FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

[66.20] ORGANIZACIÓN DE COMPUTADORAS

1º CUATRIMESTRE 2020

CURSO 1 - MARTES

TP1 - MergeSort en MIPS-32

AUTORES:

Carbón Posse, Ana Sofía - #101 187
<scarbon@fi.uba.ar>

Fandos, Nicolás Gabriel - #101 018
<nfandos@fi.uba.ar>

Torraca, Facundo - #101 046
<ftorraca@fi.uba.ar>

DOCENTES

Santi, Leandro

Natale, Luciano César

Perez Masci, Hernan

Índice

1. Enunciado

1.1. Objetivos

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

1.2. Alcance

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

1.3. Requisitos

El trabajo deberá 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.

1.4. 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 mas 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 progama deberá detenerse informando la situación inmediatamente (por `stderr`).

1.5. 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
```

1.6. Implementación

El programa a desarrollar constara 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:

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

Así mismo deberá usarse el algoritmo mergesort y el modo 1 del sistema operativo para manejo de acceso no alineado a memoria. En cuanto al manejo de memoria dinámica realizado por mergesort(), deberá realizarse en MIPS usando la implementación de referencia mymalloc disponible en el campus.

1.7. Entrega de TP

La entrega de este trabajo deberá realizarse usando el campus virtual de la materia. Asimismo, en todos los casos, estas presentaciones deberán ser realizadas durante los días martes. El feedback estará disponible de un martes hacia el otro, como ocurre durante la modalidad presencial de cursada. Por otro lado, la ultima fecha de entrega y presentación para esta trabajo sera el martes 26/5.

2. Documentación acerca del diseño e implementación del programa

El objetivo de este primer trabajo practico es familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, a través de la creación de un algoritmos de ordenamiento conocido como MergeSort.

Lo primero que decidimos hacer fue crear el trabajo completamente en el lenguaje C, escribiendo un programa que permita leer las lineas de un archivo de texto (o bien lineas escritas por entrada estándar por terminal), parsearlas a un vector de números enteros, ordenar el vector con el algoritmo mergesort y finalmente escribir el vector ordenado como una linea nueva en el archivo de salida, el cual puede ser indicado por terminal o puede ser salida estándar por la terminal.

Luego utilizando como referencia el código escrito para el ordenamiento mergesort en C procedimos a traducirlo linea a linea a código Assembly MIPS32 respetando la ABI que manejan los procesadores MIPS y finalmente utilizamos la máquina virtual qemu la cual tiene un sistema operativo Debian con kernel linux para ejecutar el código con la función en assembly y asegurarnos de que todo funciona correctamente.

En ultima instancia, ya habiendo realizado las pruebas pertinentes para verificar que el funcionamiento del programa sea el esperado, el esquema del código del mismo puede resumirse en un archivo el cual tiene los algoritmos escritos en C para leer las lineas de un archivo y pasarlas a un vector de enteros (y poder mostrar estos vectores por salida estandar o escribirlos como lineas de un archivo), otro archivo que tiene la implementación en C de una abstracción de un vector de enteros (llamado adecuadamente "vector") y finalmente los archivos de la función de ordenamiento mergesort escritos en MIPS, los cuales respetan la ABI y aseguran que los accesos a memoria realizados sean alineados a 4 bytes.

3. Comandos de compilación

El comando utilizado para compilar el programa y obtener el ejecutable correctamente es:

- gcc -Wall -g -o **nombre_del_ejecutable** vsorter.c mergesort.S

También, se puede compilar y generar el ejecutable .S, se puede hacer a través de QEMU, el emulador de un sistema operativo (Debian con kernel Linux) en MIPS32.

- gcc -Wall -S vsorter.c mergesort.S

4. Corrida de pruebas

En esta sección mostraremos diversas pruebas realizadas para comprobar el correcto funcionamiento del programa.

Para empezar, compilamos el archivo para obtener el ejecutable con la siguiente linea:

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# gcc -Wall -g -o mergesort vsorter.c mergesort.S
```

4.1. Comandos de ayuda

La primera prueba es para el comando de 'Help', expresado con el flag '-h', este le muestra al usuario por terminal las distintas opciones que tiene para ejecutar el programa.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort -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
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1#
```

Una forma alternativa de brindarle esta información al usuario, es ingresando '–help'.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort --help
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
```

4.2. Comandos de versión

De la misma manera que podemos brindar ayuda con el comando anterior, el usuario también puede obtener mediante el flag '-v' información sobre la versión del programa.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort -V
vsorter version: 1.0.0
```

Nuevamente existe una alternativa para obtener esta información, que es utilizando '–version'.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort --version
vsorter version: 1.0.0
```

4.3. Pruebas

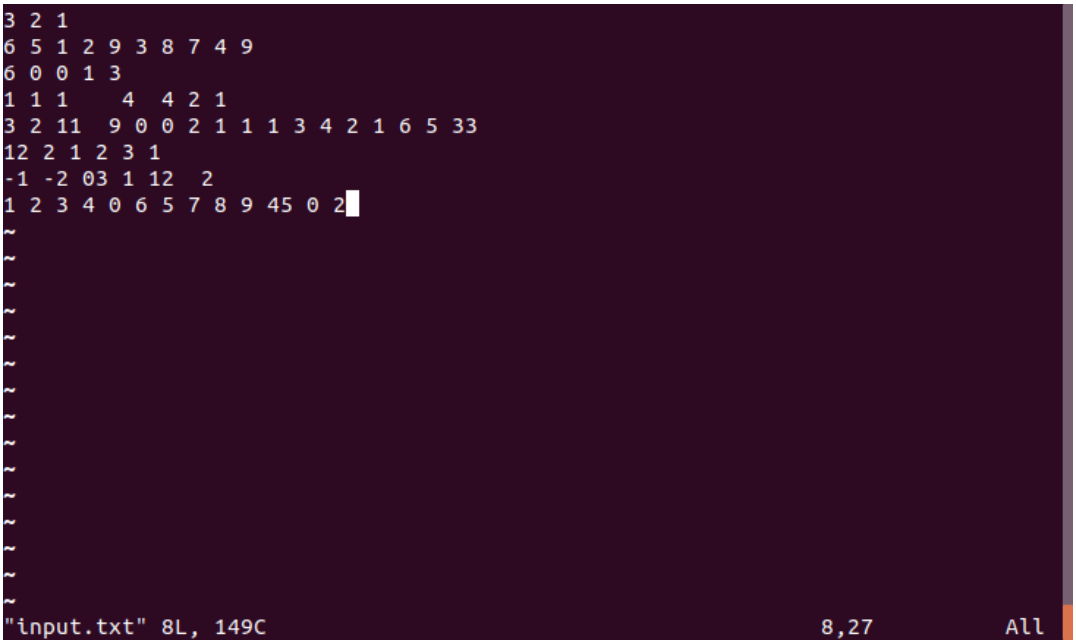
Para realizar las pruebas del algoritmo de ordenamiento Merge Sort, creamos una archivo de texto con la siguiente estructura:

```
3 2 1

6 5 1 2 9 3 8 7 4 9

6 0 0 1 3

1 1 1 4 4 2 1

3 2 11 9 0 0 2 1 1 1 3 4 2 1 6 5 33

12 2 1 2 3 1

-1 -2 03 1 12 2

1 2 3 4 0 6 5 7 8 9 45 0 2
```

Este archivo lo nombramos como **'input.txt'**.

