Trabajo Práctico Nro 1
Organización de Computadoras

Lambre, Juan Manuel *95978*
juanmlambre@gmail.com
Israel, Pablo *95849*
pabloisrael94@gmail.com

Resumen

El siguiente informe explica y detalla el trabajo práctico número 1 de la materia.

Índice

1. Objetivo

El objetivo de este Trabajo Práctico fue el de familiarizarse con la arquitectura MIPS a través del emulador "GxEmul", con el ABI que utiliza la misma y correr en esta un programa C junto con una función implementada en Assembler.

2. Introducción

2.1. GxEmul

GxEmul es un emulador de la arquitectura de computadores MIPS libre que en este caso lo utilizamos sobre la plataforma Linux. Para este trabajo práctico se pedía correr parte del programa sobre la arquitectura MIPS, por lo que fue utilizado para ese propósito.

2.2. Arquitectura MIPS

MIPS es una arquitectura de microprocesadores RISC desarrollada por MIPS Technologies. La misma es utilizada en múltiples sistemas integrados como Series2 TiVo, dispositivos Windows CE, routers Cisco y consolas de videojuegos como el Nintendo 64, la PlayStation, PlayStation 2, etc. Para este trabajo práctico se pedía realizar una función en el Assembler de esta arquitectura.

2.3. Stack Arquitectura MIPS

El Stack de una función e la Arquitectura MIPS utilizado por la cátedra se compone de la siguiente manera:

Saved Registers Area (SRA)
Float Registers Area
Local Area (LTA)
Arguments Building Area (ABA)

Cuadro 1: Stack Arquitectura MIPS

El RSA se compone por:

Alignment[4]
ra
fg
gp

Cuadro 2: RSA

donde fp, gp y ra ocupan 4 Bytes, pero ra y el segmento de alineación no son obligatorios si la función no es hoja.

El LTA es el área donde se guardan variables locales, tales como registros temporales ($ti) antes de llamar a una función, o registros permanentes ($si). Esta área es de uso opcional, pero como todas debe tener un tamaño múltiplo de 8 bytes.

Finalmente, el ABA se compone por:

a3
a2
a1
a0

Cuadro 3: ABA

El mismo debe ser mayor o igual que 16 Bytes y en caso de pasarse más argumentos que 4 (a0, a1, a2, a3), se pueden depositar los mismos arriba de a3, pero siempre respetando que la estructura del stack debe ser múltiplo de 8. Cabe mencionar que estos valores los guarda la función que es llamada (callee) en el stack de la función que la llamó (caller)

3. Código

3.1. Diseño del código

El código del programa se puede dividir en dos partes, el parseo del archivo en el cual se van cargando todos los elementos del mismo en un array y la segunda parte es la función "my_qsort"que se encarga de ordenar los elementos del array (tomándolos como chars o como números dependiendo del parámetro ingresado al invocar el programa). Esta función fue implementada en C y en Assembly.

3.2. Código Assembly

Uno de los objetivos principales del trabajo práctico es la implementación de la función "my_qsort"que se encarga de ordenar un vector de vectores de chars en código Assembly. Para ello se deben tener en cuenta los nombres de los registros que se utilizan en Assembly Mips32. En la siguiente tabla se muestran los nombres con los que se usan los 32 registros:

Nombre del Registro	Número del Registro	Uso
zero	0	constante 0
at	1	temporal del ensamblador
v0-v1	2-3	valores de retorno
a0-a3	4-7	argumentos de funciones
t0-t7	8-15	registros temporales
s0-s7	16-23	registros permanentes
t8-t9	24-25	registros temporales
k0-k1	26-27	reservados para el kernel
gp	28	global pointer
sp	29	stack pointer
fp	30	frame pointer
ra	31	return addres

Cuadro 4: Registros MIPS32.

