myfree
        jal             t9
        b               switchPos

cargarCaracter:
        lw              t0, ATAG_ARG0($fp)                      #
            Cargo la direc el texto
        lw              t1, ATAG_ARG2($fp)                      #
            Cargo la direc de la pos
        addu            t0, t1,t0
                        #Muevo la direc del texto a la pos
        lbu             t0, 0(t0)
                        #Cargo el texto en la pos(cargo un char)
        lw              t8, TAG_A_LEVANTAR($fp)         #t8 espacio
            de memoria para tagALevantar
        lw              t3, VAR_AUX($fp)                        #
            Cargo el valor de k
        addu    t6, t8, t3                                      #
            tagALevantar[k]
        sb              t0, 0(t6)
                        #tagALevantar[k] = texto[pos]

        #Actualizo variables
        #k++
        lw              t3, VAR_AUX($fp)
        addiu   t3, t3, 1
        sw              t3, VAR_AUX($fp)

        #pos++
        lw              t1, ATAG_ARG2($fp)                      #
            Cargo la pos
        addiu   t1, t1, 1
        sw              t1, ATAG_ARG2($fp)                      #
            Guardo la nueva posicion
        b               cargarTagALevantar


aumentarConTag:
```

```
        lw              t7, VAR_AUX($fp)
        addiu   t7, t7, 1                                       #
            contadorTag++
        sw              t7, VAR_AUX($fp)
        b               contadorTag

switchPos:
        lw              v0, ATAG_ARG2($fp)                      #
            Restauro el valor de v0 luego de analizar tag
        #Comprara lo que devolvio analizarTag, v0 = analizarTag()
        li              t6, -1
                    #Cargo -1 en t6
        beq             v0, t6, errorNoCerrado                  #Si v0 es
            igual a -1, es un errorNoCerrad
        li              t6, -2
                    #Cargo -2 en t6
        beq             v0, t6, errorAnidado                    #si v0 es
            igual a -2, es un errorAnidado
        #Si no devolvio ningun error
        addu    v0, v0, 1                                       #Le
             sumo uno a pos
        sw              v0, ATAG_ARG2($fp)                      #
            Guardo la pos
        b               whileDistintoDeEnd

devolverPosActual:
        lw              v0, ATAG_ARG2($fp)                      #
            Muevo el t1 que tiene la pos a v0
        b               salirATAG
                    #Recupero los registros

errorNoCerrado:
        li              v0,-1
                    # return -1;
        b               salirATAG

errorAnidado:
        li              v0,-2
                    # return -2;
        b               salirATAG

salirATAG:
        #Destruye stack frame
        move    sp, $fp
        lw              ra, ATAG_RA(sp)
        lw              $fp,ATAG_FP(sp)
        lw              gp, ATAG_GP(sp)
        addu    sp, sp,ATAG_SS
        j               ra
```

```
.end analizarTag
```

3. Diagrama de Stack

3.1. Validate

Stack Size Validate 56

errmsg	60
text	56
Padding	50
RA	48
FP	44
GP	40
VALIDATE RT	36
VALIDATE BF	32
VALIDATE TAGL	28
VALIDATE J	24
VALIDATE CL	20
VALIDATE I	16
ABA3	12
ABA2	8
ABA1	4
ABA0	0

3.2. Analizar Tag

Stack Size Analizar Tag 40

Contador Lineas	52
Posicion	48
Tag Encontrado	44
text	40
Padding	36
RA	32
FP	28
GP	24
VAR AUX	20
TAGALEVANTAR	16
ABA3	12
ABA2	8
ABA1	4
ABA0	0

4. Conclusiones

De este trabajo practico se pudo aprender como programar con el conjunto de instrucciones Assembly de MIPS32, asi como tambien la utilizacion correcta de la ABI de la catedra, entendiendo de esta manera como funciona una computadora a bajo nivel.

5. Referencias

[1] GXemul, http://gavare.se/gxemul/.
[2] The NetBSD project, http://www.netbsd.org/.
[3] System V application binary interface, MIPS RISC processor supplement(third edition). Santa Cruz Operations, Inc.
[4] func call conv.pdf, en el area de Material de los archivos del grupo de Yahoo.