

Trabajo Práctico 1:

Conjunto de instrucciones MIPS

Jimenez, Ruben, *Padrón Nro. 92.402*
rbnm.jimenez@gmail.com

Reyero, Felix, *Padrón Nro. 92.979*
felixcarp@gmail.com

Suárez, Emiliano, *Padrón Nro. 78.372*
emilianosuarez@gmail.com

Primera Entrega: 30/04/2015

1er. Cuatrimestre de 2015

66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Se implementó una versión simplificada del programa **sha1** de UNIX.
Para nuestra implementación.

1. Introducción

Este Trabajo Práctico pretende familiarizarse con la programación en assembly y el concepto de ABI.

Para ello, implementaremos el algoritmo **sha1** de UNIX en código *Assembly*, mientras que la interpretación de argumentos del programa y la lectura de archivos será realizada en lenguaje *C*.

Además, se utilizará *GXemul* para simular una máquina *MIPS* corriendo una versión reciente del sistema operativo *NetBSD*.

El programa implementado, muestra por *stdout* un checksum generado a partir del contenido de los archivos pasados por parámetro. En caso de no especificarse algún archivo, se mostrarán un checksum a partir de lo ingresado por *stdin*.

2. Implementación

2.1. Arquitectura

En una primera versión se trabajó con una función *sha* íntegramente desarrollada en *Assembly*, con un stack mas grande. Pero tuvimos problemas con el *malloc* al intentar reservar memoria para los bloques a procesar por el algoritmo. Como una alternativa, probamos utilizar la función *malloc* de *C* desde *Assembly*, realizando *syscall* e incluso, utilizando la versión en *Assembly* de *mymalloc.S* que se encuentra en el grupo.

Con ninguna de estas opciones lograr que la función *sha* funcionara correctamente, por lo que decidimos modularizarla de la siguiente manera.

main.c

Cuerpo principal del programa, donde se realiza la lectura de los parámetros de los archivos.

Cada uno de estos archivos, es guardado en memoria dinámica con un tamaño máximo de 1000 bytes (utilizando *malloc*). En caso que el tamaño del archivo supere dicho tamaño, se reasigna memoria dinámica mediante la utilización de *realloc*, para disponer un bloque de memoria donde quepa el archivo (bloque de igual tamaño al tamaño del archivo).

relleno.S

Donde se encuentra la implementación en *Assembly* de la función *calcularRelleno*.

Dicha función cuenta con un stack donde se alacena la longitud original así como la longitudRelleno que es la longitud que incluye el relleno para poder determinar la cantidad de bloques finales a procesar.

También se almacena en el stack la variable *cantBloques* que determina la cantidad de bloques a procesar.

La dificultad de este método fue la de tratar longitudes que ocupan 64 bits, para ello se tuvo que manejar dos registros uno que tenga los 32 bits mas significativos y otro para los 32 restantes, usando el carry que produce la suma de la parte menos significativa para actualizar la parte mas significativa de la longitud final.

trozo.S

Donde se encuentra la implementación en *Assembly* de la función *cargarTrozos*.

Dicha función, contiene en su stacks bytes reservados para almacenar las variables “i” y “mascara”. En el cuerpo de la función se hace uso de los 8 registros temporales para distintas asignaciones necesarias.

Luego de guardar los parámetros con los que viene la función, inicializar “i” y cargar la “mascara” a un temporal, genera las 16 primeras palabras del vector de 80, trayendo byte a byte desde “bloques” y generando con operaciones lógicas (and, or) y movimientos (sll) el word final que se guarda en “trozos”.

Luego desde la posición 16 hasta la 80 con operaciones lógicas y rotaciones (xor y rol) se genera el resto del contenido del vector operando con los 16 words anteriores.

algoritmo.S

Donde se encuentra la implementación en *Assembly* de la función *algoritmoSha1*.

Dicha función, además de los registros obligatorios, contiene en su stack siete “saved registers” s0..s6 que almacenan los punteros hacia los valores de “a,b,c,d,e,f” y los registros para almacenar las variables “i”, “k” y “temp”. En el cuerpo de la función se hace uso de los 8 registros temporales para distintas asignaciones necesarias.

Luego de guardar los parámetros con los que viene la función e inicializar “i”, se generan 4 ramas distintas de condicionales con procesos distintos (dependiendo el valor de “i” en el for) donde a través de operaciones lógicas (and, not, xor, or) generan un valor que se almacena en “f” y se carga una constante “k” determinada para que luego se terminen de generar los valores que devuelve la función y se asignen a sus respectivas variables.

Las funciones *calcularRelleno*, *cargarTrozos* y *algoritmoSha1*, son llamadas desde la función *sha1* de *main.c*.

Los stacks de cada una de ellas que pueden observarse a continuación:

calcularRelleno	
Dir Mem	Valor
48	
44	ra
40	fp
36	gp
32	
28	longOrig
24	longOrig
20	longRelleno
16	longRelleno
12	a3
8	a2
4	a1
0	a0

cargarTrozos	
Dir Mem	Valor
36	
32	ra
28	fp
24	gp
20	mascara
16	i
12	a3
8	a2
4	a1
0	a0

algoritmoSha1	
Dir Mem	Valor
76	
72	ra
68	fp
64	gp
60	
56	
52	temp
48	k
44	i
40	s6
36	s5
32	s4
28	s3
24	s2
20	s1
16	s0
12	a3
8	a2
4	a1
0	a0

2.2. Diseño

Se desarrollo un programa que realiza la lectura a través del *stdin* o a través de archivos que se reciben por parámetro.

El comando acepta 2 parámetros para mostrar la Ayuda y la Versión del programa:

```
$ ./sha -h
$ ./sha --help
```

Para desplegar la ayuda del comando. Y los siguientes comandos para mostrar la versión:

```
$ ./sha -V
$ ./sha --version
```

Inicialmente el programa revisa la cadena de parametros ingresada y determina si el checksum debe generarse a partir de lo ingresado por *stdin* o a través del contenido del (o los) archivo(s).

Para esta última opción, se procesan los archivo de uno por vez, y para cada uno de ellos se genera el checksum a partir de sus datos.