La función recibe como parámetros la dirección del primer elemento del vector, la dirección del último elemento y un flag que indica si la comparación debe ser viendo cada elemento como un vector de chars o como uno de enteros. A partir de estos argumentos se aplica el método de ordenamiento quick sort que consiste en lo siguiente:

1. Elegir un elemento del vector de elementos a ordenar, al que llamaremos pivot. En nuestra implementación elegimos siempre el primer elemento del vector como pivot.

2. Reubicar los demás elementos del vector a cada lado del pivot, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivot pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación. En este momento, el pivot ocupa exactamente el lugar que le corresponderá en el vector ordenado.

3. El vector queda separado en dos subvectores, uno formado por los elementos a la izquierda del pivot, y otro por los elementos a su derecha.

4. Repetir este proceso de forma recursiva para cada subvector mientras estos contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

El Stack que implementa esta función, de acuerdo a la ABI utilizada por la cátedra, es el siguiente:

Offset sp	Registros	Área
36	[padding]	RSA
32	ra	RSA
28	fp	RSA
24	gp	RSA
20	[padding]	LTA
16	curmax	LTA
12	a3	ABA
8	a2	ABA
4	a1	ABA
0	a0	ABA

Cuadro 5: Diagrama del Stack.

En el diagrama se puede apreciar que se guarda el registro ra en memoria ya que la función no es hoja por este motivo hubo que agregar un padding en el RSA para que sea múltiplo de 8. También se puede observar que se dejó lugar para una variable local (curmax) la cual se guarda antes de realizar la primer llamada recursiva y se accede nuevamente a la hora de llamar a la segunda llamada recursiva (también hubo que agregar un padding por el mismo motivo que en RSA).

3.3. Repositorio

`https://github.com/JuanmaLambre/6620-tp1`

3.4. Instructivo de instalación

3.4.1. Versión Assembly

El mismo se compila cómo:

```
$ chmod +x build_asm
$ ./build_asm
```

3.4.2. Versión C

El mismo se compila cómo:

```
$ chmod +x build_c
$ ./build_c
```

3.4.3. Cómo correr los tests de funcionalidad

Ejecutar

```
$ bash ./test/test
```

Esto compila automáticamente y corre los tests que estén declarados dentro del mismo archivo. Si además se desea correr una ejecución con Valgrind, setear la variable RUN_VALGRIND=y

3.4.4. Cómo utilizar el programa

Uso:

```
qsort -h
qsort -V
qsort [options] archivo
Opciones:
-h, --help Imprime ayuda.
-V, --version Versi n del programa.
-o, --output Archivo de salida.
-n, --numeric Ordenar los datos num ricamente en vez de alfab ticamente.
```

Ejemplo: **./qsort -n numeros.txt**

4. Casos de prueba

A continuación se ilustran distintos casos que ejemplifican el uso del programa.

Ayuda del programa:

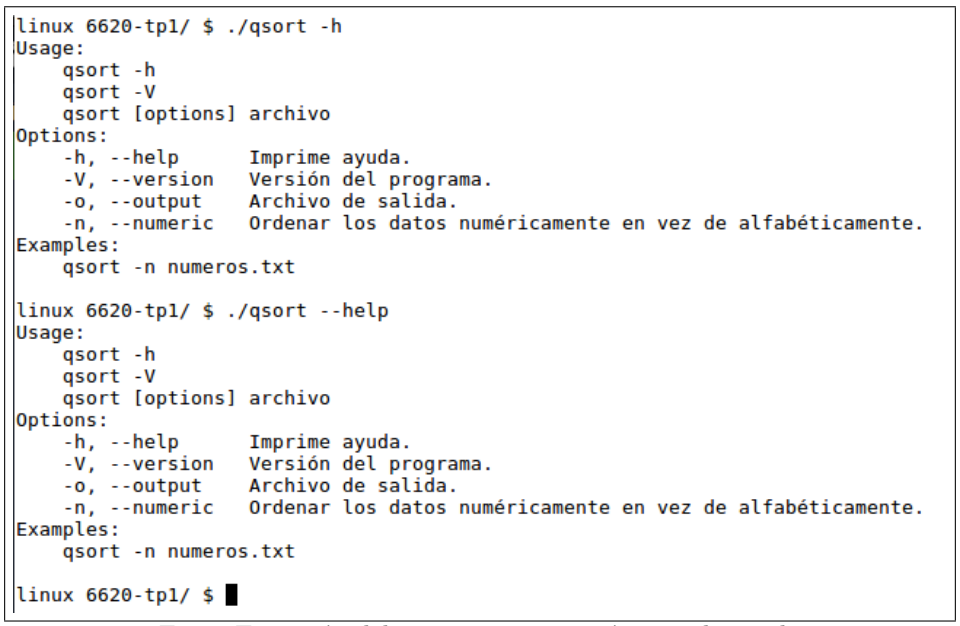

Fig 1: Ejecución del programa con parámetro de ayuda

Versión del programa:

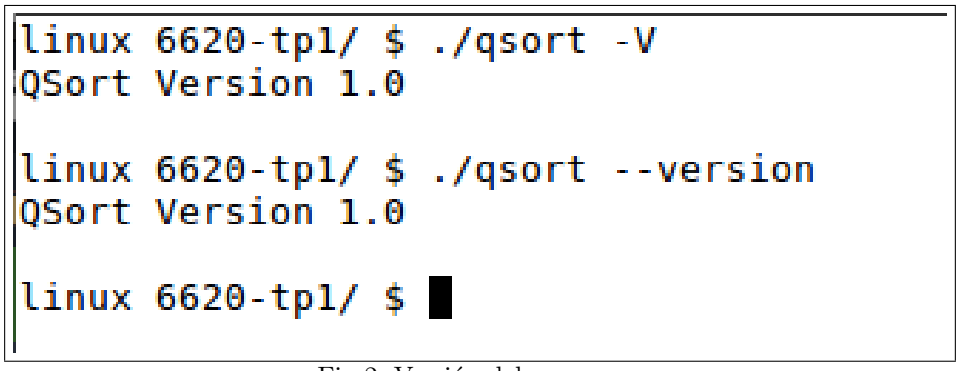

Fig 2: Versión del programa

Archivo de entrada de ejemplo:

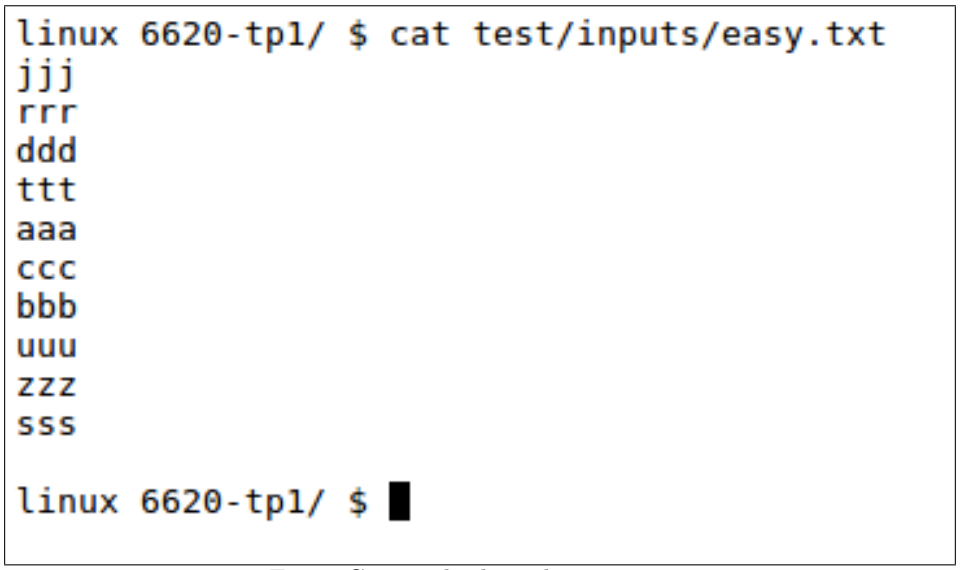

Fig 3: Contenido de archivo easy.txt

Corrida de programa con easy.txt como archivo de entrada:

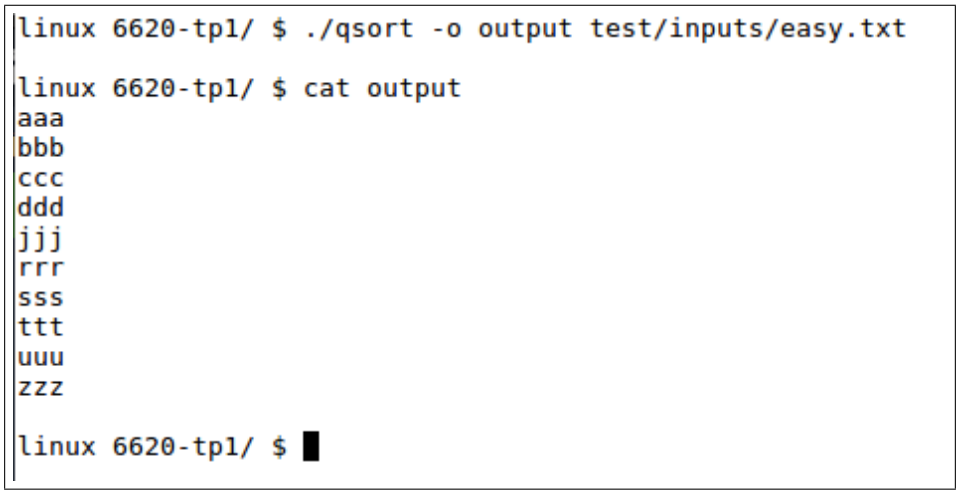

Fig 4: Resultado del programa

En la figura 4 se puede observar que se invoca el programa pasándole como parámetro el archivo donde se quiere guardar la salida y el archivo de entrada (easy.txt). Al ver el contenido del archivo de salida se ve que está ordenado alfabéticamente.

Corrida de programa con archivo de entrada con números:

```
linux 6620-tp1/ $ ./qsort test/inputs/numeros.txt
1
10
2
3
4
5
6
7
8
9

linux 6620-tp1/ $
```

Fig 5: Resultado del programa

En la figura 5 se puede observar que se invoca el programa con el archivo numeros.txt como archivo de entrada y se ordena alfabéticamente (según el código ascii).

Corrida de programa para ordenar numéricamente:

```
linux 6620-tp1/ $ ./qsort -n test/inputs/numeros.txt
1
2
3
4
5
6
7
8
9
10

linux 6620-tp1/ $
```

Fig 6: Resultado del programa

En la figura 6 se puede observar que se invoca el programa con el archivo numeros.txt como archivo de entrada y se agrega el parámetro -n para indicar que se ordene numéricamente como se puede observar en la sálida del programa.

5. Mejoras

En la siguiente sección se detallan algunas mejoras del desarrollo del trabajo que quedaron pendientes debido a cuestiones de tiempo y prioridad.

Funciones internas. En el código ASM, el funcionamiento de las comparaciones (tanto la numérica como la alfabética) quedó codificado como una porción de código al mismo nivel que el resto de my_qsort. Sería más correcto tener dicho encapsulamiento en una o dos funciones separadas al código principal.

Parser de argumentos. El parseo de argumentos en C se implementó manualmente. Dado que existen librerías públicas que automatizan esta tarea podría ser más práctico aprovecharlas.

Test de my_qsort. Las únicas pruebas que se hicieron sobre el código en ASM fueron manuales; nunca se testeó el código de manera automática. Quedaron pendientes hacer tests de funcionalidad automáticos para dicha función.