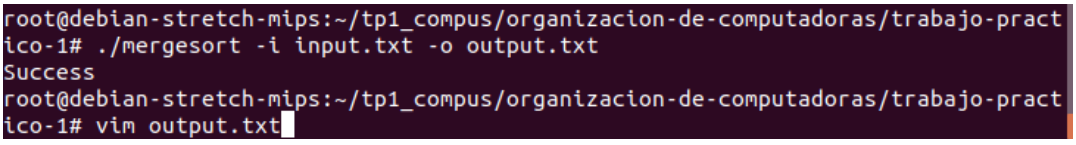
A continuación mostramos el archivo ya creado en el sistema y lo abrimos con el editor de texto en terminal VIM.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# vim input.txt
```

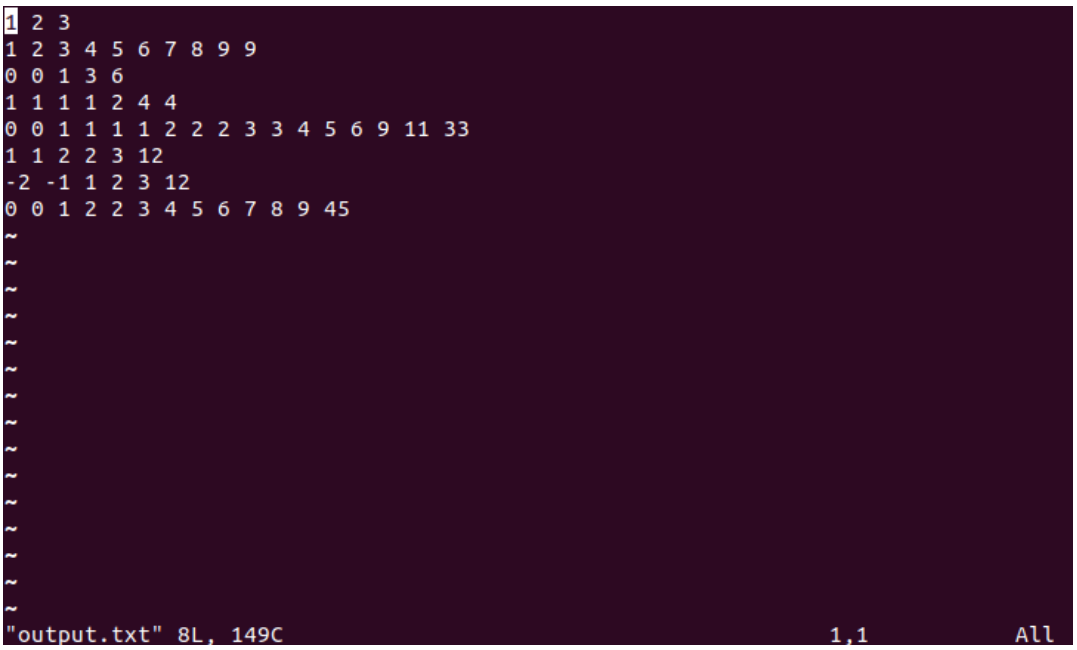


4.3.1. Prueba 1 - Ordenamiento de vectores de un archivo inicial, guardado en otro archivo

La siguiente linea nos va a permitir mostrar que los vectores recibidos por el archivo de entrada ’input.txt’ (mostrados previamente) van a ser ordenados y luego almacenados en el archivo de salida ’output.txt’

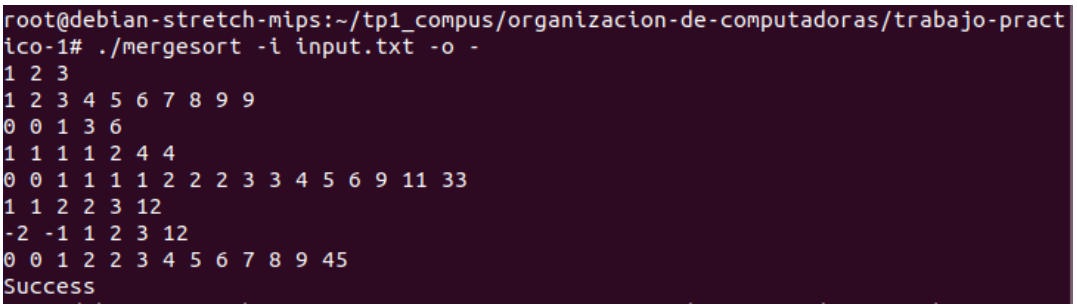


Lo que podemos observar al abrir el archivo 'output.txt' con VIM nos muestra lo siguiente:



4.3.2. Prueba 2 - Ordenamiento de vectores en un archivo, mostrados por salida estándar

Ahora vamos a mostrar el funcionamiento del programa recibiendo un archivo de entrada (el mismo de la sección anterior, llamado 'input.txt') y devolviendo el resultado por salida estándar (`stdout`):



4.3.3. Prueba 3 - Ordenamiento de vectores ingresados por entrada estándar, guardado en un archivo

Acá vamos a mostrar el funcionamiento del programa recibiendo un vector a ordenar por entrada estándar (`stdin`) y devolviendo el resultado en un archivo de salida llamado 'output_2':

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort -i - -o output_2.txt
1 5 1 2 5 1 6 76 72  24 536 5674 213 3 5 5
1 53 6 3 7 8 3 3 5 7 8 235
235 3456 567 679 54 42 5346 5 573 34632
Success
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# vim output_2.txt
```

Aca podemos observar en VIM lo que se encuentra dentro del archivo de salida 'output_2.txt', que como esperabamos, son las mismas lineas ingresadas por entrada estandar ordenadas:

```
1 1 1 2 3 5 5 5 5 6 24 72 76 213 536 5674
1 3 3 3 5 6 7 7 8 8 53 235
5 42 54 235 567 573 679 3456 5346 34632
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"output_2.txt" 3L, 112C                                       1,1           All
```

4.3.4. Prueba 4 - Ordenamiento de vectores ingresados por entrada estándar, mostrado por salida estándar

Por ultimo, vamos a mostrar el funcionamiento del programa recibiendo un vector a ordenar por entrada estándar (`stdin`) y devolviendo el resultado por salida estándar (`stdout`):

- Notar que el código no deja de funcionar al recibir números negativos, letras o números combinados con letras.

```
root@debian-stretch-mips:~/tp1_compus/organizacion-de-computadoras/trabajo-pract
ico-1# ./mergesort -i - -o -
1 2 6 6 5 2 5 6 87 2 1 6 1
1 1 1 2 2 2 5 5 6 6 6 6 87
67 523 4 78412 5 3465 2 4523 47 23 5 67 -1 -76 12
-76 -1 2 4 5 5 12 23 47 67 67 523 3465 4523 78412
124 7643 ss gh54 357 11q 6346 -q 11
0 0 0 11 11 124 357 6346 7643
-1 -2 -3 -4 5 -6 7 -8
-8 -6 -4 -3 -2 -1 5 7
```