Primero se obtiene el largo total del archivo, luego se llama a la función *calcularRelleno*, donde se obtiene una *longitudRelleno* que dividida por 8 determina la longitud del vector que se asigna en memoria dinámica para trabajar con el archivo completo.

Para completar el archivo, se llama a la función de C “*asignarDatos*” la cual se encarga de completar el vector anteriormente pedido con las características correspondientes que pide el preprocesamiento del algoritmo sha1 (a saber: el

relleno consiste en un uno seguido de los ceros que sean necesarios, aunque el mensaje ya tenga la longitud deseada se debe efectuar el relleno, por lo que el número de bits de dicho relleno está en el rango de 1 a 512 bits, luego se le añade un bloque de 64 bits que represente la longitud del mensaje original antes de ser relleno).

Es decir, la totalidad del archivo mas el relleno que lo completa, queda alojado en memoria dinámica. A continuación se procede con la función sha1.

Una vez dentro de la función sha1, se llama a la función “calcularRelleno”, donde se obtiene una longitudRelleno que, dividida por 512, determina la cantidad de bloques finales a procesar. Se inicializan las variables y por una decisión de diseño se decidió trabajar con un vector de 80 palabras en memoria dinámica porque nos parecía que el stack quedaba muy engorroso para trabajarlo.

Por esta elección, tuvimos problemas con la asignación de memoria dinámica desde assembly por lo cual se dividió (como se menciona en este informe) en partes la función.

A continuación se procede a trabajar con el bloque de datos en un while donde primero se carga el vector de 80 en “cargarTrozos” y luego el algoritmo propiamente dicho en “algoritmosha1” para finalmente devolver el valor en 160 bits.

2.3. Compilación

Se creó un archivo *Makefile* que permite compilar tanto la versión en *MIPS*, como en la versión implementada completamente en *C*.

El contenido de este archivo, puede verse a continuación:

```
1 CC=gcc -g -O0
2 CFLAGS=-Wall -lm
3 LDFLAGS=
4 OBJ_DIR=/
5 SOURCES=main.c
6 SOURCES_ANSI=ansi/main.c
7 MIPS_FILES=algoritmo.S relleno.S trozo.S
8
9 OBJECTS=$(SOURCES:.c=.o)
10 EXECUTABLE=sha
11
12 mips:
13     $(CC) $(CFLAGS) $(SOURCES) $(MIPS_FILES) -o $(EXECUTABLE)
14
15 c:
16     $(CC) $(CFLAGS) $(SOURCES_ANSI) -o $(EXECUTABLE)
17
18 clean:
19     rm -rf $(EXECUTABLE) $(OBJECTS)
```

../src/Makefile

Para compilarlo, se debe abrir una terminal en la carpeta donde están alojados los archivos fuentes (*src/*) y se ejecuta el siguiente comando:

```
../src$ make mips
```

Para compilar la versión en *Assembly* de *MIPS*.

O se puede utilizar esta opción:

```
../src$ make c
```

Para generar la versión en *C*.
Ambas opciones, generan el ejecutable *sha*.

3. Casos de Prueba

Algunos de los casos de pruebas realizados, pueden observarse a continuación:

```
root@:/home/gxemul/tprub/div# sha1 hola
SHA1 (hola) = ef443fee4da6bfb41651930de7ad99f29ed9f079
root@:/home/gxemul/tprub/div# ./sha hola
ef443fee4da6bfb41651930de7ad99f29ed9f079

root@:/home/gxemul/tprub/div# sha1 vacio
SHA1 (vacio) = da39a3ee5e6b4b0d3255bfef95601890afd80709
root@:/home/gxemul/tprub/div# ./sha vacio
da39a3ee5e6b4b0d3255bfef95601890afd80709

root@:/home/gxemul/tprub/div# sha1 prueba-dog
SHA1 (prueba-dog) = 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
root@:/home/gxemul/tprub/div# ./sha prueba-dog
2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

root@:/home/gxemul/tprub/div# sha1 prueba-cog
SHA1 (prueba-cog) = de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
root@:/home/gxemul/tprub/div# ./sha prueba-cog
de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
```

4. Conclusiones

El presente trabajo permitió la familiarización con las herramientas de compilación de código C y código assembly en un entorno que emula la arquitectura MIPS 32, asegurando la portabilidad del programa.

Además nos permitió conocer en detalle como se comporta el stack de una función en la programación en *Assembly*. Para esto último, fue de gran ayuda conocer previamente la implementación de las funciones en language C, para luego hacer la “traducción” a *Assembly* teniendo en cuenta la cantidad de argumentos, variables locales, tamaño de los datos, etc.

5. Apéndice

5.1. Código Fuente: main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define ERROR_OPEN_FILE 10
6 #define MAX_COUNT_FILES 50
7 #define HASH_LENGTH 20
8 #define MAX_FILE_SIZE 1000
9
10 void printHelp();
11 void printVersion();
12 void printError(char* msgError, int codeError);
13 void setFileSize(FILE* fp, long long int *length);
14 int readFromStdInput(int argumentCount);
15 int sha1(unsigned char *resultado, char *nombre_archivo, unsigned
    long long longitudOriginal);
16 unsigned int leftrotate(unsigned int valor, int desplazamiento);
17 void asignarDatos(FILE* fp, char *bloques, long long int
    tamanoOriginal, long long int longitudRelleno);
18 void showChecksum(unsigned char *result);
19
20 long long int calcularRelleno(long long int longitudOriginal);
21 void cargarTrozos(char *bloque, unsigned int *Trozos);
22 void algoritmoSha1(unsigned int *Trozos, unsigned *a, unsigned *b,
    unsigned *c, unsigned *d, unsigned *e);
23
24 int main(int argc, char* argv[]) {
25     int i = 0;
26     char* param;
27     char* files[MAX_COUNT_FILES];
28     unsigned char *result;
29     int fileCount = 0;
30     FILE* fp;
31     long long int length;
32
33     if (readFromStdInput(argc)) {
34         int c;
35
36         fp = fopen("archivoAuxiliar.txt", "w+");
37         while ((c = fgetc(stdin)) != EOF) {
38             fputc(c, fp);
39         };
40
41         setFileSize(fp, &length);
42         long long int longitudRelleno = calcularRelleno(length);
43         char *bloques = malloc(longitudRelleno/8);
44
45         result = malloc(HASH_LENGTH);
46         // se almacena el tamanoOriginal al final
47         asignarDatos(fp, bloques, length, longitudRelleno);
48         sha1(result, bloques, length);
49         showChecksum(result);
50
51         fclose(fp);
52         remove("archivoAuxiliar.txt");
53
54         return 0;
55     }
```

