UBA - Facultad de Ingeniería

Departamento de Electrónica

Organización de Computadoras (66.20)

Trabajo Práctico 1

Programación MIPS

1er cuatrimestre - 2020

GRUPO Nº9		
Alumno	**Padrón**	**Mail**
BAUNI, Chiara	102981	cbauni@fi.uba.ar
MARTINEZ SASTRE, Gonzalo Gabriel	102321	gmartinezs@fi.uba.ar
RIEDEL, Nicolás Agustín	102130	nriedel@fi.uba.ar

Índice

1. Introducción

El principal objetivo de este trabajo es generar una mezcla entre código en C y en MIPS32 que procese un stream de vectores de números enteros y devuelva el mismo pero ordenado de manera creciente. La parte escrita en C deberá recibir un stream de vectores y procesar cada uno, en caso de error, el programa se detiene e informa el mismo. Mientras que el código escrito en MIPS32 ejecuta el algoritmo "merge sort", generando a medida que se van leyendo los vectores, que los mismos estén ordenados de manera creciente.

El código fuente se puede encontrar en el siguiente repositorio : https://github.com/RiedelNicolas/OrgaDeCompusTP1

Allí, además, se puede encontrar el archivo `merge.c` el cual contiene el código en C en el cual se basó el grupo para la elaboración del "merge sort."[en] MIPS32 de este trabajo práctico.

2. Documentación relevante al diseño e implementación del programa

Para implementar la solución al problema planteado, basamos el código de merge sort en MIPS32 según la descripción dada en el siguiente enlace: https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla#Optimizando_merge_sort. Además en la implementación en MIPS32 se hizo uso de la biblioteca mymalloc para la utilización de memoria dinámica y la liberación de la misma.

A la hora de procesar cada vector los consideramos como vectores de caracteres, si nos encontramos con algún carácter que no sea un numero entero, es decir, una letra, símbolo o linea vacía se interrumpe el programa informando el error producido.

Se tomaron una serie de decisiones sobre la implementación del programa o en casos donde el enunciado daba lugar a interpretación propia. Dichos casos son los siguientes :

- En el caso que durante el procesamiento de los vectores del stream de entrada se encuentre con un carácter considerado invalido se detendrá la ejecución del programa inmediatamente. Notificando por stderr lo ocurrido.
- Se considera invalido a cualquier carácter no numérico imprimible(excluyendo al espacio que se usa para separar los enteros).
- El programa ordena números negativos.
- Se considera el símbolo del signo negativo (carácter '-') y el símbolo del signo positivo (carácter '+') dentro del grupo de caracteres numéricos.
- Se considera que la utilización del símbolo '+' sirve únicamente para indicar el signo del número (no se considera como suma de números)
- En el caso de encontrarse con una linea vacía '\n '(sin ningún carácter imprimible, sea valido o no) se procede a finalizar la ejecución del programa .
- Para verificar que el llamado a la función mymalloc no falla, se utiliza syscall. Por lo que frente a una falla por parte de mymalloc se prosigue a finalizar con la ejecución del código(con sys_exit).

3. Comandos para compilar el programa

La línea de compilación de programa es la siguiente:

```
gcc -Wall -Werror -I. -o tp1 main.c ordenador.c merge_sort.S mymalloc.S
```

Donde los flags utilizados se describen a continuación:

- `-Wall`: muestra por pantalla todos los warnings que lanza el compilador.
- `-Werror`: trata a todos los warnings como errores.
- `-I.`: incluye todos los archivos del directorio actual.
- `-o nombre_del_ejecutable`: asigna un nombre al ejecutable generado.

4. Corridas de prueba

4.1. Codigo MIPS

A la hora de correr el código en assembly nos encontramos frente a varios errores, a continuación comentaremos algunos de ellos.

Restar 1 a la longitud : Cuando le restamos 1 a la longitud del vector, la función no funciona correctamente ya que queda un elemento fuera en cada iteración.

```
merge_sort:
    .frame fp,40,ra
    subu    sp,sp,40    #4(SRA) + 2(LTA) + 4(ABA)
    .cprestore 24       #Saved register area.
    sw      fp,28(sp)   #almaceno valor de fp a 28bytes del sp
    sw      ra,32(sp)
    move    fp,sp       #fp = sp

    sw      a0,40(fp)   #salvo *vec
    sw      a1,44(fp)   #salvo len

    addi    a2,zero,0   #establezco un valor para el inicio
    sub     a3,a1,1     #le resto 1 a la longitud del vector y
                        #establezco un valor para el final
```

Cargar basura en t8 : Al obtener del mymalloc memoria para generar el vector aux2[], en t6 tomo una posición i de esa parte de memoria en desuso. Pero en los espacios de memoria devueltos por mymalloc hay "basura"por lo que al cargar en t8 lo que obtenemos en t6, estaríamos cargando datos que desconocemos("basura"). Además en la linea "sw t7,0(t8)"no estamos haciendo lo deseado.

```
for_loop2:

        slt     t5,t4,t1        #si t4<tope1 entonces t5=1 sino t5=0
        beq     t5,0,pre_loop_2 #si t5==0 salto a pre_loop_2

        addu    t5,t4,1         #t5 = i + 1
        addu    t5,t5,a2        #t5 = medio + i + 1
        sll     t5,t5,2         #escalo t5
        addu    t6,a0,t5        #t6 = vec + (medio+i+1) escalado
        lw      t7,0(t6)        #t7 = vec[medio+i+1]    esto puede fallar

        sll     t5,t4,2
        addu    t6,t3,t5        #t7 = aux2 + i escalado
        lw      t8,0(t6)        #t8 = aux2[i]           esto puede fallar

        sw      t7,0(t8)        #aux2[i] = vec[medio+i+1]

        addu    t4,t4,1         #i++
        b       for_loop2
```

Figura 1: Extracto de código con error

Mala asignación de valor a t0 : Cuando le asignamos un valor a t0 lo primero que hacemos es sumarle 1 al inicio y luego le asignamos la resta entre medio e inicio+1. Pero si tenemos en cuenta el caso en el que el medio vale 0 al igual que el inicio si hacemos 0-(0+1)=-1, valor no aceptable. De esta manera primero deberíamos hacer medio-inicio y luego sumarle uno.

```
merge:
        .frame  fp,40,ra
        subu    sp,sp,40        #4(SRA) + 2(LTA) + 4(ABA)
        .cprestore 24           #Saved register area.
        sw      fp,28(sp)       #almaceno valor de fp a 28bytes del sp
        sw      ra,32(sp)
        move    fp,sp           #fp = sp

        sw      a0,40(fp)       #salvo *vec
        sw      a1,44(fp)       #salvo el valor del inicio
        sw      a2,48(fp)       #salvo el valor del medio
        sw      a3,52(fp)       #salvo el valor del final

        addi    t0,a1,1         #t0 = inicio + 1
        subu    t0,a2,t0        #t0 = medio - inicio + 1, t0 = tope1 (longitud del primer vector)
        sll     a0,t0,2         #a0 = tope1*4, (4 = sizeof(int))
        jal     mymalloc        #devuelve en la posicion v0 un puntero a una direccion con espacio para guardar el vector aux1
        addi    t2,v0,0         #t2 = aux1
```

Figura 2: Extracto de código con error

4.2. Código en C

Primeros pasos :

Al ser los 3 miembros del grupo estudiantes de Ingeniería Informática, por la experiencia adquirida en las materias previas, el código en C no presentó dificultades.