5. Código

5.1. vsorter.c

El **vsorter.c** es el archivo principal del programa. Su código permite manejar archivos de entrada y salida, tanto los estándar que provee el sistema operativo (Entrada y salida por terminal) como de archivos ingresados por linea de comandos y extraer de las lineas del archivo los números y parsearlos a vectores.

```
1
2  #define _GNU_SOURCE
3
4  #include <stdio.h>
5  #include "vector.h"
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdbool.h>
9  #include "mergesort.h"
10
11 #define ERROR -1
12 #define SUCCESS 0
13
14 #define STD_FILE "-"
15
16 #define DELIMITER " "
17 #define ENDLINE '\n'
18 #define NULLTER '\0'
19
20 /*-------------------------------exec-modes-------------------------------*/
21 #define H_MODE 1
22 #define V_MODE 2
23 #define I_MODE 3
24 #define O_MODE 4
25 #define IO_MODE 5
26 #define OI_MODE 6
27 #define STD_MODE 7
28 /*------------------------------------------------------------------------*/
29
30 /*---------------------------------flags----------------------------------*/
31 #define HELP_FLAG "-h"
32 #define HELP_FLAG_EXT "--help"
33
34 #define VERSION_FLAG "-V"
35 #define VERSION_FLAG_EXT "--version"
36
37 #define INPUT_FLAG "-i"
38 #define INPUT_FLAG_EXT "--input"
39
40 #define OUTPUT_FLAG "-o"
41 #define OUTPUT_FLAG_EXT "--output"
42 /*------------------------------------------------------------------------*/
43
44
```

```c
/*-------------------------get-flags-------------------------------*/
bool is_help_flag(const char* flag) {
    return (strcmp(flag, HELP_FLAG) == 0 ||
            strcmp(flag, HELP_FLAG_EXT) == 0);
}

bool is_version_flag(const char* flag) {
    return (strcmp(flag, VERSION_FLAG) == 0 ||
            strcmp(flag, VERSION_FLAG_EXT) == 0);
}

bool is_input_flag(const char* flag) {
    return (strcmp(flag, INPUT_FLAG) == 0 ||
            strcmp(flag, INPUT_FLAG_EXT) == 0);
}

bool is_output_flag(const char* flag) {
    return (strcmp(flag, OUTPUT_FLAG) == 0 ||
            strcmp(flag, OUTPUT_FLAG_EXT) == 0);
}
/*-----------------------------------------------------------------*/


/*--------------------get-exectuion-modes--------------------------*/
int get_2args_mode(char* const argv[]) {
    const char* flag = argv[1];

    if (is_help_flag(flag))
        return H_MODE;

    if (is_version_flag(flag))
        return V_MODE;

    return ERROR;
}

int get_3args_mode(char* const argv[]) {
    const char* flag = argv[1];

    if (is_input_flag(flag))
        return I_MODE;

    if (is_output_flag(flag))
        return O_MODE;

    return ERROR;
}

int get_5args_mode(char* const argv[]) {
    const char* flag_1 = argv[1];
    const char* flag_2 = argv[3];

    bool input_flag_frt = is_input_flag(flag_1);
    bool input_flag_scd = is_input_flag(flag_2);
```

```
99
100     bool output_flag_frt = is_output_flag(flag_1);
101     bool output_flag_scd = is_output_flag(flag_2);
102
103     if (input_flag_frt && output_flag_scd)
104         return IO_MODE;
105
106     if (output_flag_frt && input_flag_scd)
107         return OI_MODE;
108
109     return ERROR;
110 }
111
112 int get_exec_mode(int argc, char* const argv[]) {
113     switch (argc) {
114         case 1:
115             return STD_MODE;
116         case 2:
117             return get_2args_mode(argv);
118         case 3:
119             return get_3args_mode(argv);
120         case 5:
121             return get_5args_mode(argv);
122         default:
123             return ERROR;
124     }
125 }
126 /*------------------------------------------------------------------*/
127
128
129 /*--------------------auxiliary-functions-------------------------*/
130 void remove_endline(char* s) {
131     size_t len = strlen(s);
132     if (s[len - 1] == ENDLINE)
133         s[len - 1] = NULLTER;
134 }
135
136 int parse_vec_buffer(char* buffer, vector_t* vector) {
137     vector_clear(vector);
138
139     remove_endline(buffer);
140
141     char* str_num = strtok(buffer, DELIMITER);
142     while (str_num != NULL) {
143         int number = strtol(str_num, NULL, 10);
144         vector_push(vector, number);
145         str_num = strtok(NULL, DELIMITER);
146     }
147
148     return SUCCESS;
149 }
150
151 int read_vector(FILE* i_file, vector_t* vector) {
152     char* buffer = NULL;
```

```
153     size_t size_buff = 0;
154 
155     ssize_t bytes_read = getline(&buffer, &size_buff, i_file);
156 
157     if (bytes_read == ERROR || bytes_read == 0) {
158         if (bytes_read == ERROR) perror(NULL);
159         free(buffer);
160         return ERROR;
161     }
162 
163     parse_vec_buffer(buffer, vector);
164 
165     free(buffer);
166     return SUCCESS;
167 }
168 
169 void print_sorted_vec(FILE* o_file, vector_t* vector) {
170     if (vector) {
171         for (size_t i = 0; i < vector->size; i++) {
172             if (i < vector->size - 1)
173                 fprintf(o_file, "%d ", vector->array[i]);
174             else // Last element avoid space
175                 fprintf(o_file, "%d", vector->array[i]);
176         }
177     }
178 
179     fprintf(o_file,"\n");
180 }
181 /*----------------------------------------------------------------------*/
182 
183 
184 /*-----------------------exectuion-modes-------------------------------*/
185 int help() {
186     printf("Usage:\n");
187     printf("\ttp1 -h\n");
188     printf("\ttp1 -V\n");
189     printf("\ttp1 -i in_file -o out_file\n");
190     printf("Options:\n");
191     printf("\t-V, --version Print version and quit.\n");
192     printf("\t-h, --help Print this information and quit.\n");
193     printf("\t-i, --input Specify input stream/file, '-' for stdin\n");
194     printf("\t-o, --output Specify output stream/file, '-' for stdout.\n")
    ;
195     printf("Examples:\n");
196     printf("\ttp1 < in.txt > out.txt\n");
197     printf("\tcat in.txt | tp1 -i - > out.txt\n");
198     return SUCCESS;
199 }
200 
201 int version() {
202     printf("vsorter version: 1.0.0\n");
203     return SUCCESS;
204 }
205 
```

```c
void sort(FILE* i_file, FILE* o_file) {
    vector_t vector;
    vector_init(&vector);

    while (read_vector(i_file, &vector) == SUCCESS) {
        if (!vector_empty(&vector)) {
            merge_sort(vector.array, vector.size);
            print_sorted_vec(o_file, &vector);
        } else
            print_sorted_vec(o_file, NULL);
    }

    vector_destroy(&vector);
}
/*-------------------------------------------------------------------*/

int main(int argc, char* const argv[]) {
    char* i_filename = STD_FILE; FILE* i_file = stdin;
    char* o_filename = STD_FILE; FILE* o_file = stdout;

    int mode = get_exec_mode(argc, argv);

    switch (mode) {
        case V_MODE:
            return version();
        case H_MODE:
            return help();
        case STD_MODE:
            break;
        case I_MODE: {
            i_filename = argv[2];
            break;
        }
        case O_MODE: {
            o_filename = argv[2];
            break;
        }
        case IO_MODE: {
            i_filename = argv[2];
            o_filename = argv[4];
            break;
        }
        case OI_MODE: {
            i_filename = argv[4];
            o_filename = argv[2];
            break;
        }
        default: {
            fprintf(stderr,"unrecognized command line option");
            return ERROR;
        }
    }

    bool input_is_std = (strcmp(i_filename, STD_FILE) == 0);
```

```
260     bool output_is_std = (strcmp(o_filename, STD_FILE) == 0);
261
262     if (!input_is_std) {
263         i_file = fopen(i_filename, "r+");
264         if (!i_file) {
265             fprintf(stderr, "could not open input file\n");
266             return ERROR;
267         }
268     }
269     if (!output_is_std) {
270         o_file = fopen(o_filename, "w+");
271         if (!o_file) {
272             perror("could not open output file\n");
273             if (!input_is_std) fclose(i_file);
274             return ERROR;
275         }
276     }
277
278     sort(i_file, o_file);
279
280     if (!input_is_std) fclose(i_file);
281     if (!output_is_std) fclose(o_file);
282     return SUCCESS;
283 }
```

5.2. vector.h

El **vector.h** contiene la abstracción de la clase vector con sus respectivos atributos y métodos. Esta clase fue creada para manejar el array de números extraído de las líneas de caracteres del archivo de entrada.

```
1
2  #ifndef VECTOR_H
3  #define VECTOR_H
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdbool.h>
9
10 #define ERROR -1
11 #define SUCCESS 0
12 #define INIT_SIZE 4096
13
14 typedef struct vector {
15     int* array;
16     size_t size;
17     size_t max_size;
18 } vector_t;
19
20 int vector_init(vector_t* self) {
21     self->array = calloc(INIT_SIZE, sizeof(int));
22     self->max_size = INIT_SIZE;
```

```
23      self->size = 0;
24      return SUCCESS;
25  }
26
27  bool vector_empty(vector_t* self) {
28      return self->size == 0;
29  }
30
31  void vector_push(vector_t* self, int n) {
32      if (self->size == self->max_size - 1) {
33          self->max_size = self->max_size * 2;
34          self->array = realloc(self->array, self->max_size * sizeof(int));
35      }
36
37      self->array[self->size] = n;
38      self->size++;
39  }
40
41  void vector_clear(vector_t* self) {
42      memset(self->array, 0, self->max_size);
43      self->size = 0;
44  }
45
46  void vector_destroy(vector_t* self) {
47      free(self->array);
48      self->max_size = INIT_SIZE;
49      self->array = NULL;
50      self->size = 0;
51  }
52
53  #endif
```

5.3. mergesort.h

El **mergesort.h** contiene la declaración, de la función en MIPS, de merge_sort().

```
1
2   #ifndef MERGE_SORT_H
3   #define MERGE_SORT_H
4
5   #include <stdlib.h>
6
7   // Receives a pointer to the array to be sorted (p)
8   // and an integer with the size of the array (size).
9   // Sorts the array modifing the original one.
10  extern void merge_sort(int p[], size_t size);
11
12  #endif
```

