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 archivos pasados por parámetro. En caso de no especificarse algún archivo, se mostrarán un checksum a partir de lo ingresado por *stdin*.

2. Diseño e Implementación

Se implementó 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 este 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.

2.1. Compilación

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

```
../src$ gcc -g -O0 -Wall main.c algoritmo.S trozo.S relleno.S -o sha
```

Para generar el ejecutable *sha*.

2.2. Arquitectura

En una primera versión se trabajo 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.

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

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

algoritmo.S

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

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

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

calcularRelleno	
Dir Mem	Valor
44	
40	ra
36	fp
32	gp
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
44	
40	ra
36	fp
32	gp
28	
24	temp
20	k
16	i
12	a3
8	a2
4	a1
0	a0

Registro	Valor
s0	bloques
s1	trozos
s2	a
s3	b
s4	c
s5	d
s6	e
s7	f

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 lenguage 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

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

#define ERROR_OPEN_FILE 10
#define MAX_COUNT_FILES 50
#define HASH_LENGTH 20

void printHelp();
void printVersion();
void printError(char* msgError, int codeError);
char* setFileSize(FILE* fp, long long int *length);

int readFromStdInput(int argumentCount);

long long int calcularRelleno(long long int longitudOriginal);
void cargarTrozos(char *bloque,unsigned int *Trozos);
void algoritmoSha1(unsigned int *Trozos,unsigned *a,unsigned *b,
    unsigned *c,unsigned *d,unsigned *e);

int sha1(unsigned char *resultado, char *nombre_archivo, unsigned
    long long longitudOriginal);
unsigned int leftrotate(unsigned int valor,int desplazamiento);
void asignarDatos(FILE* fp,unsigned char *bloques,long long int
    tamanioOriginal, long long int longitudRelleno);
void showCheckSum(unsigned char *result);

int main(int argc, char* argv[]) {
    int i = 0;
    char* param;
    char* files[MAX_COUNT_FILES];
    unsigned char *result;
    int fileCount = 0;
    FILE* fp;
    char *start;
    long long int length;

    if (readFromStdInput(argc)) {
        int c;

        fp = fopen("archivoAuxiliar.txt", "w+");
        while ((c = fgetc(stdin)) != EOF) {
            fputc(c, fp);
        };

        start = setFileSize(fp, &length);
        long long int longitudRelleno = calcularRelleno(length);
        char *bloques = malloc(longitudRelleno/8);
        asignarDatos(fp,bloques,length,longitudRelleno); // se
            almacena el tamanioOrginal al final
        sha1(result,bloques, length);
        showCheckSum(result);

        fclose (fp);
        remove("archivoAuxiliar.txt");

        return 0;

```

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

```
114 }
115
116 void printHelp()
117 {
118     fprintf(stdout, "$ tp1 -h\n");
119     fprintf(stdout, "Usage:\n");
120     fprintf(stdout, "  tp1 -h\n");
121     fprintf(stdout, "  tp1 -V\n");
122     fprintf(stdout, "  tp1 [file...]\n");
123     fprintf(stdout, "Options:\n");
124     fprintf(stdout, "  -V, --version    Print version and quit.\n")
        ;
125     fprintf(stdout, "  -h, --help       Print this information and
        quit.\n\n");
126     fprintf(stdout, "Examples:\n");
127     fprintf(stdout, "  tp1 foo\n");
128     fprintf(stdout, "  echo \"hello\" | tp1\n\n");
129 }
130
131 int sha1(unsigned char *resultado, char *bloques, unsigned long
    long longitudOriginal)
132 {
133     long long int longitudRelleno = calcularRelleno(
        longitudOriginal);
134     //prepocesamiento
135     long long int cantBloques = longitudRelleno/512; //longArchivo/
        tamanio bloque   bloque = 512bits
136
137     ////procesar el bloque en 4 rondas de 20 pasos cada ronda
138     //la memoria temporal cuenta con 5 regstros ABCDE
139     unsigned  A=0x67452301;
140     unsigned  B=0xEFCDAB89;
141     unsigned  C=0x98BADCFE;
142     unsigned  D=0x10325476;
143     unsigned  E=0xC3D2E1F0;
144
145     //unsigned int trozos[80]; //big endian
146     unsigned int *trozos = malloc(80*sizeof(int));
147
148     // int indice = 0;
149     int i;
150     unsigned a = 0;
151     unsigned b = 0;
152     unsigned c = 0;
153     unsigned d = 0;
154     unsigned e = 0;
155
156     while(cantBloques--)
157     {
158         cargarTrozos(bloques, trozos);
159
160         a = A;
161         b = B;
162         c = C;
163         d = D;
164         e = E;
165
166         algoritmoSha1(trozos,&a,&b,&c,&d,&e);
167
168         A += a;
169         B += b;
170         C += c;
```