En primer instancia, para verificar el funcionamiento general del trabajo, se utilizó como método de ordenamiento un 'bubble sort' implementado en C. Una vez compilado y observando que el código no presentaba errores, se removió el mismo y se lo reemplazó por el 'merge sort' realizado en assembly.

Luego de compilado, se procedió a hacer algunas pruebas manuales, verificando si el output era el esperado, para luego pasar a las pruebas automatizadas.

Pruebas automatizadas :

Para las pruebas automatizadas se genero un ejecutable del tipo Bash (aprovechando el entorno UNIX). Dicho ejecutable realiza 3 pasos :

- Compila el programa, generando un ejecutable.
- Ejecuta el mismo.
- Corre una serie de pruebas sobre el ejecutable.

Si en alguno de los 3 pasos se encuentra con algo inesperado (el programa no compila o su funcionamiento no es apropiado) lo notifica por pantalla.

Por otro lado, las pruebas consisten en una serie de archivos de formato '.txt'. El script ejecuta el trabajo a testear (previo a compilarlo) y carga en stdin un archivo de texto, luego verifica si en stdout se encuentra la salida esperada (comparándola con otro de estos archivos de texto). De esta forma siempre que se hace algún cambio en el trabajo, solo basta con correr el script para verificar si el mismo sigue funcionando correctamente.

```
root@debian-stretch-mips:~/OrgaDeCompusTP1# ./pruebas.sh
Archivos en el directorio de corrida:
README.md               main.c        mymalloc     pruebas.sh
comandos                main.o        mymalloc.S   resources
ejemplo                 main.s        ordenador.c  salidas
lineaDeCompilacion.txt  merge.c       ordenador.h  tests
linea_compilacion.txt   merge_sort.S  ordenador.s  tp1
README.md               main.c        mymalloc     pruebas.sh
comandos                main.o        mymalloc.S   resources
ejemplo                 main.s        ordenador.c  salidas
lineaDeCompilacion.txt  merge.c       ordenador.h  tests
linea_compilacion.txt   merge_sort.S  ordenador.s  tp1

Corriendo compilacion...
gcc -o tp1 main.c ordenador.c merge_sort.S mymalloc.S -Wall -Werror -I. -g

El programa se pudo ejecutar correctamente

~~~~~~~~ Corriendo pruebas ~~~~~~~~~

~~~~~~~~ Corriendo test1.txt ~~~~~~~~~
Salida esperada: 1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 9 9 9
60 61 62 63 64 65
10 20 30 40 50 80
El programa compilo y se pudo ejecutar correctamente

~~~~~~~~ Corriendo test2.txt ~~~~~~~~~
Salida esperada: -1 1 2 3 4 5 6
-1 0 1
-1 1
-5 0 5
El programa compilo y se pudo ejecutar correctamente

~~~~~~~~ Corriendo test3.txt ~~~~~~~~~
Salida esperada: 1 4 5 6 6 7 8 9
2 3 4 5 7 8 9 10
10 11 12 13 46 48 49
El programa compilo y se pudo ejecutar correctamente

~~~~~~~~ Corriendo test4.txt ~~~~~~~~~
Salida esperada: -1 0 2 3 4 5
0 1 3 4 5 5 6 7 8 9
-1
El programa compilo y se pudo ejecutar correctamente

~~~~~~~~ Pasaste 4 pruebas de 4 ~~~~~~~~~
root@debian-stretch-mips:~/OrgaDeCompusTP1#
```

Figura 3: Ejemplo de corrida en un caso sin fallas.

5. Código fuente, en lenguaje C y MIPS

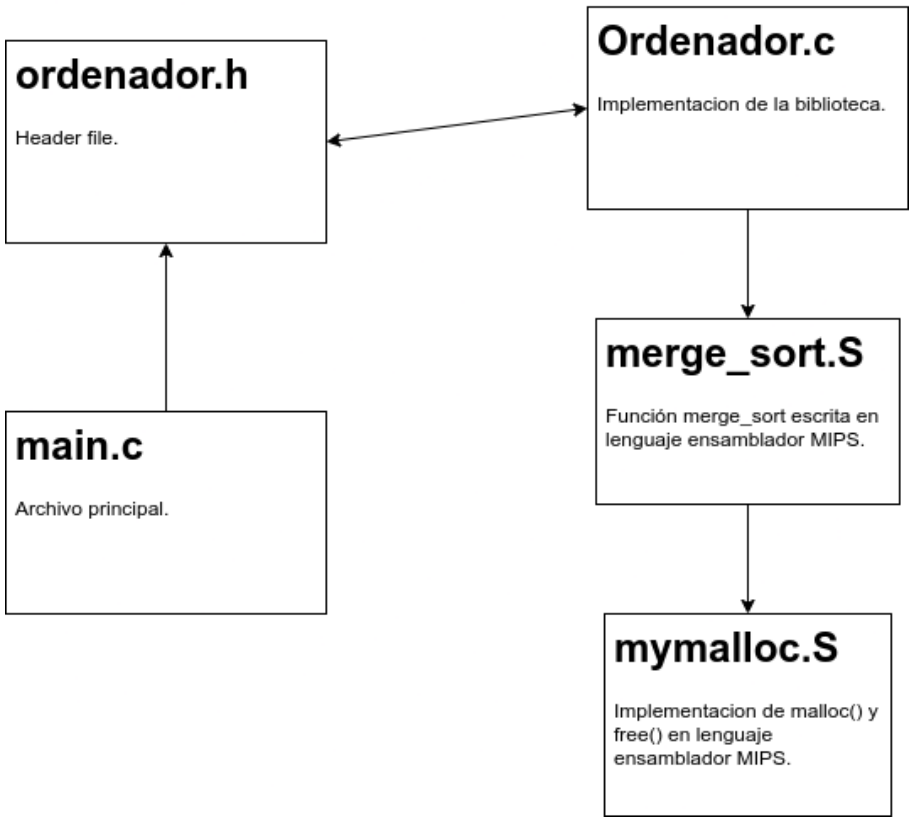

Figura 4: Diagrama de los archivos que componen el trabajo.