5.4. mergesort.S

El **mergesort.S** es la función en MIPS que se encarga de ordenar los arrays de enteros que recibe por parámetro. Utiliza un array de memoria dinámica como auxiliar el cual obtiene llamando a la función mymalloc y liberándolo con la función myfree, ambas funciones provistas por la cátedra.

```
1
2  #include <sys/regdef.h>
3  #include "mergesortrec.S"
4  #include "mymalloc.S"
5
6  .text
7  .align 2
8
9  .ent merge_sort
10 .globl merge_sort
11 #-------------------------------MERGE SORT-------------------------------#
12 merge_sort:
13     # a0 -> array pointer
14     # a1 -> array size
15
16     # fp -> stack begin
17     # 40 -> stack size
18     # ra -> retunr addr
19     .frame fp, 40, ra # 16 -> SRA | 8 -> LTA | 16 -> ABA
20
21     # stack frame creation
22     subu sp, sp, 40      # callee stack -> [sp, sp + 40]
23
24     # saved register area (SRA)
25                          # padding in (36 + sp) | callee stack
26     sw ra, 32(sp)        # save ra in (32 + sp) | callee stack
27     sw fp, 28(sp)        # save fp in (28 + sp) | callee stack
28     .cprestore 24        # save gp in (24 + sp) | callee stack
29
30     move fp, sp          # fp = sp
31
32     # fill ABA caller
33                          # a3 reserved area     | caller stack
34                          # a2 reserved area     | caller stack
35     sw a1, 44(fp)        # save a1 in (44 + sp) | caller stack
36     sw a0, 40(fp)        # save a1 in (40 + sp) | caller stack
37
38     # local and temporary area (LTA)
39                          # padding stack[sp + 20]
40     sll a0, a1, 2        # a0 = array size * 4 (bytes)
41     jal mymalloc         # call void* mymalloc(size_t)
42                          # return value stored in v0
43
44     beqz v0, _exit       # if mymalloc failed then v0=NULL and we call exit
       syscall
45
46     sw v0, 16(fp)        # aux_array saved in LTA
```

```
47
48      lw a0, 40(fp)       # retrieve a0 from ABA - a0 = stack[40 + fp]
49      lw a1, 44(fp)       # retrieve a1 from ABA - a1 = stack[44 + fp]
50
51      subu t1, a1, 1      # calculate right index (t1 = size - 1)
52
53      move a0, a0         # redundant (a0 = array pointer (P))
54      move a1, v0         # a1 = v0 = merge aux array pointer (T)
55      move a2, zero       # a2 = left index (L) = 0
56      move a3, t1         # a3 = t1 = right index (R) = size - 1
57
58      # argument building area (ABA)
59      # reserved 16 bytes for callee
60
61      jal _merge_sort_rec
62
63      lw a0, 16(fp)       # load aux_array in a0
64      jal myfree          # free memory requested by aux_array
65
66  #----------------------RETURN-----------------------------#
67  _return_ms:
68      # retreive registers from SRA
69      lw  gp, 24(sp)
70      lw  fp, 28(sp)
71      lw  ra, 32(sp)
72      addiu sp, sp, 40    # increment stack pointer
73      jr  ra              # jump to return address
74  #---------------------------------------------------------#
75
76  #---------------------EXIT_SYS----------------------#
77  _exit:
78      #Write err_msg to stderr
79      li a0, 2            # a0 = stderr fd=2
80      li a1, err_msg      # a1 = err_msg
81      li a2, 15           # a2 = strlen(err_msg)
82      li v0, SYS_write    # Syscall write(2, err_msg, strlen(err_msg)
83      syscall
84
85      # retreive registers from SRA
86      lw  gp, 24(sp)
87      lw  fp, 28(sp)
88      lw  ra, 32(sp)
89      addiu sp, sp, 40    # increment stack pointer
90      li  v0, 10          # Syscall exit = 10
91      syscall
92  #---------------------------------------------------------#
93
94  .end merge_sort
95  #-------------------------------------------------------------------------------#
```

5.5. mergesortrec.S

El **mergesortrec.S** es la función en MIPS que se encarga de hacer los llamados recursivos dividiendo el vector en dos mitades para luego ordenarlas.

```
#include <sys/regdef.h>
#include "merge.S"

.text
.align 2

.ent _merge_sort_rec
.globl _merge_sort_rec
#————————————————————————MERGE-SORT-RECURSIVE————————————————————————#
_merge_sort_rec:
    # a0 -> array pointer (P)
    # a1 -> merge aux array pointer (T)
    # a2 -> left index (L)
    # a3 -> right index (R)

    # fp -> stack begin
    # 40 -> stack size
    # ra -> retunr addr
    .frame fp, 40, ra    # 16 -> SRA | 8 -> LTA | 16 -> ABA

    # stack frame creation
    subu sp, sp, 40      # callee stack -> [sp, sp + 40]

    # saved register area (SRA)
                         # padding in (36 + sp) | callee stack
    sw ra, 32(sp)        # save ra in (32 + sp) | callee stack
    sw fp, 28(sp)        # save fp in (28 + sp) | callee stack
    .cprestore 24        # save gp in (24 + sp) | callee stack

    move fp, sp          # fp = sp

    # fill ABA caller
    sw a3, 52(fp)        # save a3 in (52 + sp) | caller stack
    sw a2, 48(fp)        # save a2 in (48 + sp) | caller stack
    sw a1, 44(fp)        # save a1 in (44 + sp) | caller stack
    sw a0, 40(fp)        # save a0 in (40 + sp) | caller stack

    bge a2, a3, _return_ms_rec # if (L >= R) return
    nop

    #calculates the middle of the array
    subu t0, a3, a2      # t0 <-- R - L
    srl t0, t0, 1        # t0 <-- t0 / 2
    addu t0, t0, a2      # t0 <-- t0 + L
    # t0 <-- L (R - L) / 2 = m

    # local and temporary area (LTA)
    # padding stack[sp + 20]
    sw t0, 16(fp)
```

```
50
51      move a0, a0             # redundant (a0 = array pointer (P))
52      move a1, a1             # redundant (a1 = merge aux array pointer (T))
53      move a2, a2             # redundant (a2 = left index (L))
54      move a3, t0             # a3 = t0 = right index (R) = m
55
56      jal _merge_sort_rec
57
58      lw t0, 16(fp)           # retrieve m
59      addiu t0, t0, 1         # m = m + 1
60
61      lw a3, 52(fp)           # retrive a3
62
63      move a0, a0             # redundant (a0 = array pointer (P))
64      move a1, a1             # redundant (a1 = merge aux array pointer (T))
65      move a2, t0             # a2 = left index (L) = m + 1
66      move a3, a3             # a3 = right index (R)
67
68      jal _merge_sort_rec
69
70      lw a3, 52(fp)           # retrieve a3 = R
71      lw a2, 48(fp)           # retrieve a2 = L
72
73      move a0, a0             # redundant (a0 = array pointer (P))
74      move a1, a1             # redundant (a1 = merge aux array pointer (T))
75      move a2, a2             # redundant (a2 = left index (L))
76      move a3, a3             # redundant (a3 = right index (R))
77
78      jal merge
79
80  #-------------------------------------RETURN-------------------------------------#
81  _return_ms_rec:
82      # retreive registers from SRA
83      lw  gp, 24(sp)
84      lw  fp, 28(sp)
85      lw  ra, 32(sp)
86      addiu sp, sp, 40        # increment stack pointer
87      jr  ra                  # jump to return address
88  #--------------------------------------------------------------------------------#
89
90  .end _merge_sort_rec
91  #--------------------------------------------------------------------------------#
```