```

56 } else if (argc >= 2) { // Parse arguments
57     param = *(argv + 1);
58     if ((strcmp(param, "-h") == 0) || (strcmp(param, "--help"
59         == 0) ) {
60         printHelp();
61     }
62     else if ((strcmp(param, "-V") == 0) || (strcmp(param, "--
63         version") == 0)) {
64         printVersion();
65     }
66 }
67 // Search for files
68 for(i = 1; i < argc; i++) {
69     if (*argv[i] != '-') {
70         files[fileCount++] = argv[i];
71     }
72 }
73 // Process each file
74 for(i = 0; i < fileCount; i++) {
75     result = malloc(HASH_LENGTH);
76
77     if (NULL == (fp = fopen(files[i], "r"))) {
78         fprintf(stderr, "Files '%s' doesn't exist.", files[i]);
79         exit(ERROR_OPEN_FILE);
80     }
81
82     setFileSize(fp, &length);
83     long long int longitudRelleno = calcularRelleno(length);
84     char *bloques = malloc(longitudRelleno/8);
85
86     fp = fopen(files[i], "r");
87
88     // se almacena el tamañoOriginal al final
89     asignarDatos(fp, bloques, length, longitudRelleno);
90     sha1(result, bloques, length);
91     fclose(fp);
92 }
93
94 showChecksum(result);
95
96 return 0;
97 }
98
99 void setFileSize(FILE* fp, long long int *length) {
100     int character;
101     char *start;
102     int n = 0;
103
104     fseek(fp, 0, SEEK_END);
105     *length = ftell(fp);
106
107     fseek(fp, 0, SEEK_SET);
108
109     start = malloc(MAX_FILE_SIZE);
110
111     if ((*length) > MAX_FILE_SIZE) {
112         start = realloc(start, (*length));
113     }
114
115     *length *= 8; // devuelve el tamaño en bits

```

```

116     while ((character = fgetc(fp)) != EOF) {
117         start[n++] = (char)character;
118     }
119     start[n] = '\0';
120 }
121
122 void printHelp()
123 {
124     fprintf(stdout, "$ tp1 -h\n");
125     fprintf(stdout, "Usage:\n");
126     fprintf(stdout, "    tp1 -h\n");
127     fprintf(stdout, "    tp1 -V\n");
128     fprintf(stdout, "    tp1 [file ...]\n");
129     fprintf(stdout, "Options:\n");
130     fprintf(stdout, "    -V, --version    Print version and quit.\n");
131     ;
132     fprintf(stdout, "    -h, --help      Print this information and
133         quit.\n\n");
134     fprintf(stdout, "Examples:\n");
135     fprintf(stdout, "    tp1 foo\n");
136     fprintf(stdout, "    echo \"hello\" | tp1\n\n");
137 }
138 int shal(unsigned char *resultado, char *bloques, unsigned long
139         long longitudOriginal)
140 {
141     long long int longitudRelleno = calcularRelleno(
142         longitudOriginal);
143     // Preprocesamiento
144     // longArchivo/tamano bloque    bloque = 512bits
145     long long int cantBloques = longitudRelleno/512;
146
147     // procesar el bloque en 4 rondas de 20 pasos cada ronda
148     // la memoria temporal cuenta con 5 registros ABCDE
149     unsigned A=0x67452301;
150     unsigned B=0xEFCDAB89;
151     unsigned C=0x98BADCFE;
152     unsigned D=0x10325476;
153     unsigned E=0xC3D2E1F0;
154
155     // big endian
156     unsigned int trozos[80];
157
158     int i;
159     unsigned a = 0;
160     unsigned b = 0;
161     unsigned c = 0;
162     unsigned d = 0;
163     unsigned e = 0;
164
165     while(cantBloques--)
166     {
167         cargarTrozos(bloques, trozos);
168
169         a = A;
170         b = B;
171         c = C;
172         d = D;
173         e = E;

```

```

174         algoritmoSha1(trozos, &a, &b, &c, &d, &e);
175
176         A += a;
177         B += b;
178         C += c;
179         D += d;
180         E += e;
181     }
182
183     // hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2
184     //         leftshift 64)
185     //         or (h3 leftshift 32) or h4
186     for(i = 0; i < 4; i++)
187     {
188         resultado[i] = (A >> (24 - 8 * i));
189         resultado[i + 4] = (B >> (24 - 8 * i));
190         resultado[i + 8] = (C >> (24 - 8 * i));
191         resultado[i + 12] = (D >> (24 - 8 * i));
192         resultado[i + 16] = (E >> (24 - 8 * i));
193     }
194
195     return 0;
196 }
197
198 // un bloque tiene 64 bytes
199 void asignarDatos(FILE *fp, char *bloques, long long int
200                 tamanioOriginal, long long int longitudRelleno)
201 {
202     char caracter;
203     int indice = 0;
204     int i = 0;
205     int cantBitsRelleno = 0;
206     char relleno = 0x80; // minimo relleno
207     char rellenoCero = 0x00;
208     unsigned mascara = 0x00000000000000FF;
209
210     while( (caracter = getc(fp)) != EOF)
211     {
212         *(bloques + indice) = caracter;
213         indice++;
214     }
215
216     cantBitsRelleno = (longitudRelleno - tamanioOriginal);
217
218     // restamos 64 bits de tamanio y los 8 bits basicos de relleno;
219     cantBitsRelleno = cantBitsRelleno - 64 - 8;
220     *(bloques + indice) = relleno;
221     indice++;
222
223     for(i = 0; i < (cantBitsRelleno / 8); i++)
224     {
225         *(bloques + indice) = rellenoCero;
226         indice++;
227     }
228
229     for(i = 0; i < 8; i++)
230     {
231         *(bloques + indice) = (tamanioOriginal >> (56 - 8 * i)) & mascara;
232         indice++;
233     }

```