A continuación se presenta el código fuente de cada uno de los archivos del programa.

5.1. Archivo main.c

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

#include "ordenador.h"

#define MAX_MENSAJE 150
#define EOL '\n'


```

```
14
15 #define COMANDO_AYUDA_CORTO 'h'
16 #define COMANDO_VERSION_CORTO 'V'
17 #define COMANDO_AYUDA_LARGO "help"
18 #define COMANDO_VERSION_LARGO "version"
19
20 #define COMANDO_INPUT_CORTO 'i'
21 #define COMANDO_OUTPUT_CORTO 'o'
22 #define COMANDO_INPUT_LARGO "input"
23 #define COMANDO_OUTPUT_LARGO "output"
24
25
26 #define RUTA_AYUDA "comandos/help.txt"
27 #define RUTA_VERSION "comandos/version.txt"
28
29
30 #define CANTIDAD_ARGUMENTOS_FUNCIONAL 5
31 #define CANTIDAD_ARGUMENTOS_INFO 2
32
33
34
35 #define MENSAJE_COMANDO_INVALIDO "\nEl comando usado es invalido, use -h para ayuda
       "
36 #define MENSAJE_ORDENAMIENTO_FALLO "\n No se pudo ordenar, el input no tiene el
       formato requerido"
37
38
39
40
41 int mostrar_en_pantalla(char* ruta);
42 void notificar_problema_ruta(char *ruta);
43
44
45 int main(int argc, char** argv){
46
47   FILE* stream_entrada = NULL;
48   FILE* stream_salida = NULL;
49   int flag_ordenamiento = -1;
50   int long_index = 0;
51   int opt = 0;
52
53   static struct option long_options[] = {
54         {COMANDO_VERSION_LARGO,      no_argument,       NULL, COMANDO_VERSION_CORTO
       },
```

```
55      {COMANDO_AYUDA_LARGO,         no_argument,       NULL,  COMANDO_AYUDA_CORTO
    },
56      {COMANDO_INPUT_LARGO,    required_argument,     NULL,  COMANDO_INPUT_CORTO
  },
57      {COMANDO_OUTPUT_LARGO,   required_argument,     NULL,  COMANDO_OUTPUT_CORTO
    },
58      {NULL,           0,                     NULL,    0 }
59  };
60
61
62  while ( (opt = getopt_long(argc, argv,"Vhi:o:",
63                  long_options, &long_index )) != -1) {
64      switch (opt) {
65          case COMANDO_VERSION_CORTO :
66            return mostrar_en_pantalla(RUTA_VERSION);
67     case COMANDO_AYUDA_CORTO :
68            return mostrar_en_pantalla(RUTA_AYUDA);
69          case COMANDO_INPUT_CORTO :
70      stream_entrada = fopen(optarg, "r");
71      if(stream_entrada == NULL){
72        notificar_problema_ruta(optarg);
73        if(stream_salida !=NULL ){ // se pudo abrir el otro archivo.
74          fclose(stream_salida);
75        }
76        return FALLO;
77      }
78            break;
79          case COMANDO_OUTPUT_CORTO :
80      stream_salida = fopen(optarg, "w");
81      if(stream_salida == NULL){
82        notificar_problema_ruta(optarg);
83        if(stream_entrada != NULL ){ // se pudo abrir el otro archivo.
84          fclose(stream_entrada);
85        }
86        return FALLO;
87      }
88              break;
89          default:
90            perror(MENSAJE_COMANDO_INVALIDO);
91            return FALLO;
92      }
93  }
94
95  // En el caso que se tenga que activar el default.
96  if(stream_entrada == NULL ) stream_entrada = stdin;
```

```
97      if(stream_salida == NULL ) stream_salida = stdout;
98
99    flag_ordenamiento = ordenar(stream_entrada, stream_salida);
100
101   if(stream_salida != stdout) fclose(stream_salida);
102   if(stream_entrada != stdin ) fclose(stream_entrada);
103
104   if(flag_ordenamiento == FALLO){
105     perror(MENSAJE_ORDENAMIENTO_FALLO);
106     return FALLO;
107   }
108
109   return EXITO;
110 }
111
112 /*Pre: Recibe una ruta a un archivo en formato de string.
113   Pos: Muestra por stdin dicho archivo (similar al comando unix "cat".
114 */
115 int mostrar_en_pantalla(char * ruta){
116   FILE* archivo = fopen(ruta,"r");
117   if (archivo == NULL){
118     perror("\n no se pudo abrir el archivo \n");
119     return FALLO;
120   }
121
122   int caracter  = 1;
123
124   while(caracter != EOF){
125     caracter = fgetc(archivo);
126     if(caracter !=EOF) putc(caracter, stdout);
127   }
128
129   fclose(archivo);
130   return FALLO;
131 }
132
133 /*Pre: Recibe una ruta a un archivo en formato de string.
134   Pos: Notifica por stderror que se tuvo un error con un archivo de dicha ruta.
135 */
136 void notificar_problema_ruta(char *ruta){
137   char mensaje [MAX_MENSAJE];
138   strcpy(mensaje,"\nEl archivo en la ruta: ");
139   strcat(mensaje,ruta);
140   strcat(mensaje,". No se pudo abrir correctamente\0"); //esto lo hago porque
      perror no recibe parametros.
```

```
141   perror(mensaje);
142 }
```

5.2. Archivo merge_sort.S

```
1
2  #include <sys/regdef.h>
3  #include <sys/syscall.h>
4
5      .abicalls
6      .text
7      .align  2
8      .globl merge_sort
9      .ent merge_sort
10
11 merge_sort:
12   .frame fp,40,ra
13   subu    sp,sp,40  #4(SRA) + 2(LTA) + 4(ABA)
14   .cprestore 24   #Saved register area.
15   sw      fp,28(sp) #almaceno valor de fp a 28bytes del sp
16   sw      ra,32(sp)
17   move    fp,sp   #fp = sp
18
19   sub     t0,a1,1     #t0 = len - 1
20   move  a2,t0
21   move  a1,zero
22
23   jal merge_sort_rec
24
25   lw      ra,32(sp)
26   lw      gp,24(sp)
27   lw      fp,28(sp)
28   addu    sp,sp,40
29   jr      ra
30
31   .end  merge_sort
32
33
34   .ent merge_sort_rec
35
36 merge_sort_rec:
37   .frame  fp,40,ra
38   subu    sp,sp,40  #4(SRA) + 2(LTA) + 4(ABA)
39   .cprestore 24   #Saved register area.
40   sw      fp,28(sp) #almaceno valor de fp a 28bytes del sp
```