```
171         D += d;
172         E += e;
173 
174     }
175 
176     // hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2
        leftshift 64) or (h3 leftshift 32) or h4
177     for(i = 0;i<4;i++)
178     {
179         resultado[i]    = (A>>(24-8*i));
180         resultado[i+4]  = (B>>(24-8*i));
181         resultado[i+8]  = (C>>(24-8*i));
182         resultado[i+12] = (D>>(24-8*i));
183         resultado[i+16] = (E>>(24-8*i));
184     }
185 
186     return 0;
187 }
188 
189 
190 //un bloque tiene 64 bytes
191 void asignarDatos(FILE *fp, unsigned char *bloques,long long int
        tamanioOriginal, long long int longitudRelleno)
192 {
193     // FILE *fp = fopen(file,"r");
194 
195     char caracter;
196     int indice =0;
197     int i =0;
198     int cantBitsRelleno = 0;
199     char relleno = 0x80; //minimo relleno
200     char rellenoCero = 0x00;
201     unsigned mascara = 0x00000000000000FF;
202 
203     while( (caracter = getc(fp)) != EOF)
204     {
205             *(bloques+indice) = caracter;
206             indice++;
207     }
208 
209     cantBitsRelleno = (longitudRelleno - tamanioOriginal);
210     cantBitsRelleno = cantBitsRelleno - 64 - 8;//restamos 64 bits
        de tamanio y los 8 bits basicos de relleno;
211     *(bloques+indice) = relleno;
212     indice++;
213 
214     for(i = 0;i<(cantBitsRelleno/8);i++)
215     {
216         *(bloques+indice) = rellenoCero;
217         indice++;
218     }
219 
220     for(i = 0;i<8;i++)
221     {
222         *(bloques+indice) = (tamanioOriginal>>(56-8*i)) & mascara;
223         indice++;
224     }
225 
226 }
227 
228 //devuleve el tamanio en bits
229 long long int calcularTamanioArchivo(char* nombreFile)
```