5.6. merge.S

El **merge.S** es la función en MIPS que se encarga de ordenar el vector recibido utilizando una referencia a un vector auxiliar en memoria dinámica.

```
1
2  #include <sys/regdef.h>
3
4  .text
5  .align 2
```

```
6
7  .ent merge
8  .globl merge
9  #-------------------------------------MERGE-------------------------------------#
10 merge:
11     # a0 -> array pointer (P)
12     # a1 -> merge aux array pointer (T)
13     # a2 -> left index (L)
14     # a3 -> right index (R)
15
16     # fp -> stack begin
17     # 32 -> stack size
18     # ra -> retunr addr
19     .frame fp, 32, ra  # 16 -> SRA | 16 -> LTA
20
21     #.set noreorder
22     #.cpload t9
23     #.set reorder
24
25     # leaf-function -> ABA is not created
26
27     # stack frame creation
28     subu sp, sp, 32      # callee stack -> [sp, sp + 32]
29
30     # fill ABA caller
31     sw a3, 44(sp)        # save a3 in (44 + sp) | caller stack
32     sw a2, 40(sp)        # save a2 in (40 + sp) | caller stack
33     sw a1, 36(sp)        # save a1 in (36 + sp) | caller stack
34     sw a0, 32(sp)        # save a0 in (32 + sp) | caller stack
35
36     # saved register area (SRA)
37                          # padding in (28 + sp) | callee stack
38     sw ra, 24(sp)        # save ra in (24 + sp) | callee stack
39     sw fp, 20(sp)        # save fp in (20 + sp) | callee stack
40     .cprestore 16        # save gp in (16 + sp) | callee stack
41
42     move fp, sp          # fp = sp
43
44     # temps regs that dont change values during the execution
45     # t0 -> array middle index  (m)
46     # t1 -> left subarray size  (l_size)
47     # t2 -> right subarray size (r_size)
48
49     # t3, t4, t5, t6, t7 -> free to use
50
51     #calculates the middle of the array
52     subu t0, a3, a2      # t0 <-- R - L
53     srl t0, t0, 1        # t0 <-- t0 / 2
54     addu t0, t0, a2      # t0 <-- t0 + L
55     # t0 <-- L (R - L) / 2 = m
56
57     # calculate l_size
58     subu t1, t0, a2      # t1 <-- m - L
59     addiu t1, t1, 1      # t1 <-- t1 + 1 = l_size
```

```
60
61      # calculate r_size
62      subu t2, a3, t0       #t2 <-- R - m (r_size)
63
64      sw zero, 0(fp)        # save i = 0 in (0 + fp)
65      sw zero, 4(fp)        # save j = 0 in (4 + fp)
66      sw a2, 8(fp)          # save k = L in (8 + fp)
67
68  #-------------------------WHILE-LEFT-AND-RIGHT-------------------------------#
69  while_lr:                 # while (i < l_size && j < r_size)
70      lw t3, 0(fp)          # t3 <-- i
71      lw t4, 4(fp)          # t4 <-- j
72
73      bge t3, t1, end_lr    # if (i >= l_size) jump to end_lr
74      bge t4, t2, end_lr    # if (j >= r_size) jump to end_lr
75
76      addu t3, a2, t3       # t3 <-- L + i | index
77      addu t4, t0, t4       # t4 <-- m + j
78      addiu t4, t4, 1       # t4 <-- m + j + 1 | index
79
80      sll t3, t3, 2         # t3 <-- (t3 * 4) | memory pos
81      sll t4, t4, 2         # t4 <-- (t4 * 4) | memory pos
82
83      addu t5, t3, a0       # t5 <-- &p[l + i] | memory pos
84      lw t5, 0(t5)          # t5 <-- p[l + i]  | value
85
86      addu t6, t4, a0       # t6 <-- &p[m + 1 + j] | memory pos
87      lw t6, 0(t6)          # t6 <-- p[m + 1 + j]  | value
88
89      blt t6, t5, else_lr # if (p[m + 1 + j] < p[l + i] ) jump to else_lr
90
91      lw t7, 8(fp)          # t7 <-- k
92      sll t7, t7, 2         # t7 <-- (k*4)
93      addu t7, a1, t7       # t7 <-- t + (k*4) = &t[k] | memory pos
94      sw t5, 0(t7)          # t[k] = p[l + i]
95
96      lw t7, 0(fp)          # t7 <-- i
97      addiu t7, t7, 1       # t7 <-- i + 1
98      sw t7, 0(fp)          # i  <-- i + 1
99
100     la t7, inc_k_lr       # t7 <-- address of inc_k_lr
101     jr t7                 # jump to inc_k_lr
102
103 else_lr:
104
105     lw t7, 8(fp)          # t7 <-- k
106     sll t7, t7, 2         # t7 <-- (k*4)
107     addu t7, a1, t7       # t7 <-- t + (k*4) = &t[k] | memory pos
108     sw t6, 0(t7)          # t[k] = p[m + 1 + j]
109
110     lw t7, 4(fp)          # t7 <-- j
111     addiu t7, t7, 1       # t7 <-- j + 1
112     sw t7, 4(fp)          # j  <-- j + 1
113
```

```
114     # continue to inc_k_lr
115
116 inc_k_lr:
117     lw t7, 8(fp)          # t7 <-- k
118     addiu t7, t7, 1       # t7 <-- k + 1
119     sw t7, 8(fp)          # k  <-- k + 1
120
121     la t7, while_lr       # t7 <-- address of while_lr
122     jr t7                 # jump to while_lr
123
124 end_lr:
125     nop                   # continue to while_l
126
127 #-------------------------------------------------------------------------------#
128
129 #-----------------------------------WHILE LEFT----------------------------------#
130 # copy the remaining elements of left side, if there are any.
131 while_l:                  # while (i < l_size)
132
133     lw t3, 0(fp)          # t3 <-- i
134     bge t3, t1, end_l     # if (i >= l_size) jump to end_l
135
136     addu t3, t3, a2       # t3 <-- L + i
137     sll t3, t3, 2         # t3 <-- (t3 * 4)   | memory pos
138     addu t5, t3, a0       # t5 <-- &p[l + i]  | memory pos
139     lw t5, 0(t5)          # t5 <-- p[l + i]   | value
140
141     lw t7, 8(fp)          # t7 <-- k
142     sll t7, t7, 2         # t7 <-- (k*4)
143     addu t7, a1, t7       # t7 <-- t + (k*4) = &t[k] | mem_pos
144     sw t5, 0(t7)          # t[k] = p[l + i]
145
146     lw t3, 0(fp)          # t3 <-- i
147     addiu t3, t3, 1       # t3 <-- i + 1
148     sw t3, 0(fp)          # i  <-- i + 1
149
150     lw t7, 8(fp)          # t7 <-- k
151     addiu t7, t7, 1       # t7 <-- k + 1
152     sw t7, 8(fp)          # k  <-- k + 1
153
154     la t7, while_l        # t7 <-- address of while_l
155     jr t7                 # jump to while_l
156
157 end_l:
158     nop                   # continue to while_r
159 #-------------------------------------------------------------------------------#
160
161 #-----------------------------------WHILE RIGHT---------------------------------#
162 # copy the remaining elements of right side, if there are any.
163 while_r:                  # while (j < r_size)
164
165     lw t4, 4(fp)          # t4 <-- j
166     bge t4, t2, end_r     # if (j >= r_size) jump to end_r
167
```

```
168     addu t4, t0, t4       # t4 <-- m + j
169     addiu t4, t4, 1       # t4 <-- m + j + 1 | index
170     sll t4, t4, 2         # t4 <-- (t4 * 4) | mem_pos
171     addu t6, t4, a0       # t6 <-- &p[m + j + 1] | mem_pos
172     lw t6, 0(t6)          # t6 <-- p[m + j + 1] | value
173
174     lw t7, 8(fp)          # t7 <-- k
175     sll t7, t7, 2         # t7 <-- (k*4)
176     addu t7, a1, t7       # t7 <-- t + (k*4) = &t[k] | mem_pos
177     sw t6, 0(t7)          # t[k] = p[m + j + 1]
178
179     lw t4, 4(fp)          # t7 <-- j
180     addiu t4, t4, 1       # t7 <-- j + 1
181     sw t4, 4(fp)          # j  <-- j + 1
182
183     lw t7, 8(fp)          # t7 <-- k
184     addiu t7, t7, 1       # t7 <-- k + 1
185     sw t7, 8(fp)          # k  <-- k + 1
186
187     la t7, while_r        # t7 <-- address of while_r
188     jr t7                 # jump to while_r
189
190 end_r:
191     nop                   # continue to for
192
193 #-------------------------------------------------------------------------------#
194
195 #-------------------------------------FOR---------------------------------------#
196     # Copy auxiliary elements on the original array.
197
198     lw t3, 8(fp)          # t3 <-- k
199     move t4, a2           # t4 <-- i = L
200
201 for:
202     ble t3, t4, _return # for (i = l; i < k; i++)
203     sll t5, t4, 2         # t5 <-- (i*4)
204     addu t6, t5, a1       # t6 <-- &t[i] | mem_pos
205     lw t6, 0(t6)          # t6 <-- t[i] | value
206
207     addu t7, t5, a0       # t7 <-- &p[i] | mem_pos
208     sw  t6, 0(t7)         # p[i] = t[i];
209
210     addiu t4, t4, 1       # t4 <-- t4 + 1
211
212     la t7, for            # t7 <-- address of for
213     jr t7                 # jump to for
214 #-------------------------------------------------------------------------------#
215
216 #------------------------------RETURN-------------------------------------------#
217 _return:
218     # retreive registers from SRA
219     lw  gp, 16(sp)
220     lw  fp, 20(sp)
221     lw  ra, 24(sp)
```

```
222     addiu sp, sp, 32     # increment stack pointer
223     jr   ra              # jump to return address
224 #------------------------------------------------------------------------------#
225
226 .end merge
227 #------------------------------------------------------------------------------#
```