6. Conclusiones

El trabajo realizado nos permitió conocer y estudiar en un nivel práctico la arquitectura MIPS a través de un emulador Pudimos comparar las dos versiones del trabajo práctico (la realizada en lenguaje C y la realizada en lenguaje C junto con Assembly Mips) a través de los archivos de salida que estos generaban al correr el programa con los mismos parámetro. Comparando ambos archivos de salida y observar que son iguales (como era de esperarse) podemos concluir que la función en Assembly funciona correctamente.

Referencias

[1] GXemul, http://gavare.se/gxemul/.

[2] Apunte que provee la cátedra para instalar GxEmul.

[3] Apunte que provee la cátedra sobre cómo utilizar la ABI.

7. Anexo - Código fuente

7.1. main.c

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

#define CHUNK_SIZE 10


typedef struct {
    FILE* output;
    FILE* input;
    int numeric;
} Arguments;


int compare(char* ptr1, char* ptr2, int num) {
    if (num) return atoi(ptr1) - atoi(ptr2);
    else return strcmp(ptr1, ptr2);
}

#ifdef ASM_QSORT
extern my_qsort(char** left, char** right, int num);
#else
void my_qsort(char** left, char** right, int num) {
    if (left < right) {
        char* pivot = *left;
        char** curMin = left+1;
        char** curMax = right;

        while (curMin <= curMax) {
            while (curMin <= right && compare(*curMin, pivot, num) <= 0)
                ++curMin;

            while (curMax >= left && compare(*curMax, pivot, num) > 0)
                --curMax;

            if (curMin < curMax) {
                char* aux = *curMin;
                *curMin = *curMax;
                *curMax = aux;
            }
        }

        char* aux = *left;
        *left = *curMax;
        *curMax = aux;

        my_qsort(left, curMax-1, num);
        my_qsort(curMax+1, right, num);
    }
}
#endif

void print_help() {
    printf("Usage:\n");
    printf("    qsort -h\n");
    printf("    qsort -V\n");
```

```
58      printf("    qsort [options] archivo\n");
59      printf("Options:\n");
60      printf("    -h, --help      Imprime ayuda.\n");
61      printf("    -V, --version   Versi n del programa.\n");
62      printf("    -o, --output    Archivo de salida.\n");
63      printf("    -n, --numeric   Ordenar los datos num ricamente en vez de
            alfab ticamente.\n");
64      printf("Examples:\n");
65      printf("    qsort -n numeros.txt\n");
66  }
67
68  void print_version() {
69      printf("QSort Version 1.0\n");
70  }
71
72  void print_output(char** lines, int count, FILE* output) {
73      for (int i = 0; i < count; ++i) {
74          fprintf(output, "%s\n", lines[i]);
75      }
76  }
77
78  void free_lines(char** lines, int count) {
79      for (int i = 0; i < count; ++i) {
80          free(lines[i]);
81      }
82      free(lines);
83  }
84
85  char* read_line(FILE* input) {
86      char* str = (char*) malloc(sizeof(char));
87      *str = '\0';
88      int len = 0;
89      char c;
90
91      if (feof(input)) return NULL;
92
93      while (EOF != (c=fgetc(input)) && c != '\n') {
94          ++len;
95          str = (char*) realloc(str, sizeof(char)*(len+1));
96          str[len-1] = c;
97          str[len] = '\0';
98      }
99
100     if (c == EOF && len == 0) {
101         free(str);
102         return NULL;
103     }
104
105     return str;
106 }
107
108 int parse_file(FILE* input, void** dest) {
109     char** lines = NULL;
110     char* line = NULL;
111     int count = 0;
112
113     while (line = read_line(input)) {
114         int length = strlen(line);
115
116         if (length > 0) {
117             if ((count % CHUNK_SIZE) == 0) {
118                 int amount = (count/CHUNK_SIZE+1)*CHUNK_SIZE;
```

```
                lines = (char**) realloc(lines, amount*sizeof(void*));
            }
            lines[count] = line;
            ++count;
        }
    }