```
41    sw      ra,32(sp)
42    move    fp,sp   #fp = sp
43
44
45    sw      a0,40(fp) #salvo vec en ABA de caller
46    sw      a1,44(fp) #salvo inicio
47    sw      a2,48(fp) #salvo fin
48
49    slt     t0,a1,a2  #si a1<a2 entonces t0=1 sino t0=0
50    beq     t0,0,return #si t0=0 salto a return
51
52    add     t0,a1,a2  #t0 = inicio + final
53    subu    t0,t0,1     #t0 = inicio + final - 1
54    sra     t0,t0,1   #t0 = (inicio + final - 1)/2, t0=medio
55    sw      t0,16(fp) #como medio es propio de cada scope se almacena en la LTA
56    addi    a2,t0,0   #establezco el medio como nuevo fin
57    jal     merge_sort_rec  #tomo la primera mitad del vector y aplico merge sort
58
59
60    lw      a0,40(fp) #cargo en a0 el valor de vec
61    lw      a2,48(fp) #cargo en a2 el valor de fin
62    lw      t0,16(fp) #cargo medio en t0
63    addi    t0,t0,1   #sumo t0=t0+1
64    addi    a1,t0,0   #establezco medio+1 como el nuevo inicio
65    jal     merge_sort_rec  #tomo la segunda mitad del vector y aplico merge sort
66
67    lw      a0,40(fp) #cargo en a0 el valor de vec
68    lw      a1,44(fp) #cargo en a1 el valor del inicio
69    lw      a2,16(fp) #cargo en a2 el valor del medio
70    lw      a3,48(fp) #cargo en a3 el valor del final
71    jal     merge
72
73  return:
74    lw      ra,32(sp)
75    lw      gp,24(sp)
76    lw      fp,28(sp)
77    addu    sp,sp,40
78    jr      ra
79
80    .end  merge_sort_rec
81
82
83    .ent merge
84
85  merge:
```

```
86  .frame  fp,48,ra
87  subu    sp,sp,48  #4(SRA) + 4(LTA) + 4(ABA)
88  .cprestore 32   #Saved register area.
89  sw      fp,36(sp) #almaceno valor de fp a 28bytes del sp
90  sw      ra,40(sp)
91  move    fp,sp   #fp = sp
92
93  sw      a0,48(fp) #salvo vec en ABA de caller
94  sw      a1,52(fp) #salvo el valor del inicio
95  sw      a2,56(fp) #salvo el valor del medio
96  sw      a3,60(fp) #salvo el valor del final
97
98  subu    t0,a2,a1  #t0 = medio - inicio
99  addi    t0,t0,1   #t0 = medio - inicio + 1 , t0 = tope1 (longitud del primer
      vector)
100 sw      t0,16(fp) #guardo tope1 en LTA (16(fp))
101 sll     a0,t0,2   #a0 = tope1*4, (4 = sizeof(int))
102 jal     mymalloc  #v0 tiene la direc del primer elemento de la region solicitada
      (aux1)
103 beq     v0,0,kill_exec1   #si el puntero recibido es NULL,cortar ejecucion
104 sw      v0,24(fp)   #guardo aux1 en LTA (24(fp))
105
106 lw      a2,56(fp)
107 lw      a3,60(fp)
108 subu    t1,a3,a2  #t1 = fin - medio, t1 = tope2 (longitud del segundo vector)
109 sw      t1,20(fp) #guardo tope2 en LTA (20(fp))
110 sll     a0,t1,2   #a0 = tope2*4
111 jal     mymalloc  #v0 tiene la direc del segundo elemento de la region solicitada
      (aux2)
112 beq     v0,0,kill_exec2   #si el puntero recibido es NULL,cortar ejecucion
113 sw      v0,28(fp) #guardo aux2 en LTA (28(fp))
114
115
116 lw      a0,48(fp) #a0 = vec
117 lw      a1,52(fp) #a1 = inicio
118 lw      a2,56(fp) #a2 = medio
119 lw      a3,60(fp) #a3 = fin
120
121 lw      t0,16(fp) #t0 = tope1
122 lw      t1,20(fp) #t1 = tope2
123 lw      t2,24(fp) #t2 = aux1
124 move    t3,v0   #t3 = aux2
125
126 move    t4,zero   #t4 = 0, t4 = i
127
```

```
128
129 for_loop:
130
131   slt     t5,t4,t0  #si i<tope1 (t4<t0) entonces t5=1 sino t5=0
132   beq     t5,0,pre_loop   #si t5==0 salto a pre_loop
133
134   addu    t5,a1,t4  #t5 = inicio + i
135   sll     t5,t5,2   #escalo t5
136   addu    t6,a0,t5  #t6 = vec + (inicio+i) escalado
137   lw      t7,0(t6)  #t7 = vec[inicio+i]
138
139   sll     t5,t4,2
140   addu    t6,t2,t5  #t7 = aux1 + (i escalado)
141
142   sw      t7,0(t6)  #aux1[i] = vec[inicio+i]
143
144   addu    t4,t4,1   #i++
145   b       for_loop
146
147 pre_loop:
148   move    t4,zero   #t4 = 0, t4 = i
149
150 for_loop2:
151
152   slt     t5,t4,t1  #si i<tope2 (t4<t1) entonces t5=1 sino t5=0
153   beq     t5,0,pre_loop_2 #si t5==0 salto a pre_loop_2
154
155   addu    t5,t4,1   #t5 = i + 1
156   addu    t5,t5,a2  #t5 = medio + i + 1
157   sll     t5,t5,2   #escalo t5
158   addu    t6,a0,t5  #t6 = vec + (medio+i+1) escalado
159   lw      t7,0(t6)  #t7 = vec[medio+i+1]
160
161   sll     t5,t4,2
162   addu    t6,t3,t5  #t7 = aux2 + (i escalado)
163
164   sw      t7,0(t6)  #aux2[i] = vec[medio+i+1]
165
166   addu    t4,t4,1   #i++
167   b       for_loop2
168
169 pre_loop_2:
170   sw      t0,16(fp) #guardo tope1 en 16(fp)
171   sw      t1,20(fp) #guardo tope2 en 20(fp)
172   move    t4,zero   #t4 = 0, t4 = i
```