6. Código MIPS por el compilador

Para finalizar este informe vamos a mostrar el código MIPS del archivo **vsorter.S** generado por el compilador **GCC**:

```
1   .file 1 "vsorter.c"
2   .section .mdebug.abi32
3   .previous
4   .nan legacy
5   .module fp=xx
6   .module nooddspreg
7   .abicalls
8   .text
9   .align 2
10  .globl vector_init
11  .set nomips16
12  .set nomicromips
13  .ent vector_init
14  .type vector_init, @function
15 vector_init:
16  .frame $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
17  .mask 0xc0000000,-4
18  .fmask 0x00000000,0
19  .set noreorder
20  .cpload $25
21  .set nomacro
22  addiu $sp,$sp,-32
23  sw $31,28($sp)
24  sw $fp,24($sp)
25  move $fp,$sp
26  .cprestore 16
27  sw $4,32($fp)
28  li $5,4          # 0x4
29  li $4,256        # 0x100
30  lw $2,%call16(calloc)($28)
31  move $25,$2
32  .reloc 1f,R_MIPS_JALR,calloc
33 1: jalr $25
34  nop
35
36  lw $28,16($fp)
37  move $3,$2
38  lw $2,32($fp)
39  sw $3,0($2)
40  lw $2,32($fp)
```

```
  li   $3,256        # 0x100
  sw   $3,8($2)
  lw   $2,32($fp)
  sw   $0,4($2)
  move   $2,$0
  move   $sp,$fp
  lw   $31,28($sp)
  lw   $fp,24($sp)
  addiu  $sp,$sp,32
  jr   $31
  nop

  .set   macro
  .set   reorder
  .end   vector_init
  .size  vector_init, .-vector_init
  .align   2
  .globl   vector_empty
  .set   nomips16
  .set   nomicromips
  .ent   vector_empty
  .type  vector_empty, @function
vector_empty:
  .frame   $fp,8,$31    # vars= 0, regs= 1/0, args= 0, gp= 0
  .mask  0x40000000,-4
  .fmask   0x00000000,0
  .set   noreorder
  .set   nomacro
  addiu  $sp,$sp,-8
  sw   $fp,4($sp)
  move   $fp,$sp
  sw   $4,8($fp)
  lw   $2,8($fp)
  lw   $2,4($2)
  sltu   $2,$2,1
  andi   $2,$2,0x00ff
  move   $sp,$fp
  lw   $fp,4($sp)
  addiu  $sp,$sp,8
  jr   $31
  nop

  .set   macro
  .set   reorder
  .end   vector_empty
  .size  vector_empty, .-vector_empty
  .align   2
  .globl   vector_push
  .set   nomips16
  .set   nomicromips
  .ent   vector_push
  .type  vector_push, @function
vector_push:
  .frame   $fp,8,$31    # vars= 0, regs= 1/0, args= 0, gp= 0
```

```
95    .mask 0x40000000,-4
96    .fmask  0x00000000,0
97    .set  noreorder
98    .set  nomacro
99    addiu $sp,$sp,-8
100   sw  $fp,4($sp)
101   move  $fp,$sp
102   sw  $4,8($fp)
103   sw  $5,12($fp)
104   lw  $2,8($fp)
105   lw  $3,0($2)
106   lw  $2,8($fp)
107   lw  $2,4($2)
108   sll $2,$2,2
109   addu  $2,$3,$2
110   lw  $3,12($fp)
111   sw  $3,0($2)
112   lw  $2,8($fp)
113   lw  $2,4($2)
114   addiu $3,$2,1
115   lw  $2,8($fp)
116   sw  $3,4($2)
117   nop
118   move  $sp,$fp
119   lw  $fp,4($sp)
120   addiu $sp,$sp,8
121   jr  $31
122   nop
123
124   .set  macro
125   .set  reorder
126   .end  vector_push
127   .size vector_push, .-vector_push
128   .align  2
129   .globl  vector_clear
130   .set  nomips16
131   .set  nomicromips
132   .ent  vector_clear
133   .type vector_clear, @function
134 vector_clear:
135   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
136   .mask 0xc0000000,-4
137   .fmask  0x00000000,0
138   .set  noreorder
139   .cpload $25
140   .set  nomacro
141   addiu $sp,$sp,-32
142   sw  $31,28($sp)
143   sw  $fp,24($sp)
144   move  $fp,$sp
145   .cprestore  16
146   sw  $4,32($fp)
147   lw  $2,32($fp)
148   lw  $3,0($2)
```

```
149   lw   $2,32($fp)
150   lw   $2,8($2)
151   move   $6,$2
152   move   $5,$0
153   move   $4,$3
154   lw   $2,%call16(memset)($28)
155   move   $25,$2
156   .reloc   1f,R_MIPS_JALR,memset
157 1:   jalr   $25
158   nop
159
160   lw   $28,16($fp)
161   lw   $2,32($fp)
162   sw   $0,4($2)
163   nop
164   move   $sp,$fp
165   lw   $31,28($sp)
166   lw   $fp,24($sp)
167   addiu $sp,$sp,32
168   jr   $31
169   nop
170
171   .set   macro
172   .set   reorder
173   .end   vector_clear
174   .size vector_clear, .-vector_clear
175   .align   2
176   .globl   vector_destroy
177   .set   nomips16
178   .set   nomicromips
179   .ent   vector_destroy
180   .type vector_destroy, @function
181 vector_destroy:
182   .frame   $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
183   .mask 0xc0000000,-4
184   .fmask   0x00000000,0
185   .set   noreorder
186   .cpload $25
187   .set   nomacro
188   addiu $sp,$sp,-32
189   sw   $31,28($sp)
190   sw   $fp,24($sp)
191   move   $fp,$sp
192   .cprestore   16
193   sw   $4,32($fp)
194   lw   $2,32($fp)
195   lw   $2,0($2)
196   move   $4,$2
197   lw   $2,%call16(free)($28)
198   move   $25,$2
199   .reloc   1f,R_MIPS_JALR,free
200 1:   jalr   $25
201   nop
202
```

```
  lw   $28,16($fp)
  lw   $2,32($fp)
  li   $3,256        # 0x100
  sw   $3,8($2)
  lw   $2,32($fp)
  sw   $0,0($2)
  lw   $2,32($fp)
  sw   $0,4($2)
  nop
  move   $sp,$fp
  lw   $31,28($sp)
  lw   $fp,24($sp)
  addiu  $sp,$sp,32
  jr   $31
  nop

  .set   macro
  .set   reorder
  .end   vector_destroy
  .size  vector_destroy, .-vector_destroy
  .rdata
  .align   2
$LC0:
  .ascii   "-h\000"
  .align   2
$LC1:
  .ascii   "--help\000"
  .text
  .align   2
  .globl   is_help_flag
  .set   nomips16
  .set   nomicromips
  .ent   is_help_flag
  .type  is_help_flag, @function
is_help_flag:
  .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
  .mask  0xc0000000,-4
  .fmask   0x00000000,0
  .set   noreorder
  .cpload  $25
  .set   nomacro
  addiu  $sp,$sp,-32
  sw   $31,28($sp)
  sw   $fp,24($sp)
  move   $fp,$sp
  .cprestore   16
  sw   $4,32($fp)
  lw   $2,%got($LC0)($28)
  addiu  $5,$2,%lo($LC0)
  lw   $4,32($fp)
  lw   $2,%call16(strcmp)($28)
  move   $25,$2
  .reloc   1f,R_MIPS_JALR,strcmp
1:   jalr   $25
```

```
257   nop
258
259   lw   $28,16($fp)
260   beq  $2,$0,$L9
261   nop
262
263   lw   $2,%got($LC1)($28)
264   addiu  $5,$2,%lo($LC1)
265   lw   $4,32($fp)
266   lw   $2,%call16(strcmp)($28)
267   move   $25,$2
268   .reloc   1f,R_MIPS_JALR,strcmp
269 1:  jalr   $25
270   nop
271
272   lw   $28,16($fp)
273   bne  $2,$0,$L10
274   nop
275
276 $L9:
277   li   $2,1        # 0x1
278   b $L11
279   nop
280
281 $L10:
282   move   $2,$0
283 $L11:
284   andi   $2,$2,0x1
285   andi   $2,$2,0x00ff
286   move   $sp,$fp
287   lw   $31,28($sp)
288   lw   $fp,24($sp)
289   addiu  $sp,$sp,32
290   jr   $31
291   nop
292
293   .set   macro
294   .set   reorder
295   .end   is_help_flag
296   .size  is_help_flag, .-is_help_flag
297   .rdata
298   .align   2
299 $LC2:
300   .ascii   "-V\000"
301   .align   2
302 $LC3:
303   .ascii   "--version\000"
304   .text
305   .align   2
306   .globl   is_version_flag
307   .set   nomips16