```

234 }
235
236 // devuelve el tamaño en bits
237 long long int calcularTamañoArchivo(char* nombreFile)
238 {
239     long long int tamaño =0;
240     FILE *fp = fopen(nombreFile, "r");
241     fseek(fp,0,SEEK_END);
242     tamaño = ftell(fp)*8;
243     return tamaño;
244 }
245
246
247 unsigned int leftrotate(unsigned int valor,int desplazamiento)
248 {
249     desplazamiento %=32;
250     unsigned retorno;
251     retorno = (valor << desplazamiento) | (valor >> (32 -
252         desplazamiento));
253     return retorno;
254 }
255
256 void printVersion()
257 {
258     fprintf(stdout, "Copyright (c) 2015\n");
259     fprintf(stdout, "Conjunto de instrucciones MIPS. v1.0.0\n\n");
260 }
261
262 void printError(char* msgError, int codeError)
263 {
264     fprintf(stderr, "%s\n", msgError);
265     exit(codeError);
266 }
267
268 void showChecksum(unsigned char *result) {
269     int i;
270     for(i = 0; i < 20; i++) {
271         unsigned char aux = result[i];
272         aux<<=4;
273         aux>>=4;
274         printf("%x", (result[i]>>4));
275         printf("%x",aux);
276     }
277     printf("\n");
278 }
279
280 int readFromStdInput(int argumentCount) {
281     return (int)(argumentCount < 2);
282 }

```

../src/main.c

5.2. Código Fuente: relleno.S

```

1
2 #include <mips/regdef.h>
3
4 .text
5 .align 2
6
7 .globl calcularRelleno

```

```

8      .ent    calcularRelleno
9
10     calcularRelleno:
11         .frame $fp, 48, ra
12         .set noreorder
13         .cpload t9
14         .set reorder
15         subu  sp, sp, 48
16         .cpstore 0
17
18
19         sw   ra, 44(sp)
20         sw   $fp, 40(sp)
21         sw   gp, 36(sp)
22         move  $fp, sp
23
24         sw   a0, 48($fp)
25         sw   a1, 52($fp)
26         #sw  a2, 56($fp)
27         #sw  a3, 60($fp)
28
29         #obtencion del tamaño del relleno
30
31         #-----operaciones en 64 bits-----
32
33         li    t0, 0
34         li    t1, 0      #el tamaño debe venir en bits
35         move  t0, a0      #t1|t0 = longArchivoRelleno
36         move  t1, a1
37         li    t4, 512
38
39     bucle_relleno:
40         remu  t2, t0, t4   #calculo el valor de la longitud en modulo 512 (
41             resto)
42         beqz  t2, fin_bucle_relleno #no es necesario considerar la parte
43             mas significativa
44         #suma de los 64 bits
45         addiu t2, t0, 8     #agrego 1 byte
46         sltu  t3, t2, t0    #(t3 =1) de carry si el resultado es mas chico
47             que el sumando
48         move  t0, t2        #volvamos el valor a t0
49         beqz  t3, bucle_relleno
50         addiu t1, t1, 1     #sumamos 1 a la parte mas significativa
51         b     bucle_relleno #solucionar la suma en 64 bits
52
53     fin_bucle_relleno:
54         slti  t2, t0, 65    #si t0 < 65 entonces t2 =1 y agregamos 512 bits
55         beqz  t2, obtener_bloque
56         addiu t2, t0, 512   #agrega un bloque de 512 extra
57         sltu  t3, t2, t0
58         move  t0, t2
59         beqz  t3, obtener_bloque #si no hay acarreo no sumar al mas
60             significativo
61         addiu t1, t1, 1     #t1|t0 = longArchivoRelleno
62
63     obtener_bloque:
64         sw    t0, 16($fp)   #guardo la longitud relleno en 64 bits en el
65             stack
66         sw    t1, 20($fp)
67         sw    a0, 24($fp)   #guardo la longitud original en 64 bits en el
68             stack

```

```

63  sw  a1,28($fp)
64
65  move v0,t0
66  move v1,t1
67
68  lw  $fp, 40(sp)
69  lw  ra, 44(sp)
70  lw  gp, 36(sp)
71  addu sp, sp, 48
72
73  # Retorno.
74  #
75  j  ra
76  .end  calcularRelleno

```

../src/relleno.S

5.3. Código Fuente: trozo.S

```

1  #
2  #t8 puntero a bloques
3  #t9 puntero a trozos
4  #
5
6  #include <mips/regdef.h>
7
8  .text
9  .align 2
10 .globl cargarTrozos
11 .ent  cargarTrozos
12
13 cargarTrozos:
14 .frame $fp, 48, ra
15 .set noreorder
16 .set reorder
17 subu sp, sp, 40
18 .cprestore 0
19
20 sw  gp,24(sp)
21 sw  $fp,28(sp)
22 sw  ra,32(sp)
23 move $fp, sp
24
25 #guardo los parametros con que vine #void cargarTrozos(char *
    bloques, int *indice, unsigned int *trozos)
26
27 sw  a0, 40($fp)  #ptero a bloques
28 sw  a1, 44($fp)  # ptero a trozos
29 sw  a2, 48($fp)  # nada
30 sw  a3, 52($fp)  # nada
31
32 #cargo los parametros
33 move t8,a0
34 move t9,a1
35
36 sw  zero,16($fp)  # i=0
37 li  t0,255  # mascara 0x000000ff
38 sw  t0,20($fp)  #mascara en 28
39
40 FOR16PRIMEROS:
41  #traigo los datos

```