    *dest = lines;
    return count;
}

void process_file(FILE* input, int numeric, FILE* output) {
    char** lines = NULL;
    int count = 0;

    count = parse_file(input, &lines);

    my_qsort(lines, &lines[count-1], numeric);

    print_output(lines, count, output);

    free_lines(lines, count);
}

int parse_args(char** argv, int argc, Arguments* args, int* error) {
    if (argc == 1) {
        fprintf(stderr, "No arguments passed. Use -h option for usage help\n");
        *error = EINVAL;
        return 1;
    }

    // Set defaults
    args->output = stdout;
    args->input = NULL;
    args->numeric = 0;
    char* output_name = NULL, *input_name = NULL;

    for (int i = 1; i < argc; ++i) {
        char* arg = argv[i];
        if (strcmp(arg, "-h") == 0 || strcmp(arg, "--help") == 0) {
            print_help();
            return 1;
        } else if (strcmp(arg, "-V") == 0 || strcmp(arg, "--version") == 0) {
            print_version();
            return 1;
        } else if (strcmp(arg, "-o") == 0 || strcmp(arg, "--output") == 0) {
            if (i == argc - 1) {
                fprintf(stderr, "No file specified after -o argument\n");
                *error = EINVAL;
                return 1;
            } else if (strcmp(argv[i+1], "-") != 0) {
                output_name = argv[i+1];
                args->output = fopen(output_name, "w");
            }
        } else if (strcmp(arg, "-n") == 0 || strcmp(arg, "--numeric") == 0) {
            args->numeric = 1;
        }
    }

    input_name = argv[argc-1];
    if (input_name == output_name) {
        fprintf(stderr, "No input specified\n");
```

```
        *error = EINVAL;
        return 1;
    }

    args->input = fopen(input_name, "r");
    if (!args->input) {
        fprintf(stderr, "Failed to open file %s, error %d\n", input_name, errno);
        *error = errno;
        return 1;
    }

    return 0;
}

int main(int argc, const char** argv) {
    Arguments args;
    int ret = 0;

    int finish = parse_args(argv, argc, &args, &ret);
    if (finish) return ret;

    process_file(args.input, args.numeric, args.output);

    return 0;
}
```

src/main.c

7.2. my_qsort.S

```
1  #include <mips/regdef.h>
2          .text
3          .align 2
4          .abicalls
5          .globl my_qsort
6          .ent my_qsort
7
8  my_qsort:
9          #creo StackFrame
10
11
12         subu    sp, sp, 40              # Inicializo stack
13         sw      ra, 32(sp)
14         sw      $fp, 28(sp)
15         sw      gp, 24(sp)
16         sw      a0, 40(sp)
17         sw      a1, 44(sp)
18         sw      a2, 48(sp)
19
20         subu    t0, a1, a0
21         blez    t0, EXIT
22         lw      t0, 0(a0)               # t0: pivot
23         addi    t1, a0, 4               # t1: curmin
24         addi    t2, a1, 0               # t2: curmax
25
26 GLOBAL_WHILE:
27         subu    t4, t2, t1              # First while checks for curmin <= curmax
28         bltz    t4, END_GLOBAL_WHILE
29
30 FIRST_WHILE:
31         subu    t4, a1, t1              # checks for curMin <= right
32         bltz    t4, END_FIRST_WHILE
33         lw      t4, 0(t1)               # set params to compare them later (curmin
               and pivot)
34         addi    t5, t0, 0
35         li      t8, 0
36         j       COMP
37 FIRST_COMP:
38         bgtz    v0, END_FIRST_WHILE
39         addi    t1, t1, 4
40         j       FIRST_WHILE
41 END_FIRST_WHILE:
42
43 SECOND_WHILE:
44         subu    t4, t2, a0              # checks for curMax >= left
45         blez    t4, END_SECOND_WHILE
46         lw      t4, 0(t2)               # set params to compare them later (curmax
               and pivot)
47         addi    t5, t0, 0
48         li      t8, 1
49         j       COMP
50 SECOND_COMP:
51         bltz    v0, END_SECOND_WHILE
52         subu    t2, t2, 4
53         j       SECOND_WHILE
54 END_SECOND_WHILE:
55         subu    t4, t1, t2              # checks for curMin < curMax
56         bgez    t4, GLOBAL_WHILE
57         lw      t3, 0(t1)               # get curmin value
```