308   .set   nomicromips
309   .ent   is_version_flag
310   .type  is_version_flag, @function
```

```
311 is_version_flag:
312   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
313   .mask 0xc0000000,-4
314   .fmask  0x00000000,0
315   .set  noreorder
316   .cpload $25
317   .set  nomacro
318   addiu $sp,$sp,-32
319   sw  $31,28($sp)
320   sw  $fp,24($sp)
321   move  $fp,$sp
322   .cprestore  16
323   sw  $4,32($fp)
324   lw  $2,%got($LC2)($28)
325   addiu $5,$2,%lo($LC2)
326   lw  $4,32($fp)
327   lw  $2,%call16(strcmp)($28)
328   move  $25,$2
329   .reloc  1f,R_MIPS_JALR,strcmp
330 1:  jalr  $25
331   nop
332
333   lw  $28,16($fp)
334   beq $2,$0,$L14
335   nop
336
337   lw  $2,%got($LC3)($28)
338   addiu $5,$2,%lo($LC3)
339   lw  $4,32($fp)
340   lw  $2,%call16(strcmp)($28)
341   move  $25,$2
342   .reloc  1f,R_MIPS_JALR,strcmp
343 1:  jalr  $25
344   nop
345
346   lw  $28,16($fp)
347   bne $2,$0,$L15
348   nop
349
350 $L14:
351   li  $2,1      # 0x1
352   b $L16
353   nop
354
355 $L15:
356   move  $2,$0
357 $L16:
358   andi  $2,$2,0x1
359   andi  $2,$2,0x00ff
360   move  $sp,$fp
361   lw  $31,28($sp)
362   lw  $fp,24($sp)
363   addiu $sp,$sp,32
364   jr  $31
```

```
365   nop
366
367   .set   macro
368   .set   reorder
369   .end   is_version_flag
370   .size  is_version_flag, .-is_version_flag
371   .rdata
372   .align   2
373 $LC4:
374   .ascii   "-i\000"
375   .align   2
376 $LC5:
377   .ascii   "--input\000"
378   .text
379   .align   2
380   .globl   is_input_flag
381   .set   nomips16
382   .set   nomicromips
383   .ent   is_input_flag
384   .type  is_input_flag, @function
385 is_input_flag:
386   .frame   $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
387   .mask  0xc0000000,-4
388   .fmask   0x00000000,0
389   .set   noreorder
390   .cpload  $25
391   .set   nomacro
392   addiu  $sp,$sp,-32
393   sw   $31,28($sp)
394   sw   $fp,24($sp)
395   move   $fp,$sp
396   .cprestore   16
397   sw   $4,32($fp)
398   lw   $2,%got($LC4)($28)
399   addiu  $5,$2,%lo($LC4)
400   lw   $4,32($fp)
401   lw   $2,%call16(strcmp)($28)
402   move   $25,$2
403   .reloc   1f,R_MIPS_JALR,strcmp
404 1:   jalr   $25
405   nop
406
407   lw   $28,16($fp)
408   beq  $2,$0,$L19
409   nop
410
411   lw   $2,%got($LC5)($28)
412   addiu  $5,$2,%lo($LC5)
413   lw   $4,32($fp)
414   lw   $2,%call16(strcmp)($28)
415   move   $25,$2
416   .reloc   1f,R_MIPS_JALR,strcmp
417 1:   jalr   $25
418   nop
```

```
419
420   lw   $28,16($fp)
421   bne  $2,$0,$L20
422   nop
423
424 $L19:
425   li   $2,1        # 0x1
426   b $L21
427   nop
428
429 $L20:
430   move   $2,$0
431 $L21:
432   andi   $2,$2,0x1
433   andi   $2,$2,0x00ff
434   move   $sp,$fp
435   lw   $31,28($sp)
436   lw   $fp,24($sp)
437   addiu  $sp,$sp,32
438   jr   $31
439   nop
440
441   .set   macro
442   .set   reorder
443   .end   is_input_flag
444   .size  is_input_flag, .-is_input_flag
445   .rdata
446   .align   2
447 $LC6:
448   .ascii   "-o\000"
449   .align   2
450 $LC7:
451   .ascii   "--output\000"
452   .text
453   .align   2
454   .globl   is_output_flag
455   .set   nomips16
456   .set   nomicromips
457   .ent   is_output_flag
458   .type  is_output_flag, @function
459 is_output_flag:
460   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
461   .mask  0xc0000000,-4
462   .fmask   0x00000000,0
463   .set   noreorder
464   .cpload  $25
465   .set   nomacro
466   addiu  $sp,$sp,-32
467   sw   $31,28($sp)
468   sw   $fp,24($sp)
469   move   $fp,$sp
470   .cprestore   16
471   sw   $4,32($fp)
472   lw   $2,%got($LC6)($28)
```

```
473   addiu $5,$2,%lo($LC6)
474   lw  $4,32($fp)
475   lw  $2,%call16(strcmp)($28)
476   move  $25,$2
477   .reloc  1f,R_MIPS_JALR,strcmp
478 1:  jalr  $25
479   nop
480
481   lw  $28,16($fp)
482   beq $2,$0,$L24
483   nop
484
485   lw  $2,%got($LC7)($28)
486   addiu $5,$2,%lo($LC7)
487   lw  $4,32($fp)
488   lw  $2,%call16(strcmp)($28)
489   move  $25,$2
490   .reloc  1f,R_MIPS_JALR,strcmp
491 1:  jalr  $25
492   nop
493
494   lw  $28,16($fp)
495   bne $2,$0,$L25
496   nop
497
498 $L24:
499   li  $2,1      # 0x1
500   b $L26
501   nop
502
503 $L25:
504   move  $2,$0
505 $L26:
506   andi  $2,$2,0x1
507   andi  $2,$2,0x00ff
508   move  $sp,$fp
509   lw  $31,28($sp)
510   lw  $fp,24($sp)
511   addiu $sp,$sp,32
512   jr  $31
513   nop
514
515   .set  macro
516   .set  reorder
517   .end  is_output_flag
518   .size is_output_flag, .-is_output_flag
519   .align  2
520   .globl  get_2args_mode
521   .set  nomips16
522   .set  nomicromips
523   .ent  get_2args_mode
524   .type get_2args_mode, @function
525 get_2args_mode:
526   .frame  $fp,40,$31    # vars= 8, regs= 2/0, args= 16, gp= 8
```

```
527   .mask 0xc0000000,-4
528   .fmask  0x00000000,0
529   .set  noreorder
530   .cpload $25
531   .set  nomacro
532   addiu $sp,$sp,-40
533   sw  $31,36($sp)
534   sw  $fp,32($sp)
535   move  $fp,$sp
536   .cprestore  16
537   sw  $4,40($fp)
538   lw  $2,40($fp)
539   lw  $2,4($2)
540   sw  $2,24($fp)
541   lw  $4,24($fp)
542   lw  $2,%got(is_help_flag)($28)
543   move  $25,$2
544   .reloc  1f,R_MIPS_JALR,is_help_flag
545 1:  jalr  $25
546   nop
547
548   lw  $28,16($fp)
549   beq $2,$0,$L29
550   nop
551
552   li  $2,1      # 0x1
553   b $L30
554   nop
555
556 $L29:
557   lw  $4,24($fp)
558   lw  $2,%got(is_version_flag)($28)
559   move  $25,$2
560   .reloc  1f,R_MIPS_JALR,is_version_flag
561 1:  jalr  $25
562   nop
563
564   lw  $28,16($fp)
565   beq $2,$0,$L31
566   nop
567
568   li  $2,2      # 0x2
569   b $L30
570   nop
571
572 $L31:
573   li  $2,-1     # 0xffffffffffffffff
574 $L30:
575   move  $sp,$fp
576   lw  $31,36($sp)
577   lw  $fp,32($sp)
578   addiu $sp,$sp,40
579   jr  $31
580   nop
```

```
581
582   .set   macro
583   .set   reorder
584   .end   get_2args_mode
585   .size  get_2args_mode, .-get_2args_mode
586   .align   2
587   .globl   get_3args_mode
588   .set   nomips16
589   .set   nomicromips
590   .ent   get_3args_mode
591   .type  get_3args_mode, @function
592 get_3args_mode:
593   .frame   $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
594   .mask 0xc0000000,-4
595   .fmask   0x00000000,0
596   .set   noreorder
597   .cpload $25
598   .set   nomacro
599   addiu $sp,$sp,-40
600   sw   $31,36($sp)
601   sw   $fp,32($sp)
602   move   $fp,$sp
603   .cprestore   16
604   sw   $4,40($fp)
605   lw   $2,40($fp)
606   lw   $2,4($2)
607   sw   $2,24($fp)
608   lw   $4,24($fp)
609   lw   $2,%got(is_input_flag)($28)