```
173   move    t5,zero   #t5 = 0, t5 = j
174   move    t6,a1     #t6 = inicio, t6 = k
175
176 while1:
177   lw      t0,16(fp)
178   lw      t1,20(fp)
179
180   ble     t0,t4,pre_loop_3    #si tope1 <= i (t0<=t4) salto a pre_loop_3
181   ble     t1,t5,pre_loop_3    #si tope2 <= j (t1<=t5) salto a pre_loop_3
182
183 if:
184   sll     t0,t4,2   #escalo i: t0 = i*4
185   addu    t0,t2,t0  #t0 = direccion de aux1[i]
186   lw      t1,0(t0)  #cargo el valor de aux1[i] en t1
187
188   sll     t0,t5,2   #escalo j: t0 = j*4
189   addu    t0,t3,t0  #t0 = direccion de aux2[j]
190   lw      t7,0(t0)  #cargo el valor de aux2[j] en t7
191
192   sll     t0,t6,2   #escalo k: t0 = k*4
193   addu    t0,a0,t0  #t0 = direccion de vec[k]
194
195   blt     t7,t1,else  #si aux2[j]<aux1[i] (t7<t1) salto a else
196
197   sw      t1,0(t0)  #vec[k]=aux1[i]
198   addi    t4,t4,1   #i++
199   addi    t6,t6,1   #k++
200   b       while1
201
202 else:
203   sw      t7,0(t0)  #vec[k]=aux2[j]
204   addi    t5,t5,1   #j++
205   addi    t6,t6,1   #k++
206   b       while1
207
208 pre_loop_3:
209   lw      t0,16(fp)
210
211 while2:
212   ble     t0,t4,pre_loop_4    #si tope1<=i (t0<=t4) salto a pre_loop_4
213
214   sll     t1,t4,2     #escalo i: t1 = i*4
215   addu    t1,t2,t1  #t1 = direccion de aux1[i]
216   lw      t7,0(t1)  #cargo el valor de aux1[i] en t7
217
```

```
218   sll     t1,t6,2   #escalo k: t1 = k*4
219   addu    t1,a0,t1  #t1 = direccion de vec[k]
220   sw      t7,0(t1)  #vec[k]=aux1[i]
221
222   addi    t4,t4,1   #i++
223   addi    t6,t6,1   #k++
224   b       while2
225
226 pre_loop_4:
227   lw      t1,20(fp)
228
229 while3:
230   ble     t1,t5,_return   #si tope2<=j (t1<=t5) salto a _return
231
232   sll     t0,t5,2   #escalo j: t0 = j*4
233   addu    t0,t3,t0  #t0 = direccion de aux2[j]
234   lw      t7,0(t0)  #cargo el valor de aux2[j] en t7, puede fallar
235
236   sll     t0,t6,2   #escalo k: t0 = k*4
237   addu    t0,a0,t0  #t0 = direccion de vec[k]
238   sw      t7,0(t0)  #vec[k]=aux1[i]
239
240   addi    t5,t5,1   #j++
241   addi    t6,t6,1   #k++
242   b       while3
243
244 _return:
245   lw      a0,24(fp) #a0 = aux1
246   jal     myfree    #libero memoria de aux1[]
247   lw      a0,28(fp) #a0 = aux2
248   jal     myfree    #libero memoria de aux2[]
249   lw      ra,40(sp)
250
251   lw      gp,32(sp)
252   lw      fp,36(sp)
253   addu    sp,sp,48
254   jr      ra
255
256
257 kill_exec1:
258     li      v0,SYS_exit
259     syscall
260
261 kill_exec2:
262     lw      a0,24(fp)   #a0 = aux1
```

```
263     jal     myfree      #libero memoria de aux1[]
264     li      v0,SYS_exit
265     syscall
266
267   .end  merge
```

5.3. Archivo ordenador.h

```
1  #define EXITO  0
2  #define FALLO -1
3
4
5  /*Pre: Recibe un stream correctamente abierto en modo de lectura
6      y otro en modo escritura.
7   *Pos: En el caso que el stream de entrada tenga formato de lineas consecutivas
8      de numeros enteros separados por espacios. Ordena cada linea en modo ascendente
9      y dicho resultado lo imprime en el stream de salida. Devolviendo
10     el flag "EXITO". En caso que no se respete el formato se devuelve "FALLO"
11 */
12 int ordenar(FILE* entrada, FILE* salida);
13
14
15 #endif
```

5.4. Archivo ordenador.c

```
1  #include "ordenador.h"
2
3  #define MAX_PATH 150
4  #define EOL '\n'
5  #define MAX_DIGITOS 20
6  #define FLAG_FIN_DE_ARCHIVO 1
7  #define FLAG_LINEA_INVALIDA -1
8  #define FLAG_CONTINUAR 0
9
10
11
12 int ordenar(FILE* entrada, FILE* salida);
13 int leer (FILE* stream, int *largo_linea, char** linea);
14 int* pasar_a_enteros(char* linea, int largo_linea, size_t* largo_enteros);
15 bool es_fin_de_linea(char caracter);
16 bool es_numerico(char caracter);
17 extern void merge_sort(int *vec, size_t len);
18 void imprimir_enteros(int *enteros, size_t largo, FILE* salida);
19 bool es_caracter_invalido(char caracter);
```

```
20
21
22
23
24 /*Pre: Recibe un stream correctamente abierto en modo de lectura
25     y otro en modo escritura.
26  *Pos: En el caso que el stream de entrada tenga formato de lineas consecutivas
27     de numeros enteros separados por espacios. Ordena cada linea en modo ascendente
28     y dicho resultado lo imprime en el stream de salida. Devolviendo el flag "EXITO
       "
29     En caso que no se respete el formato se devuelve "FALLO"
30 */
31
32 int ordenar(FILE* entrada, FILE* salida){
33
34   int flag_lectura = FLAG_CONTINUAR ;
35   int *enteros = NULL;
36   size_t largo_enteros = 0;
37   int largo_linea = 0;
38   char* linea = NULL;
39
40   while(flag_lectura == FLAG_CONTINUAR){
41
42     flag_lectura = leer(entrada, &largo_linea, &linea);
43
44     if(flag_lectura != FLAG_LINEA_INVALIDA ){
45
46       enteros = pasar_a_enteros(linea, largo_linea, &largo_enteros);
47       merge_sort(enteros, largo_enteros);
48       imprimir_enteros(enteros, largo_enteros, salida);
49       free(enteros);
50     }
51
52     free(linea);
53     }
54
55     if(flag_lectura == FLAG_LINEA_INVALIDA) return FALLO;
56
57     return EXITO;
58 }
59
60
61 /*POS: Recibe un file stream correctamente abierto en modo lectura.
62   Pre: Lee una linea de dicho stream, devolviendo por parametros la misma
63       en forma de array de caracteres y el largo de la linea. En forma de retorno
```