```

42 lw t0,16($fp)    #t0 cargo i
43 lw t5,20($fp)    #t5 = mascara
44
45 #comienzo
46
47 lbu t6,0(t8)      #t6 cargo el byte al que apunta (t4 = bloques +
                     indice), lo carga de 0 a 7
48 and t6,t6,t5      #t6 con el and de la mascara, queda 00 00 00
                     ALGO
49 sll t6,t6,8        #lo nuevo 8 lugares, quedaria 00 00 ALGO 00
50
51 addiu t8,t8,1      #(*(bloques+indice+1)
52 lbu t7,0(t8)      #t7 cargo el byte al que apunta (t4 = bloques +
                     indice + 1), lo carga de 0 a 7
53 and t7,t7,t5      #t7 con el and de la mascara, queda 00 00 00
                     ALGO
54 or t6,t6,t7        #suma logica de t6 y t7, quedaria 00 00 ALGO ALGO
55 sll t6,t6,8        #nuevo quedaria 00 ALGO ALGO 00
56
57 addiu t8,t8,1      #(*(bloques+indice+2) (ya habia sumado 1
                     antes)
58 lbu t7,0(t8)      #t7 cargo el byte al que apunta (t4 = bloques +
                     indice + 2), lo carga de 0 a 7
59 and t7,t7,t5      #t7 con el and de la mascara, queda 00 00 00
                     ALGO
60 or t6,t6,t7        #suma logica de t6 y t7, quedaria 00 ALGO ALGO
                     ALGO
61 sll t6,t6,8        #nuevo quedaria ALGO ALGO ALGO 00
62
63 addiu t8,t8,1      #(*(bloques+indice+3) (ya habia sumado 1
                     antes)
64 lbu t7,0(t8)      #t6 cargo el byte al que apunta (t4 = bloques +
                     indice + 3), lo carga de 0 a 7
65 and t7,t7,t5      #t6 con el and de la mascara, queda 00 00 00
                     ALGO
66 or t6,t6,t7        #suma logica de t6 y t7, quedaria ALGO ALGO ALGO
                     ALGO
67
68 #GUARDO
69 sw t6, 0(t9)      #guardo t6 en (t2 = trozos + i*4)
70
71 #AUMENTO VARIABLES
72
73
74 addiu t0,t0,1      #i = i + 1
75 sw t0,16($fp)      #guardo
76
77 addiu t9,t9,4      # avanzo a la sig palabra
78 addiu t8,t8,1      # avanzo al sig byte
79
80 addiu t7,t0,-16
81 beqz t7,FOR16TO80
82
83 b FOR16PRIMEROS
84
85
86 FOR16TO80:
87 #traigo los datos
88 lw t0,16($fp)      #t0 cargo i
89
90 addiu t9,t9,-12     #t3 apunta a trozos(i - 3).. 3 lugares = 4
                     bytesx3 = 12 bytes

```



```

91 lw t4,0(t9)    #t4 traigo la palabra a la q apunta t3
92 addiu t9,t9,12
93
94 addiu t9,t9,-32 #t5 apunta a trozos(i - 8)
95 lw t6,0(t9)    #t6 traigo la palabra a la que apunta t5
96 addiu t9,t9,32
97
98 xor t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8))
99
100 addiu t9,t9,-56 #t5 apunta a trozos(i - 14)
101 lw t6,0(t9)    #t6 traigo la palabra a la que apunta t5
102 addiu t9,t9,56
103
104 xor t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8)) ^
    *(trozos + (i-14))
105
106 addiu t9,t9,-64 #t5 apunta a trozos(i - 16)
107 lw t6,0(t9)    #t6 traigo la palabra a la que apunta t5
108 addiu t9,t9,64
109
110 xor t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8)) ^
    *(trozos + (i-14)) ^ *(trozos + (i-16));
111
112 rol t4,t4,1     #leftrotate 1 Â¿?
113
114 sw t4,0(t9)     #guardo t4 en t2 = trozos + i*4 es decir apunta a
    trozos[i]
115
116 addiu t0,t0,1   #i = i + 1
117 sw t0,16($fp)  #guardo
118 addiu t9,t9,4   #t2 = avanza a la siguiente palabra
119
120 addiu t7,t0,-80
121 beqz t7,FinDelFor
122
123 b FOR16TO80
124
125 FinDelFor:
126
127 lw $fp,28(sp)
128 lw ra, 32(sp)
129 lw gp, 24(sp)
130 addu sp, sp, 40
131
132 # Retorno.
133 #
134 j ra
135 .end cargarTrozos

```

../src/trozo.S

5.4. Código Fuente: algoritmo.S

```

1 #
2 #a0 puntero a trozos
3 #a1 a
4 #a2 b
5 #a3 c
6 #a4 d
7 #a5 e

```

```

8  #
9
10 #include <mips/regdef.h>
11
12 .text
13 .align 2
14 .globl algoritmoSha1
15 .ent  algoritmoSha1
16
17 algoritmoSha1:
18 .frame $fp, 80, ra
19 .set noreorder
20 .cload t9
21 .set reorder
22 subu  sp, sp, 80
23 .cprestore 0
24
25 sw   ra,72(sp)
26 sw   $fp,68(sp)
27 sw   gp,64(sp)
28
29 move  $fp, sp
30
31 # Salvo los registros s0..s6, ya que se van a utilizar.
32 sw   s0, 16($fp)
33 sw   s1, 20($fp)
34 sw   s2, 24($fp)
35 sw   s3, 28($fp)
36 sw   s4, 32($fp)
37 sw   s5, 36($fp)
38 sw   s6, 40($fp)
39
40 sw   a0, 80($fp)    # trozos
41 sw   a1, 84($fp)    # a
42 sw   a2, 88($fp)    # b
43 sw   a3, 92($fp)    # c
44
45 la   t0,4(a3)      # dir
46 la   t1,8(a3)      # dir
47
48 sw   t0, 96($fp)    # d
49 sw   t1, 100($fp)   # e
50
51
52 #-----algoritmo-----
53
54     # s1 = a
55     # s2 = b
56     # s3 = c
57     # s4 = d
58     # s5 = e
59     # s6 = f (funcion)
60 # cargo variables en s0..s6
61
62 move  s0,a0
63 lw   s1,0(a1)
64 lw   s2,0(a2)
65 lw   s3,0(a3)
66 lw   s4,4(a3)
67 lw   s5,8(a3)
68
69 #inicializar i de vuelta

```