```
58          lw      t4, 0(t2)               # get curmax value
59          sw      t4, 0(t1)               # swaps values between curmin and curmax
60          sw      t3, 0(t2)
61          j       GLOBAL_WHILE
62
63  END_GLOBAL_WHILE:
64          lw      t3, 0(a0)               # get left value
65          lw      t4, 0(t2)               # get curmax value
66          sw      t4, 0(a0)               # swaps values between left and curmax
67          sw      t3, 0(t2)
68
69  RECURSIVE:
70          sw      t2, 16(sp)
71          subu    a1, t2, 4
72          jal     my_qsort                # my_qsort(left, curMax-1, num)
73          lw      a1, 44(sp)              # right
74          lw      t2, 16(sp)              # retrieves curmax value
75          addiu   a0, t2, 4
76          jal     my_qsort                # my_qsort(curMax+1, right, num);
77  EXIT:
78
79          lw      ra, 32(sp)
80          lw      $fp, 28(sp)
81          lw      gp, 24(sp)
82          addiu   sp, sp, 40
83          jr      ra
84
85
86  COMP:
87          beqz    a2, COMP_STR
88          j       COMP_NUMBER
89
90  END_COMP:
91          beqz    t8, FIRST_COMP
92          j       SECOND_COMP
93
94
95
96
97  COMP_STR:
98          # compare two strings
99          # t4: first element char pointer
100         # t5: second element char pointer
101         lb      t6, 0(t4)               # t6: first element char
102         lb      t7, 0(t5)               # t7: second element chat
103         beq     t6, zero, ENDZ_CUR
104         beq     t7, zero, ENDZ_PIVOT
105         slt     t9, t6, t7
106         bgtz    t9, END_SMALLER
107         slt     t9, t7, t6
108         bgtz    t9, END_BIGGER
109         addi    t4, t4, 1
110         addi    t5, t5, 1
111         j       COMP_STR
112 ENDZ_CUR:
113         beq     t7, zero, END_EQ
114         j       END_SMALLER
115 ENDZ_PIVOT:
116         beq     t6, zero, END_EQ
117         j       END_BIGGER
118 END_EQ:
119         li      v0, 1                   # yafu, idem a comp_number
```

```
120         j       FINISH_COMP
121 END_BIGGER:
122         li      v0, 1
123         j       FINISH_COMP
124 END_SMALLER:
125         li      v0, -1
126 FINISH_COMP:
127         j       END_COMP
128
129
130
131
132
133 COMP_NUMBER:
134         # Compare numbers
135         # t4: first element char pointer
136         # t5: second element chat pointer
137         li      t3, 0
138         j       ATOI
139 FIRST_END_ATOI:
140         addi    t6, v0, 0               # t6: first int value
141         li      t3, 1
142         addi    t4, t5, 0
143         j       ATOI
144 SECOND_END_ATOI:
145         addi    t7, v0, 0               # t7: second int value
146         subu    t7, t7, t6
147         li      v0, -1
148         bgtz    t7, END_COMP
149         li      v0, 1
150 END_COMP_NUMBER:
151         j       END_COMP
152
153 ATOI:
154         li      v0, 0                   # v0 va a tener el resultado final
155 LOOP:
156         lb      t7, 0(t4)
157         beqz    t7, END
158         subu    t7, t7, 48
159         li      a3, 10
160         mult    v0, a3                  # muli doesn't wotk, i had to use mult
161         mflo    v0
162         addu    v0, v0, t7
163         addi    t4, t4, 1
164         j       LOOP
165 END:
166         beqz    t3, FIRST_END_ATOI
167         j       SECOND_END_ATOI
168
169         .end my_qsort
```

src/my_qsort.S