```
64        devuelve un flag indicando el resultado de la lectura :
65        FLAG_CONTINUAR:      En caso de que el archivo continue.
66        FLAG_FIN_DE_ARCHIVO: En caso de encontrase con un EOF o una linea
67                que solo contenga -1(indicador para dejar de iterar).
68      FLAG_LINEA_INVALIDA: En que se haya tenido que detener la ejecucion debido a un
69                caracter invalido encontrado en el stream.
70  */
71  int leer(FILE* stream, int *largo_linea, char** linea){
72
73    int largo_buffer = 20;
74    *linea = (char*) malloc(sizeof(char) * largo_buffer); // Asigno un lugar en
        memoria para el linea.
75    (*largo_linea) = 0;
76    int caracter = 1; // un valor trivial
77
78    while ( (caracter != EOL ) && (caracter != EOF) ) {
79
80        if( (*largo_linea) == (largo_buffer-1) ){ // tengo que agrandar mi memoria.(
        Dejo lugar para /0)
81            largo_buffer +=10; //Voy agregando de a 10 lugares.
82              (*linea) = (char*) realloc((*linea), sizeof(char) * largo_buffer); // re
        ubico en la memoria.
83        }
84
85        caracter = getc(stream); // Leo un caracter del stream.
86        if( es_caracter_invalido(caracter) ){
87           return FLAG_LINEA_INVALIDA; // Si lee un caracter que no corresponde,
        devuelve linea invalida.
88        }
89        (*linea) [ (*largo_linea) ] = (char) caracter; //Lo guardo en el linea.
90        (*largo_linea)+=1; //Incremento mi tope.
91    }
92
93    if(caracter == EOF || (*largo_linea) <=1){// Siempre va a leer por lo menos un
        caracter, sea eof o fin de linea
94      return FLAG_FIN_DE_ARCHIVO;
95    }
96
97    return  FLAG_CONTINUAR;
98
99  }
100
101 /*Pre: Recibe un array de caracteres que contiene numeros enteros
102       separados por un espacio(Esto es previamente validado) junto con su largo.
```

```
103   Pos: Devuelve en forma de retorno el puntero a un array de enteros equivalente al
          de caracteres
104         Y por parametro devuelve su largo.
105 */
106 int* pasar_a_enteros(char* linea, int largo_linea, size_t* largo_enteros){
107
108   char temporal [MAX_DIGITOS];
109   char caracter = 'A';
110   int largo_buffer = 10;
111   int *enteros = (int*) malloc(sizeof(int) * largo_buffer);
112
113   (*largo_enteros) = 0;
114
115   int i = 0;
116   int j = 0;
117
118   while( i<largo_linea ){
119
120     if( (*largo_enteros) == (largo_buffer-1) ){
121       largo_buffer +=10; //Voy agregando de a 10 lugares.
122           enteros = (int*) realloc(enteros, sizeof(int)* largo_buffer); // re ubico
       en la memoria.
123     }
124
125     caracter = linea[i]; i++;
126     if( es_numerico(caracter) ){
127       temporal[j] = caracter; j++;
128     }
129     else if( (caracter =' ' || es_fin_de_linea(caracter) ) && j!=0 ){
130       temporal[j] = '\0';
131       enteros[(*largo_enteros)] = atoi(temporal);
132       (*largo_enteros) += 1;
133       j = 0;
134     }
135
136   }
137
138   return enteros;
139 }
140
141 /*Pre: Recibe un entero que representa un..
142   Pos: Devuelve TRUE si se encuentra un caracter que indice el fin de linea
143 */
144 bool es_fin_de_linea(char caracter){
145
```

```
146   return(caracter == EOL || caracter == (char) EOF);
147 }
148
149
150 /*Pre: Recibe un caracter.
151   Pos: Responde a la pregunta es numerico?(se considera los signos de + y - como
        numericos)
152 */
153 bool es_numerico(char caracter){
154   return( (caracter >= '0' && caracter <= '9') || caracter=='-'||caracter=='+');
155 }
156
157
158 /*Pre: Recibe un caracter
159    Pos: Si el caracter es numerico, EOF, EOL o espacio devuelve TRUE.
160       en otro caso devuelve FALSE. (dicho caracter no se deberia encontrar en el
        input)
161 */
162 bool es_caracter_invalido(char caracter){
163   return !(es_numerico(caracter) || es_fin_de_linea(caracter) || caracter ==' ');
164 }
165
166
167 /*
168 Pre : Recibe un array de enteros, su largo y un stream de salida.
169 Pos:  "Imprime" dicho array en el stream.
170 */
171 void imprimir_enteros(int *enteros, size_t largo, FILE* salida){
172
173   if(largo == 0 ) return;
174
175   for (int i = 0; i < largo; i++){
176     fprintf(salida, "%i ",enteros[i] );
177   }
178
179   fprintf(salida,"\n");
180 }
```

6. Código MIPS32 generado por el compilador

Por cuestiones practicas solo mostramos las primeras 100 lineas, el resto aparece en el repositorio adjunto.

```
7f45 4c46 0102 0100 0000 0000 0000 0000
0003 0008 0000 0001 0000 0ac0 0000 0034
0000 531c 7000 1007 0034 0020 000a 0028
0028 0027 0000 0006 0000 0034 0000 0034
0000 0034 0000 0140 0000 0140 0000 0005
0000 0004 0000 0003 0000 0174 0000 0174
0000 0174 0000 000d 0000 000d 0000 0004
0000 0001 7000 0003 0000 01a8 0000 01a8
0000 01a8 0000 0018 0000 0018 0000 0004
0000 0008 7000 0000 0000 01c0 0000 01c0
0000 01c0 0000 0018 0000 0018 0000 0004
0000 0004 0000 0001 0000 0000 0000 0000
0000 0000 0000 2584 0000 2584 0000 0005
0001 0000 0000 0001 0000 2584 0001 2584
0001 2584 0000 010c 0000 011c 0000 0006
0001 0000 0000 0002 0000 01fc 0000 01fc
0000 01fc 0000 0118 0000 0118 0000 0004
0000 0004 0000 0004 0000 0184 0000 0184
0000 0184 0000 0020 0000 0020 0000 0004
0000 0004 0000 0004 0000 01d8 0000 01d8
0000 01d8 0000 0024 0000 0024 0000 0004
0000 0004 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0004 2f6c 6962 2f6c 642e 736f 2e31
0000 0000 0000 0004 0000 0010 0000 0001
474e 5500 0000 0000 0000 0003 0000 0002
0000 0000 0000 0000 0000 2002 0101 0005
0000 0000 0000 0000 0000 0000 0000 0000
b200 01f6 0000 0000 0000 0000 0000 0000
0000 0000 0001 a5a0 0000 0004 0000 0014
0000 0003 474e 5500 5ac3 a662 bbe0 975e
37bc b62f 8fed 30bd 2569 03ff 0000 0001
0000 0213 0000 000c 0000 0a78 0000 000d
0000 23e0 0000 0019 0001 2584 0000 001b
0000 0004 0000 001a 0001 2588 0000 001c
0000 0004 0000 0004 0000 0314 0000 0005
0000 0784 0000 0006 0000 0474 0000 000a
0000 0240 0000 000b 0000 0010 7000 0035
0001 2344 0000 0015 0000 0000 0000 0003
0001 25b0 0000 0011 0000 0a58 0000 0012
```