```

70  sw  zero,44($fp)    # i = 0
71  lw  t6,44($fp)      # traigo i, t6 = i
72
73  PROCESO0A19:
74  lw  t6,44($fp)      # traigo i, t6 = i
75  move s6,zero
76
77  and  s6,s2,s3        #en f(s6) <— (b & c)
78  not  t0,s2          #t0 niego b
79  and  t1,t0,s4        #en t1 <— ((~b) & d)
80  xor  s6,s6,t1        #f(s6) <— (b & c) ^ ((~b) & d)
81
82  lw  t2,ctek1         #cargo en t2, k1
83  sw  t2,48($fp)       #es donde esta k en el stack
84
85
86  jal  ASIGNACIONTEMPORAL
87
88  addiu t6,t6,1
89  sw  t6,44($fp)
90
91  addiu t7,t6,-20
92  beqz t7,PROCESO20A39
93
94  b  PROCESO0A19
95
96  PROCESO20A39:
97  lw  t6,44($fp)      # traigo i, t6 = i
98  move s6,zero
99
100  xor  s6,s2,s3        #f <— b ^ c
101  xor  s6,s6,s4        #f <— b ^ c ^ d
102
103  lw  t0,ctek2         #cargo en t0, k2
104  sw  t0,48($fp)       #es donde esta k en el stack
105
106  jal  ASIGNACIONTEMPORAL
107
108  addiu t6,t6,1
109  sw  t6,44($fp)
110
111  addiu t7,t6,-40
112  beqz t7,PROCESO40A59
113
114  b  PROCESO20A39
115
116  PROCESO40A59:
117  lw  t6,44($fp)      # traigo i, t6 = i
118  move s6,zero
119
120  and  s6,s2,s3        #en f(s6) <— (b & c)
121  and  t0,s2,s4        #en t0 <— (b & d)
122
123  or   s6,s6,t0        #f <— (b & c) | (b & d)
124
125  and  t0,s3,s4        #en t0 <— (c & d)
126
127  or   s6,s6,t0        #f <— (b & c) | (b & d) | (c & d)
128
129  lw  t0,ctek3         #cargo en t0, k3
130  sw  t0,48($fp)       #es donde esta k en el stack
131

```

```

132 jal ASIGNACIONTEMPORAL
133
134 addiu t6,t6,1
135 sw t6,44($fp)
136
137 addiu t7,t6,-60
138 beqz t7,PROCESO60A79
139
140 b PROCESO40A59
141
142 PROCESO60A79:
143 lw t6,44($fp) # traigo i, t6 = i
144 move s6,zero
145
146 xor s6,s2,s3 #f <— b ^ c
147 xor s6,s6,s4 #f <— b ^ c ^ d
148
149 lw t0,ctek4 #carga en t0, k4
150 sw t0,48($fp) #es donde esta k en el stack
151
152 jal ASIGNACIONTEMPORAL
153
154 addiu t6,t6,1
155 sw t6,44($fp)
156
157 addiu t7,t6,-80
158 beqz t7,return_algoritmo
159
160 b PROCESO60A79
161
162 ASIGNACIONTEMPORAL:
163
164 #ESTO FINALIZA EL FOR DE 80 CON EL ALGORITMO PER SE
165
166 rol t0,s1,5 #en t0 leftrotate a 5
167 addu t0,t0,s6 #t0 = t0 + f
168 addu t0,t0,s5 #t0 = t0 + f + e
169 lw t1,48($fp) #t1 cargo k
170 addu t0,t0,t1 #t0 = t0 + f + e + k
171
172 lw t1,44($fp) #t1 cargo i
173 sll t2,t1,2 #t2 = i*4
174 addu t3,s0,t2 #t3 = trozos + i*4 es decir apunta a trozos[i]
175 lw t4,0(t3) #t4 traigo la palabra a la q apunta t3
176
177 addu t0,t0,t4 #t0 = t0 + f + e + k + trozos[i]
178
179 sw t0,52($fp) #guardo t0 en temp (52($fp))
180 #sw s4,0(s5) #e = d;
181 move s5,s4
182 #sw s3,0(s4) #d = c;
183 move s4,s3
184 rol s3,s2,30 #c = leftrotate(b ,30);
185 #sw s1,0(s2) #b = a;
186 move s2,s1
187 #sw t0,0(s1) #a = temp;
188 move s1,t0
189
190 #AUMENTO I
191 #addiu t1,t1,1 #i = i + 1
192 #sw t1,44($fp) #guardo
193

```

```

194     jalr    ra
195
196 return_algoritmo:
197 #-----FIN algoritmo
198
199     lw      a0, 80($fp)      # trozos
200     lw      a1, 84($fp)      # a
201     lw      a2, 88($fp)      # b
202     lw      a3, 92($fp)      # c
203     lw      t6, 96($fp)      # d
204     lw      t7, 100($fp)     # e
205
206     sw      s1, 0(a1)
207     sw      s2, 0(a2)
208     sw      s3, 0(a3)
209     sw      s4, 0(t6)
210     sw      s5, 0(t7)
211
212     # Restauro los valores los registros s0..s6.
213     lw      s0, 16(sp)
214     lw      s1, 20(sp)
215     lw      s2, 24(sp)
216     lw      s3, 28(sp)
217     lw      s4, 32(sp)
218     lw      s5, 36(sp)
219     lw      s6, 40(sp)
220
221     lw      $fp, 68(sp)
222     lw      gp, 64(sp)
223     lw      ra, 72(sp)
224     addu    sp, sp, 80
225
226     # Retorno.
227     j       ra
228     .end    algoritmoSha1
229
230     .data # comienza zona de datos
231
232     cteA: .word 0x67452301
233     cteB: .word 0xEFCDAB89
234     cteC: .word 0x98BADCFE
235     cteD: .word 0x10325476
236     cteE: .word 0xC3D2E1F0
237     ctek1: .word 0x5A827999;
238     ctek2: .word 0x6ED9EBA1;
239     ctek3: .word 0x8F1BBCDC;
240     ctek4: .word 0xCA62C1D6;

```

../src/algoritmo.S

5.5. Código Fuente Completo en language C

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define ERROR_OPEN_FILE 10
6 #define MAX_COUNT_FILES 50
7 #define HASH_LENGTH 20
8 #define MAX_FILE_SIZE 1000