```
230 {
231     long long int tamanio =0;
232     FILE *fp = fopen(nombreFile,"r");
233     fseek(fp,0,SEEK_END);
234     tamanio = ftell(fp)*8;
235     return tamanio;
236 }
237
238
239 unsigned int leftrotate(unsigned int valor,int desplazamiento)
240 {
241     desplazamiento %=32;
242     unsigned retorno;
243     retorno = (valor << desplazamiento) |  (valor >> (32 -
        desplazamiento));
244     return retorno;
245 }
246
247 void printVersion()
248 {
249     fprintf(stdout, "Copyright (c) 2015\n");
250     fprintf(stdout, "Conjunto de instrucciones MIPS. v1.0.0\n\n");
251 }
252
253 void printError(char* msgError, int codeError)
254 {
255     fprintf(stderr, "%s\n", msgError);
256     exit(codeError);
257 }
258
259 void showCheckSum(unsigned char *result) {
260     int i;
261     for(i = 0; i < 20; i++) {
262         unsigned char aux = result[i];
263         aux<<=4;
264         aux>>=4;
265         printf("%x",(result[i]>>4));
266         printf("%x",aux);
267     }
268     printf("\n");
269 }
270
271 int readFromStdInput(int argumentCount) {
272     return (int)(argumentCount < 2);
273 }
```

../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    .cprestore 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($fp)
28
29    #obtencion del tamanio del relleno
30
31  #-----------------operaciones en 64 bits
        ----------------------------------------
32
33    li  t0,0
34    li  t1,0      #el tamanio debe venir en bits
35    move  t0,a0     #t1|t0 = longArchivoRelleno
36    move  t1,a1
37    li  t4,512
38
39  bucle_relleno:
40    remu  t2,t0,t4    #caculo el valor de la longitud en modulo 512 (
          resto)
41    beqz  t2,fin_bucle_relleno  #no es necesario considerar la parte
          mas significativa
42    #suma de los 64 bits
43    addiu t2,t0,8     #agrego 1 byte
44    sltu  t3,t2,t0    #(t3 =1) de carry si el resultado es mas chico
          que el sumando
45    move  t0,t2     #volvamos el valor a t0
46    beqz  t3,bucle_relleno
47    addiu t1,t1,1     #sumamos 1 al la parte mas significativa
48    b bucle_relleno   #solucionar la suma en 64 bits
49
50  fin_bucle_relleno:
51    slti  t2,t0,65    #si t0<65 entonces t2 =1 y agregamos 512 bits
52    beqz  t2,obtener_bloque
53    addiu t2,t0,512   #agrega un bloque de 512 extra
54    sltu  t3,t2,t0
55    move  t0,t2
56    beqz  t3,obtener_bloque #si no hay acarreo no sumar al mas
          significativo
57    addiu t1,t1,1     #t1|t0 = longArchivoRelleno
58
59  obtener_bloque:
60    sw  t0,16($fp)    #guardo la longitud relleno en 64 bits en el
          stack
61    sw  t1,20($fp)
62    sw  a0,24($fp)    #guardo la longitud original en 64 bits en el
          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  #s0 puntero a bloques
3  #s1 puntero a trozos
4  #
5
6
7  #include <mips/regdef.h>
8
9     .text
10    .align  2
11
12    .globl  cargarTrozos
13    .ent  cargarTrozos
14
15 cargarTrozos:
16    .frame $fp, 48, ra
17    .set noreorder
18    .cpload t9
19    .set reorder
20    subu  sp, sp, 40
21    .cprestore 0
22
23    sw  gp,24(sp)
24    sw  $fp,28(sp)
25    sw  ra,32(sp)
26    move  $fp, sp
27
28    #guardo los parametros con que vine #void cargarTrozos(char *
          bloques,int *indice,unsigned int *trozos)
29
30    sw  a0, 40($fp)   #ptero a bloques
31    sw  a1, 44($fp)   # ptero a trozos
32    sw  a2, 48($fp)   # nada
33    sw  a3, 52($fp) # nada
34
35    #cargo los parametros
36    move  s0,a0
37    move  s1,a1
38
39    sw  zero,16($fp)    # i=0
40    li  t0,255      # mascara 0x000000ff
41    sw  t0,20($fp)    #mascara en 28
42
43 FOR16PRIMEROS:
44    #traigo los datos
45    lw  t0,16($fp)    #t0 cargo i
```

```
46    lw  t5,20($fp)    #t5 = mascara
47
48    #comienzo
49
50    lbu t6,0(s0)    #t6 cargo el byte al que apunta (t4 = bloques +
          indice), lo carga de 0 a 7
51    and   t6,t6,t5    #t6 con el and de la mascara, queda 00 00 00
          ALGO