```
0000 0020 0000 0013 0000 0008 7000 0001
0000 0001 7000 0005 0000 0002 7000 0006
0000 0000 7000 000a 0000 001f 7000 0011
0000 0031 7000 0012 0000 0025 7000 0013
0000 0019 6fff fffb 0800 0000 6fff fffe
0000 0a28 6fff ffff 0000 0001 6fff fff0
0000 09c4 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0025 0000 0031 0000 0014
0000 001c 0000 002b 0000 000d 0000 0013
0000 0000 0000 0000 0000 0002 0000 0024
0000 0003 0000 0016 0000 0009 0000 000f
0000 0000 0000 0022 0000 001b 0000 0000
0000 000c 0000 0018 0000 0017 0000 0000
0000 000b 0000 001a 0000 0025 0000 0004
0000 0000 0000 0008 0000 0019 0000 0020
0000 001e 0000 002f 0000 0011 0000 001d
0000 0021 0000 0007 0000 0027 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 002a 0000 0000
0000 0000 0000 0000 0000 0006 0000 0000
0000 002e 0000 0000 0000 0000 0000 0000
0000 0030 0000 001f 0000 0005 0000 0012
0000 0000 0000 0000 0000 000e 0000 0015
0000 0000 0000 0026 0000 0000 0000 0028
0000 0010 0000 0000 0000 0023 0000 002c
0000 000a 0000 0000 0000 002d 0000 0029
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0a78 0000 0000
0300 000d 0000 002c 0000 22b4 0000 0008
1200 000e 0000 0001 0000 0001 0000 0000
1300 fff1 0000 00c7 0000 1314 0000 0124
1200 000e 0000 019d 0000 1c24 0000 00dc
1200 000e 0000 0182 0000 18ac 0000 01d8
1200 000e 0000 00db 0000 1128 0000 00a4
1200 000e 0000 012b 0000 1438 0000 0124
1200 000e 0000 00f5 0000 11cc 0000 00a4
1200 000e 0000 01bf 0000 1b64 0000 00c0
1200 000e 0000 021d 0000 2410 0000 0004
1100 0011 0000 017d 0000 16f8 0000 01b4
1200 000e 0000 0012 0001 25a0 0000 0000
```