```

```

9
10 void printHelp();
11 void printVersion();
12 void printError(char* msgError, int codeError);
13 void setFileSize(FILE* fp, long long int *length);
14 int readFromStdInput(int argumentCount);
15 int sha1(unsigned char *resultado, char *nombre_archivo, unsigned
    long long longitudOriginal);
16 unsigned int leftrotate(unsigned int valor, int desplazamiento);
17 void asignarDatos(FILE* fp, char *bloques, long long int
    tamanioOriginal, long long int longitudRelleno);
18 void showChecksum(unsigned char *result);
19
20 long long int calcularRelleno(long long int longitudOriginal);
21 void cargarTrozos(char *bloque, unsigned int *Trozos);
22 void algoritmoSha1(unsigned int *Trozos, unsigned *a, unsigned *b,
    unsigned *c, unsigned *d, unsigned *e);
23
24 int main(int argc, char* argv[]) {
25     int i = 0;
26     char* param;
27     char* files[MAX_COUNT_FILES];
28     unsigned char *result;
29     int fileCount = 0;
30     FILE* fp;
31     long long int length;
32
33     if (readFromStdInput(argc)) {
34         int c;
35
36         fp = fopen("archivoAuxiliar.txt", "w+");
37         while ((c = fgetc(stdin)) != EOF) {
38             fputc(c, fp);
39         };
40
41         setFileSize(fp, &length);
42         long long int longitudRelleno = calcularRelleno(length);
43         char *bloques = malloc(longitudRelleno/8);
44
45         result = malloc(HASH_LENGTH);
46         // se almacena el tamanioOriginal al final
47         asignarDatos(fp, bloques, length, longitudRelleno);
48         sha1(result, bloques, length);
49         showChecksum(result);
50
51         fclose (fp);
52         remove("archivoAuxiliar.txt");
53
54         return 0;
55
56     } else if (argc >= 2) { // Parse arguments
57         param = *(argv + 1);
58         if ((strcmp(param, "-h") == 0) || (strcmp(param, "--help"
            == 0) ) {
59             printHelp();
60         }
61         else if ((strcmp(param, "-V") == 0) || (strcmp(param, "--
            version") == 0)) {
62             printVersion();
63         }
64     }
65 }

```

```

66 // Search for files
67 for(i = 1; i < argc; i++) {
68     if (*argv[i] != '-') {
69         files[fileCount++] = argv[i];
70     }
71 }
72
73 // Process each file
74 for(i = 0; i < fileCount; i++) {
75     result = malloc(HASH_LENGTH);
76
77     if (NULL == (fp = fopen(files[i], "r"))) {
78         fprintf(stderr, "Files '%s' doesn't exist.", files[i]);
79         exit(ERROR_OPEN_FILE);
80     }
81
82     setFileSize(fp, &length);
83     long long int longitudRelleno = calcularRelleno(length);
84     char *bloques = malloc(longitudRelleno/8);
85
86     fp = fopen(files[i], "r");
87
88     // se almacena el tamañoOriginal al final
89     asignarDatos(fp, bloques, length, longitudRelleno);
90     sha1(result, bloques, length);
91     fclose(fp);
92 }
93
94 showChecksum(result);
95
96 return 0;
97 }
98
99 void setFileSize(FILE* fp, long long int *length) {
100     int character;
101     char *start;
102     int n = 0;
103
104     fseek(fp, 0, SEEK_END);
105     *length = ftell(fp);
106
107     fseek(fp, 0, SEEK_SET);
108
109     start = malloc(MAX_FILE_SIZE);
110
111     if ((*length) > MAX_FILE_SIZE) {
112         start = realloc(start, (*length));
113     }
114
115     *length *= 8; // devuelve el tamaño en bits
116
117     while ((character = fgetc(fp)) != EOF) {
118         start[n++] = (char)character;
119     }
120     start[n] = '\0';
121 }
122
123 void printHelp()
124 {
125     fprintf(stdout, "$ tp1 -h\n");
126     fprintf(stdout, "Usage:\n");
127     fprintf(stdout, "    tp1 -h\n");

```

```

128     fprintf(stdout, "    tp1 -V\n");
129     fprintf(stdout, "    tp1 [file ...]\n");
130     fprintf(stdout, "Options:\n");
131     fprintf(stdout, "    -V, --version    Print version and quit.\n");
132     ;
133     fprintf(stdout, "    -h, --help    Print this information and
quit.\n\n");
134     fprintf(stdout, "Examples:\n");
135     fprintf(stdout, "    tp1 foo\n");
136     fprintf(stdout, "    echo \"hello\" | tp1\n\n");
137 }
138 int sha1(unsigned char *resultado, char *bloques, unsigned long
long longitudOriginal)
139 {
140     long long int longitudRelleno = calcularRelleno(
longitudOriginal);
141     // Preprocesamiento
142
143     // longArchivo/tamano bloque    bloque = 512bits
144     long long int cantBloques = longitudRelleno/512;
145
146     // procesar el bloque en 4 rondas de 20 pasos cada ronda
147     // la memoria temporal cuenta con 5 registros ABCDE
148     unsigned A=0x67452301;
149     unsigned B=0xEFCDAB89;
150     unsigned C=0x98BADCFE;
151     unsigned D=0x10325476;
152     unsigned E=0xC3D2E1F0;
153
154     // big endian
155     unsigned int trozos[80];
156
157     int i;
158     unsigned a = 0;
159     unsigned b = 0;
160     unsigned c = 0;
161     unsigned d = 0;
162     unsigned e = 0;
163
164     while(cantBloques--)
165     {
166         cargarTrozos(bloques, trozos);
167
168         a = A;
169         b = B;
170         c = C;
171         d = D;
172         e = E;
173
174         algoritmoSha1(trozos, &a, &b, &c, &d, &e);
175
176         A += a;
177         B += b;
178         C += c;
179         D += d;
180         E += e;
181     }
182
183     // hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2
leftshift 64)
184     //         or (h3 leftshift 32) or h4