52    sll t6,t6,8     #lo muevo 8 lugares, quedaria 00 00 ALGO 00
53
54    addiu   s0,s0,1     #(*(bloques+indice+1)
55    lbu t7,0(s0)    #t7 cargo el byte al que apunta (t4 = bloques +
          indice + 1), lo carga de 0 a 7
56    and   t7,t7,t5    #t7 con el and de la mascara, queda 00 00 00
          ALGO
57    or  t6,t6,t7    #suma logica de t6 y t7, quedaria 00 00 ALGO ALGO
58    sll t6,t6,8     #muevo quedaria 00 ALGO ALGO 00
59
60    addiu   s0,s0,1     #(*(bloques+indice+2) (ya habia sumado 1
          antes)
61    lbu t7,0(s0)    #t7 cargo el byte al que apunta (t4 = bloques +
          indice + 2), lo carga de 0 a 7
62    and   t7,t7,t5    #t7 con el and de la mascara, queda 00 00 00
          ALGO
63    or  t6,t6,t7    #suma logica de t6 y t7, quedaria 00 ALGO ALGO
          ALGO
64    sll t6,t6,8     #muevo quedaria ALGO ALGO ALGO 00
65
66    addiu   s0,s0,1     #(*(bloques+indice+3) (ya habia sumado 1
          antes)
67    lbu t7,0(s0)    #t6 cargo el byte al que apunta (t4 = bloques +
          indice + 3), lo carga de 0 a 7
68    and   t7,t7,t5    #t6 con el and de la mascara, queda 00 00 00
          ALGO
69    or  t6,t6,t7    #suma logica de t6 y t7, quedaria ALGO ALGO ALGO
          ALGO
70
71    #GUARDO
72    sw  t6, 0(s1)   #guardo t6 en (t2 = trozos + i*4)
73
74    #AUMENTO VARIABLES
75
76
77    addiu   t0,t0,1     #i = i + 1
78    sw  t0,16($fp)    #guardo
79
80    addiu   s1,s1,4     # avanzo a la sig palabra
81    addiu   s0,s0,1     # avanzo al sig byte
82
83    addiu t7,t0,-16
84    beqz  t7,FOR16TO80
85
86    b FOR16PRIMEROS
87
88
89  FOR16TO80:
90    #traigo los datos
91    lw  t0,16($fp)    #t0 cargo i
92
93    addiu   s1,s1,-12   #t3 apunta a trozos(i - 3).. 3 lugares = 4
          bytesx3 = 12 bytes
94    lw  t4,0(s1)    #t4 traigo la palabra a la q apunta t3
```

```
95    addiu   s1,s1,12
96
97    addiu   s1,s1,-32   #t5 apunta a trozos(i - 8)
98    lw  t6,0(s1)    #t6 traigo la palabra a la que apunta t5
99    addiu   s1,s1,32
100
101   xor   t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8))
102
103   addiu   s1,s1,-56   #t5 apunta a trozos(i - 14)
104   lw  t6,0(s1)    #t6 traigo la palabra a la que apunta t5
105   addiu   s1,s1,56
106
107   xor   t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8)) ^
          *(trozos + (i-14))
108
109   addiu   s1,s1,-64   #t5 apunta a trozos(i - 16)
110   lw  t6,0(s1)    #t6 traigo la palabra a la que apunta t5
111   addiu   s1,s1,64
112
113   xor   t4,t4,t6    #t4 = *(trozos + (i-3)) ^ *(trozos + (i-8)) ^
          *(trozos + (i-14)) ^ *(trozos + (i-16)));
114
115   rol t4,t4,1     #leftrotate 1 Â¡?
116
117   sw  t4,0(s1)    #guardo t4 en t2 = trozos + i*4 es decir apunta a
           trozos[i]
118
119   addiu t0,t0,1     #i = i + 1
120   sw  t0,16($fp)    #guardo
121   addiu s1,s1,4     #t2 = avanzo a la siguiente palabra
122
123   addiu t7,t0,-80
124   beqz  t7,FinDelFor
125
126   b FOR16TO80
127
128 FinDelFor:
129
130   lw  $fp,28(sp)
131   lw  ra, 32(sp)
132   lw  gp, 24(sp)
133   addu  sp, sp, 40
134
135   # Retorno.
136   #
137   j ra
138   .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
11
12 #include <mips/regdef.h>
13
14    .text
15    .align   2
16
17    .globl   algoritmoSha1
18    .ent   algoritmoSha1
19
20 algoritmoSha1:
21    .frame $fp, 48, ra
22    .set noreorder
23    .cpload t9
24    .set reorder
25    subu   sp, sp, 48
26    .cprestore 0
27
28
29    sw   gp,32(sp)
30    sw   $fp,36(sp)
31    sw   ra,40(sp)
32    move   $fp, sp
33
34    sw   a0, 48($fp)      # trozos
35    sw   a1, 52($fp)      # a
36    sw   a2, 56($fp)      # b
37    sw   a3, 60($fp)    # c
38
39    la   t0,4(a3)     # dir
40    la   t1,8(a3)     # dir
41
42    sw   t0, 64($fp)    # d
43    sw   t1, 68($fp)    # e
44
45
46 #--------------------algoritmo-----------------------------------------
47
48 #
49 #s2 = a
50 #s3 = b
51 #s4 = c
52 #s5 = d
53 #s6 = e
54 #s7 = f (funcion)
55 #
56   #cargo variables en s0..s7
57