```
1100 0017 0000 001c 0000 2210 0000 00a4
1200 000e 0000 0203 0000 2080 0000 00e8
1200 000e 0000 0192 0000 1d00 0000 004c
1200 000e 0000 014a 0000 1560 0000 0198
1200 000e 0000 0049 0000 0c80 0000 0404
1200 000e 0000 0106 0000 1270 0000 00a4
1200 000e 0000 01e5 0000 1a84 0000 0060
1200 000e 0000 00b6 0000 1084 0000 00a4
1200 000e 0000 01d4 0000 1ae4 0000 0080
1200 000e 0000 0026 0000 0a78 0000 0000
1200 000d 0000 020c 0000 2168 0000 009c
1200 000e 0000 0079 0000 0000 0000 0000
2000 0000 0000 01e0 0000 23c0 0000 0000
1200 0000 0000 0152 0000 23b0 0000 0000
1200 0000 0000 00a2 0000 0000 0000 0000
```

7. Diagramas de Stack Frame

A continuación se muestran cómo quedan los stack frames, bajo determinadas circunstancias, de las funciones codificadas en Assembly en `merge_sort.S`.

7.1. merge_sort

Antes de realizar el llamado a `merge_sort_rec` el stack frame de esta función se encuentra de la siguiente forma.

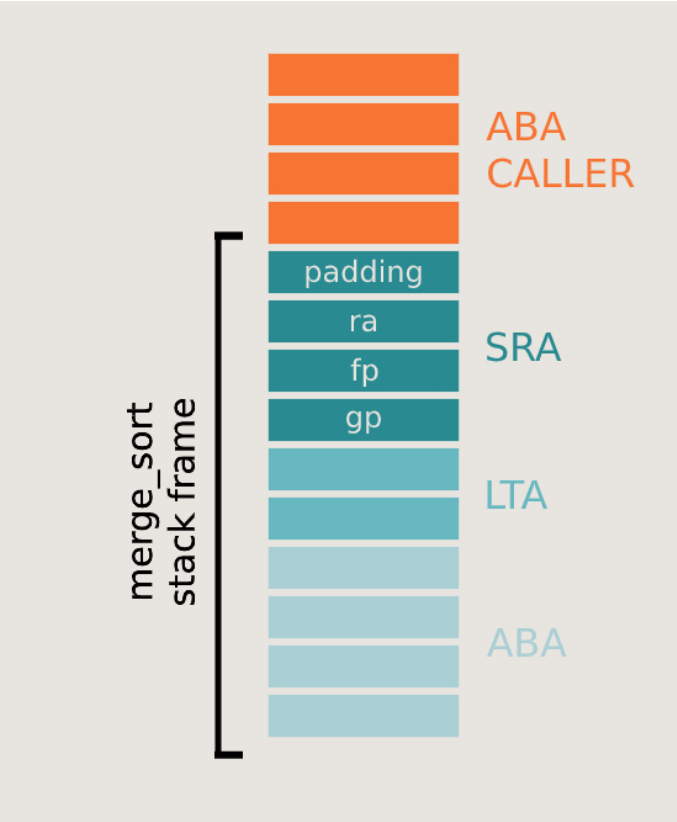

Figura 5: Stack frame de merge_sort

7.2. merge_sort_rec

En este caso, se ilustra el estado del stack frame luego de haber calculado y salvado exitosamente el valor de la variable considerada como 'medio'.

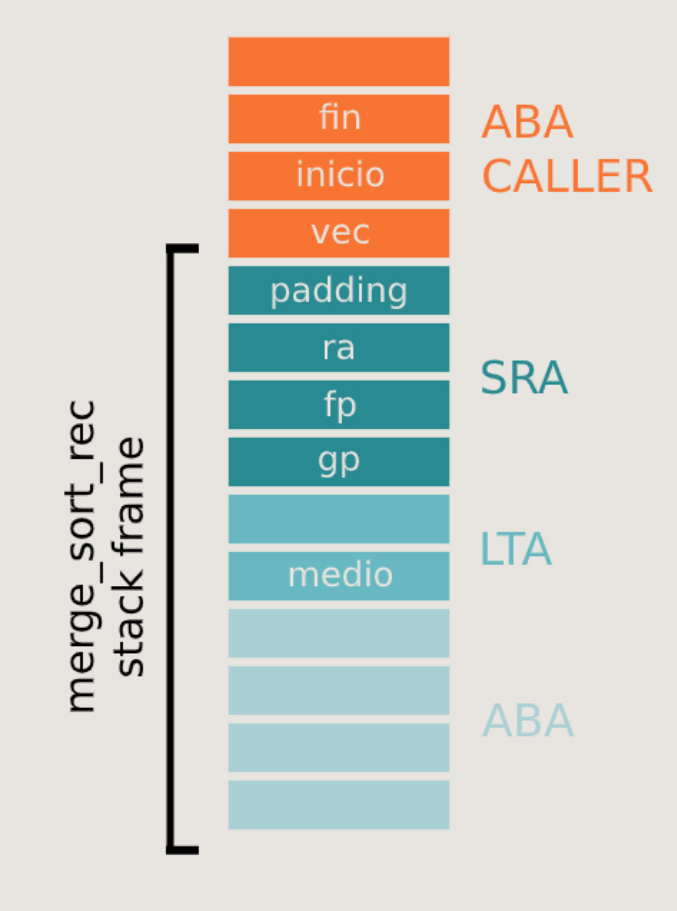

Figura 6: Stack frame de merge_sort_rec

7.3. merge

Finalmente se muestra cómo se encuentra el stack frame de esta función una vez se reservó exitosamente la memoria para los vectores `aux1` y `aux2`.

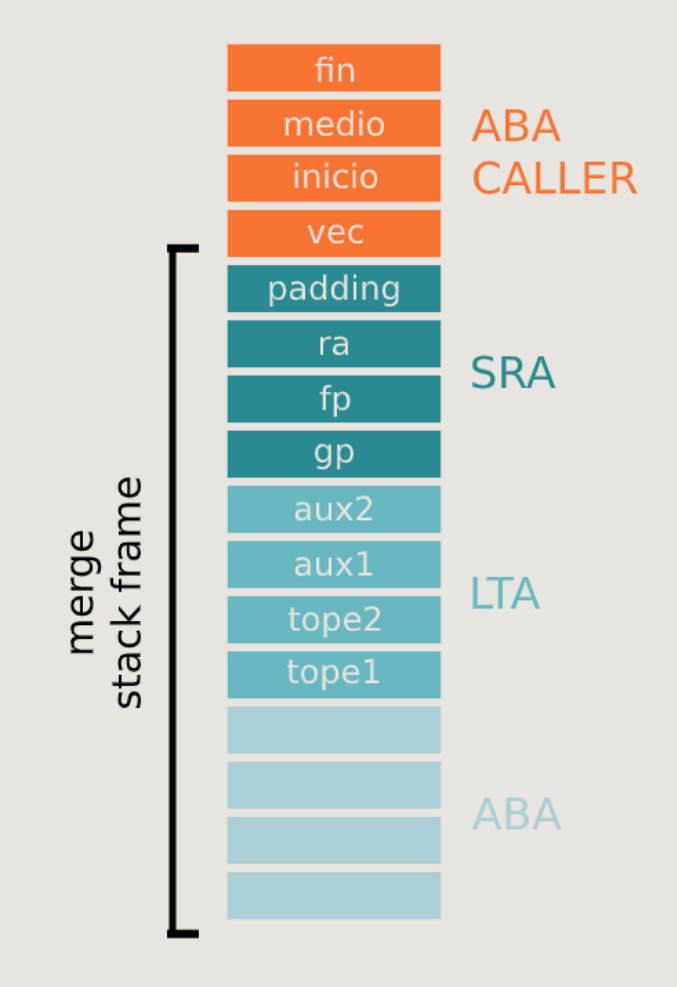

Figura 7: Stack frame de merge

8. Conclusiones

Durante la resolución del problema planteado nos encontramos frente a varios problemas. Algunos fueron durante la escritura del código en MIPS32: valores en ciertos registros mal asignados, dificultad a la hora de manejarnos con tantos registros pero a la vez limitados por el uso que se le debe dar a cada tipo(los registros de tipo argumento son únicamente 4 y los temporales son 8), entre otros. Sin embargo escribir el código lo más claro posible fue de gran ayuda para a superar estos problemas. Otra herramienta muy útil fueron los diagramas del stack frame que nos permitieron visualizar mejor en que sector del stack frame debería encontrarse cada registro. En conclusión, logramos familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI, logrando de esta manera desarrollar un programa que cumple con lo pedido por la consigna.

9. Enunciado del TP1

Universidad de Buenos Aires - FIUBA66.20 Organización de ComputadorasTrabajo práctico 1: Programación MIPS1er cuatrimestre de 2020

9 de junio de 2020

9.1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto a continuación.

9.2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

9.3. Requisitos

El trabajo debería ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

9.4. Descripción

El programa a desarrollar deberá procesar un *stream* de vectores de números enteros. A medida que el programa avance en la lectura de estos, deberá ordenar cada vector en forma creciente, e imprimir inmediatamente el resultado por el *stream* de salida.

Los vectores ingresaran como texto por entrada estándar (**stdin**), donde cada linea describe completamente el contenido del mismo, según el siguiente formato:

```
v1 v2 ... vN
```

El fin de linea es el carácter n (*newline*). Debido a que cada linea contiene exactamente un único vector, el fin del mismo coincidirá siempre con el final de la linea que lo contiene. A su vez, cada entero del vector estará separado de otros elementos por uno o más caracteres de espacio en blanco.

Por ejemplo, dado el siguiente flujo de entrada:

```
$ cat input.txt
3 2 1
6 5 1 2 9 3 8 7 4    9
6 0 0 1 3
-1
```

Al ejecutar el programa la salida sería:

```
$ tp1 -i input.txt -o -
1 2 3
1 2 3 4 5 6 7 8 9
0 0 1 3 6
-1
```

Ante un error, el programa debería detenerse informando la situación inmediatamente (por **stderr**).

9.4.1. Ejemplos

Primero, usamos la opción **-h** para ver el mensaje de ayuda:

```
$ tp1 -h
Usage:
tp1 -h
tp1 -V
tp1 -i in_file -o out_file
Options:
-V, --version Print version and quit.
-h, --help Print this information and quit.
-i, --input Specify input stream/file, "-" for stdin.
-o, --output Specify output stream/file, "-" for stdout.
Examples:
tp1 < in.txt > out.txt
cat in.txt | tp1 -i - > out.txt
```

A continuación, ejecutamos algunas pruebas:

```
$ cat example.txt
1
-1     +1
$ cat example.txt | ./tp1
1
-1 1
```

9.5. Implementación

El programa a desarrollar constará de una mezcla entre código MIPS32 y C, siendo la parte escrita en *assembly* la encargada de ordenar un vector de enteros pasado por parámetro. El formato de dicha función será:

```
void merge_sort(int *vec, size_t len);
```

Asimismo deberá usarse el algoritmo merge sort y el modo 1 del sistema operativo para manejo de acceso no alineado a memoria.

9.6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C y MIPS;
- El código MIPS32 generado por el compilador;
- Este enunciado.

9.7. Fechas

Fecha de vencimiento: martes 26/5.