```



```

185     for(i = 0;i<4;i++)
186     {
187         resultado[i] = (A>>(24-8*i));
188         resultado[i+4] = (B>>(24-8*i));
189         resultado[i+8] = (C>>(24-8*i));
190         resultado[i+12] = (D>>(24-8*i));
191         resultado[i+16] = (E>>(24-8*i));
192     }
193
194     return 0;
195 }
196
197 // un bloque tiene 64 bytes
198 void asignarDatos(FILE *fp, char *bloques, long long int
199     tamanioOriginal, long long int longitudRelleno)
200 {
201     char caracter;
202     int indice =0;
203     int i =0;
204     int cantBitsRelleno = 0;
205     char relleno = 0x80; //minimo relleno
206     char rellenoCero = 0x00;
207     unsigned mascara = 0x00000000000000FF;
208
209     while( (caracter = getc(fp)) != EOF)
210     {
211         *(bloques+indice) = caracter;
212         indice++;
213     }
214
215     cantBitsRelleno = (longitudRelleno - tamanioOriginal);
216
217     // restamos 64 bits de tamanio y los 8 bits basicos de relleno;
218     cantBitsRelleno = cantBitsRelleno - 64 - 8;
219     *(bloques+indice) = relleno;
220     indice++;
221
222     for(i = 0;i<(cantBitsRelleno/8);i++)
223     {
224         *(bloques+indice) = rellenoCero;
225         indice++;
226     }
227
228     for(i = 0;i<8;i++)
229     {
230         *(bloques+indice) = (tamanioOriginal>>(56-8*i)) & mascara;
231         indice++;
232     }
233 }
234
235 // devuelve el tamanio en bits
236 long long int calcularTamanioArchivo(char* nombreFile)
237 {
238     long long int tamanio =0;
239     FILE *fp = fopen(nombreFile, "r");
240     fseek(fp,0,SEEK_END);
241     tamanio = ftell(fp)*8;
242     return tamanio;
243 }
244
245

```

```

246
247 unsigned int leftrotate(unsigned int valor,int desplazamiento)
248 {
249     desplazamiento %=32;
250     unsigned retorno;
251     retorno = (valor << desplazamiento) | (valor >> (32 -
        desplazamiento));
252     return retorno;
253 }
254
255 void printVersion()
256 {
257     fprintf(stdout, "Copyright (c) 2015\n");
258     fprintf(stdout, "Conjunto de instrucciones MIPS. v1.0.0\n\n");
259 }
260
261 void printError(char* msgError, int codeError)
262 {
263     fprintf(stderr, "%s\n", msgError);
264     exit(codeError);
265 }
266
267 void showChecksum(unsigned char *result) {
268     int i;
269     for(i = 0; i < 20; i++) {
270         unsigned char aux = result[i];
271         aux<<=4;
272         aux>>=4;
273         printf("%x", (result[i]>>4));
274         printf("%x",aux);
275     }
276     printf("\n");
277 }
278
279 int readFromStdInput(int argumentCount) {
280     return (int)(argumentCount < 2);
281 }
282
283 // Funciones en C, que fueron implementadas en Assembly.
284 void cargarTrozos(char *bloques, unsigned int *trozos)
285 {
286     int i;
287     unsigned mascara = 0x000000FF;
288     for (i = 0; i < 16; i++) {
289         trozos[i] = (*(bloques++) & mascara);
290         trozos[i]<=<8;
291         trozos[i]|= (*(bloques++) & mascara);
292         trozos[i]<=<8;
293         trozos[i]|= (*(bloques++) & mascara);
294         trozos[i]<=<8;
295         trozos[i]|= (*(bloques++) & mascara);
296     }
297
298     for (i = 16; i < 80; i++) {
299         trozos[i] = (trozos[i-3] ^ trozos[i-8] ^ trozos[i-14] ^
            trozos[i-16]);
300         trozos[i] = leftrotate(trozos[i], 1);
301     }
302 }
303
304
305 long long int calcularRelleno(long long int longitudOriginal)

```

```

306 {
307     unsigned long long int longitudRelleno;
308     longitudRelleno = longitudOriginal;
309
310     //incorporacion de bits de relleno 0..512 bits
311     while ((longitudRelleno % 512) != 0) {
312         longitudRelleno++;
313     }
314
315     // si hay al menos 65 bits agregamos 512 mas
316     if ((longitudRelleno - longitudOriginal) < 65)
317     {
318         longitudRelleno += 512;
319     }
320
321     return longitudRelleno;
322 }
323
324 void algoritmoSha1(unsigned int *trozos, unsigned *a, unsigned *b,
325                  unsigned *c, unsigned *d, unsigned *e)
326 {
327     int i;
328     unsigned int k;
329     unsigned int f;
330     unsigned int temp;
331
332     for (i = 0; i < 80; i++) {
333         if (i <= 19) {
334             f = (*b & *c) ^ ((~*b) & *d);
335             k = 0x5A827999;
336         } else if ((i >= 20) && (i <= 39)) {
337             f = *b ^ *c ^ *d;
338             k = 0x6ED9EBA1;
339         } else if ((i >= 40) && (i <= 59)) {
340             f = (*b & *c) | (*b & *d) | (*c & *d);
341             k = 0x8F1BBCDC;
342         } else if ((i >= 60) && (i <= 79)) {
343             f = *b ^ *c ^ *d;
344             k = 0xCA62C1D6;
345         }
346
347         temp = leftrotate(*a, 5);
348         temp = temp + f + *e + k + trozos[i];
349         *e = *d;
350         *d = *c;
351         *c = leftrotate(*b, 30);
352         *b = *a;
353         *a = temp;
354     }
355 }

```

../src/ansi/main.c

5.6. Bibliografía

Referencias

- [1] *US Secure Hash Algorithm 1 (SHA1)*
<http://tools.ietf.org/html/rfc3174>

- [2] *SHA1 Message Digest Algorithm Overview*
<http://www.herongyang.com/Cryptography/SHA1-Message-Digest-Algorithm-Overview.html>
- [3] *SHA 1*
<http://en.wikipedia.org/wiki/SHA-1>
- [4] *MIPS Assembly Language Guide*
http://www.cs.uni.edu/~fienup/cs041s08/lectures/lec20_MIPS.pdf
- [5] *MIPS Instruction Reference*
<http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>