58    move   s1,a0
59    lw   s2,0(a1)
60    lw   s3,0(a2)
61    lw   s4,0(a3)
62    lw   s5,4(a3)
63    lw   s6,8(a3)
64
65   #inicializar i de vuelta
66    sw   zero,16($fp)     # i = 0
67    lw   t6,16($fp)     # traigo i, t6 = i
68
69 PROCESO0A19:
70    lw   t6,16($fp)     # traigo i, t6 = i
```

```
71    move  s7,zero
72
73    and s7,s3,s4     #en f(s7) <-- (b & c)
74    not t0,s3      #t0 niego b
75    and t1,t0,s5     #en t1 <-- ((~b) & d)
76    xor s7,s7,t1     #f(s7) <-- (b & c) ^ ((~b) & d)
77
78    lw  t2,ctek1     #cargo en t2, k1
79    sw  t2,20($fp)     #es donde esta k en el stack
80
81
82    jal ASIGNACIONTEMPORAL
83
84
85    addiu t6,t6,1
86    sw  t6,16($fp)
87
88    addiu t7,t6,-20
89    beqz  t7,PROCESO20A39
90
91    b PROCESO0A19
92
93
94  PROCESO20A39:
95    lw  t6,16($fp)     # traigo i, t6 = i
96    move  s7,zero
97
98    xor   s7,s3,s4     #f <-- b ^ c
99    xor   s7,s7,s5     #f <-- b ^ c ^ d
100
101   lw  t0,ctek2     #cargo en t0, k2
102   sw  t0,20($fp)     #es donde esta k en el stack
103
104
105   jal ASIGNACIONTEMPORAL
106
107
108   addiu t6,t6,1
109   sw  t6,16($fp)
110
111   addiu t7,t6,-40
112   beqz  t7,PROCESO40A59
113
114   b PROCESO20A39
115
116 PROCESO40A59:
117   lw  t6,16($fp)     # traigo i, t6 = i
118   move  s7,zero
119
120   and   s7,s3,s4     #en f(s7) <-- (b & c)
121   and   t0,s3,s5       #en t0 <-- (b & d)
122
123   or  s7,s7,t0     #f <-- (b & c) | (b & d)
124
125   and   t0,s4,s5       #en t0 <-- (c & d)
126
127   or  s7,s7,t0     #f <-- (b & c) | (b & d) | (c & d)
128
129   lw  t0,ctek3     #cargo en t0, k3
130   sw  t0,20($fp)     #es donde esta k en el stack
131
132
```

```
133     jal ASIGNACIONTEMPORAL
134
135
136     addiu t6,t6,1
137     sw  t6,16($fp)
138
139     addiu t7,t6,-60
140     beqz  t7,PROCESO60A79
141
142     b PROCESO40A59
143
144 PROCESO60A79:
145     lw  t6,16($fp)    # traigo i, t6 = i
146     move  s7,zero
147
148     xor   s7,s3,s4    #f <-- b ^ c
149     xor   s7,s7,s5    #f <-- b ^ c ^ d
150
151     lw  t0,ctek4    #cargo en t0, k4
152     sw  t0,20($fp)    #es donde esta k en el stack
153
154
155     jal ASIGNACIONTEMPORAL
156
157     addiu t6,t6,1
158     sw  t6,16($fp)
159
160     addiu t7,t6,-80
161     beqz  t7,return_algoritmo
162
163     b PROCESO60A79
164
165
166 ASIGNACIONTEMPORAL:
167
168     #ESTO FINALIZA EL FOR DE 80 CON EL ALGORITMO PER SE
169
170     rol   t0,s2,5     #en t0 leftrotate a 5
171     addu  t0,t0,s7    #t0 = t0 + f
172     addu  t0,t0,s6    #t0 = t0 + f + e
173     lw  t1,20($fp)    #t1 cargo k
174     addu  t0,t0,t1    #t0 = t0 + f + e + k
175
176     lw  t1,16($fp)    #t1 cargo i
177     sll t2,t1,2     #t2 = i*4
178     addu  t3,s1,t2    #t3 = trozos + i*4 es decir apunta a trozos[i]
179     lw  t4,0(t3)    #t4 traigo la palabra a la q apunta t3
180
181     addu  t0,t0,t4    #t0 = t0 + f + e + k + trozos[i]
182
183     sw  t0,24($fp)    #guardo t0 en temp (24($fp))
184     #sw s5,0(s6)    #e = d;
185     move s6,s5
186     #sw s4,0(s5)    #d = c;
187     move s5,s4
188     rol s4,s3,30    #c = leftrotate(b ,30);
189     #sw s2,0(s3)    #b = a;
190     move s3,s2
191     #sw t0,0(s2)    #a = temp;
192     move s2,t0
193
194     #AUMENTO I
```

```
195   #addiu   t1,t1,1      #i = i + 1
196   #sw    t1,16($fp)     #guardo
197
198   jalr   ra
199
200 return_algoritmo:
201 #----------------------FIN algoritmo
      ----------------------------------------
202
203
204   lw   a0, 48($fp)      # trozos
205   lw   a1, 52($fp)      # a
206   lw   a2, 56($fp)      # b
207   lw   a3, 60($fp)    # c
208   lw   t6, 64($fp)    # d
209   lw   t7, 68($fp)    # e
210
211   sw   s2,0(a1)
212   sw   s3,0(a2)
213   sw   s4,0(a3)
214   sw   s5,0(t6)
215   sw   s6,0(t7)
216
217
218   lw   $fp,36(sp)
219   lw   ra, 40(sp)
220   lw   gp, 32(sp)
221   addu   sp, sp, 48
222
223   # Retorno.
224   #
225   j ra
226   .end   algoritmoSha1
227
228
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