610   move   $25,$2
611   .reloc   1f,R_MIPS_JALR,is_input_flag
612 1:   jalr   $25
613   nop
614
615   lw   $28,16($fp)
616   beq $2,$0,$L33
617   nop
618
619   li   $2,3       # 0x3
620   b $L34
621   nop
622
623 $L33:
624   lw   $4,24($fp)
625   lw   $2,%got(is_output_flag)($28)
626   move   $25,$2
627   .reloc   1f,R_MIPS_JALR,is_output_flag
628 1:   jalr   $25
629   nop
630
631   lw   $28,16($fp)
632   beq $2,$0,$L35
633   nop
634
```

```
635   li  $2,4        # 0x4
636   b $L34
637   nop
638
639 $L35:
640   li  $2,-1       # 0xffffffffffffffff
641 $L34:
642   move  $sp,$fp
643   lw  $31,36($sp)
644   lw  $fp,32($sp)
645   addiu $sp,$sp,40
646   jr  $31
647   nop
648
649   .set  macro
650   .set  reorder
651   .end  get_3args_mode
652   .size get_3args_mode, .-get_3args_mode
653   .align  2
654   .globl  get_5args_mode
655   .set  nomips16
656   .set  nomicromips
657   .ent  get_5args_mode
658   .type get_5args_mode, @function
659 get_5args_mode:
660   .frame  $fp,48,$31    # vars= 16, regs= 2/0, args= 16, gp= 8
661   .mask 0xc0000000,-4
662   .fmask  0x00000000,0
663   .set  noreorder
664   .cpload $25
665   .set  nomacro
666   addiu $sp,$sp,-48
667   sw  $31,44($sp)
668   sw  $fp,40($sp)
669   move  $fp,$sp
670   .cprestore  16
671   sw  $4,48($fp)
672   lw  $2,48($fp)
673   lw  $2,4($2)
674   sw  $2,24($fp)
675   lw  $2,48($fp)
676   lw  $2,12($2)
677   sw  $2,28($fp)
678   lw  $4,24($fp)
679   lw  $2,%got(is_input_flag)($28)
680   move  $25,$2
681   .reloc  1f,R_MIPS_JALR,is_input_flag
682 1:  jalr  $25
683   nop
684
685   lw  $28,16($fp)
686   sb  $2,32($fp)
687   lw  $4,28($fp)
688   lw  $2,%got(is_input_flag)($28)
```

```
689   move   $25,$2
690   .reloc   1f,R_MIPS_JALR,is_input_flag
691 1:   jalr   $25
692   nop
693
694   lw   $28,16($fp)
695   sb   $2,33($fp)
696   lw   $4,24($fp)
697   lw   $2,%got(is_output_flag)($28)
698   move   $25,$2
699   .reloc   1f,R_MIPS_JALR,is_output_flag
700 1:   jalr   $25
701   nop
702
703   lw   $28,16($fp)
704   sb   $2,34($fp)
705   lw   $4,28($fp)
706   lw   $2,%got(is_output_flag)($28)
707   move   $25,$2
708   .reloc   1f,R_MIPS_JALR,is_output_flag
709 1:   jalr   $25
710   nop
711
712   lw   $28,16($fp)
713   sb   $2,35($fp)
714   lbu $2,32($fp)
715   beq $2,$0,$L37
716   nop
717
718   lbu $2,35($fp)
719   beq $2,$0,$L37
720   nop
721
722   li   $2,5       # 0x5
723   b $L38
724   nop
725
726 $L37:
727   lbu $2,34($fp)
728   beq $2,$0,$L39
729   nop
730
731   lbu $2,33($fp)
732   beq $2,$0,$L39
733   nop
734
735   li   $2,6       # 0x6
736   b $L38
737   nop
738
739 $L39:
740   li   $2,-1      # 0xffffffffffffffff
741 $L38:
742   move   $sp,$fp
```

```
743   lw   $31,44($sp)
744   lw   $fp,40($sp)
745   addiu $sp,$sp,48
746   jr   $31
747   nop
748
749   .set   macro
750   .set   reorder
751   .end   get_5args_mode
752   .size  get_5args_mode, .-get_5args_mode
753   .align   2
754   .globl   get_exec_mode
755   .set   nomips16
756   .set   nomicromips
757   .ent   get_exec_mode
758   .type  get_exec_mode, @function
759 get_exec_mode:
760   .frame   $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
761   .mask 0xc0000000,-4
762   .fmask   0x00000000,0
763   .set   noreorder
764   .cpload $25
765   .set   nomacro
766   addiu $sp,$sp,-32
767   sw   $31,28($sp)
768   sw   $fp,24($sp)
769   move   $fp,$sp
770   .cprestore   16
771   sw   $4,32($fp)
772   sw   $5,36($fp)
773   lw   $2,32($fp)
774   li   $3,2       # 0x2
775   beq  $2,$3,$L42
776   nop
777
778   slt  $3,$2,3
779   beq  $3,$0,$L43
780   nop
781
782   li   $3,1       # 0x1
783   beq  $2,$3,$L44
784   nop
785
786   b $L41
787   nop
788
789 $L43:
790   li   $3,3       # 0x3
791   beq  $2,$3,$L45
792   nop
793
794   li   $3,5       # 0x5
795   beq  $2,$3,$L46
796   nop
```

```
797
798   b $L41
799   nop
800
801 $L44:
802   li  $2,7        # 0x7
803   b $L47
804   nop
805
806 $L42:
807   lw  $4,36($fp)
808   lw  $2,%got(get_2args_mode)($28)
809   move  $25,$2
810   .reloc  1f,R_MIPS_JALR,get_2args_mode
811 1:  jalr  $25
812   nop
813
814   lw  $28,16($fp)
815   b $L47
816   nop
817
818 $L45:
819   lw  $4,36($fp)
820   lw  $2,%got(get_3args_mode)($28)
821   move  $25,$2
822   .reloc  1f,R_MIPS_JALR,get_3args_mode
823 1:  jalr  $25
824   nop
825
826   lw  $28,16($fp)
827   b $L47
828   nop
829
830 $L46:
831   lw  $4,36($fp)
832   lw  $2,%got(get_5args_mode)($28)
833   move  $25,$2
834   .reloc  1f,R_MIPS_JALR,get_5args_mode
835 1:  jalr  $25
836   nop
837
838   lw  $28,16($fp)
839   b $L47
840   nop
841
842 $L41:
843   li  $2,-1       # 0xffffffffffffffff
844 $L47:
845   move  $sp,$fp
846   lw  $31,28($sp)
847   lw  $fp,24($sp)
848   addiu $sp,$sp,32
849   jr  $31
850   nop
```

```
851
852   .set   macro
853   .set   reorder
854   .end   get_exec_mode
855   .size  get_exec_mode, .-get_exec_mode
856   .align  2
857   .globl  remove_endline
858   .set   nomips16
859   .set   nomicromips
860   .ent   remove_endline
861   .type  remove_endline, @function
862 remove_endline:
863   .frame  $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
864   .mask  0xc0000000,-4
865   .fmask  0x00000000,0
866   .set   noreorder
867   .cpload $25
868   .set   nomacro
869   addiu $sp,$sp,-40
870   sw   $31,36($sp)
871   sw   $fp,32($sp)
872   move   $fp,$sp
873   .cprestore  16
874   sw   $4,40($fp)
875   lw   $4,40($fp)
876   lw   $2,%call16(strlen)($28)
877   move   $25,$2
878   .reloc  1f,R_MIPS_JALR,strlen
879 1:  jalr   $25
880   nop
881
882   lw   $28,16($fp)
883   sw   $2,24($fp)
884   lw   $2,24($fp)
885   addiu $2,$2,-1
886   lw   $3,40($fp)
887   addu   $2,$3,$2
888   lb   $3,0($2)
889   li   $2,10      # 0xa
890   bne  $3,$2,$L50
891   nop
892
893   lw   $2,24($fp)
894   addiu $2,$2,-1
895   lw   $3,40($fp)
896   addu   $2,$3,$2
897   sb   $0,0($2)
898 $L50:
899   nop
900   move   $sp,$fp
901   lw   $31,36($sp)
902   lw   $fp,32($sp)
903   addiu $sp,$sp,40
904   jr   $31
```

```
905   nop
906
907   .set   macro
908   .set   reorder
909   .end   remove_endline
910   .size  remove_endline, .-remove_endline
911   .rdata
912   .align   2
913 $LC8:
914   .ascii   " \000"
915   .text
916   .align   2
917   .globl   parse_vec_buffer
918   .set   nomips16
919   .set   nomicromips
920   .ent   parse_vec_buffer
921   .type  parse_vec_buffer, @function
922 parse_vec_buffer:
923   .frame   $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
924   .mask  0xc0000000,-4
925   .fmask   0x00000000,0
926   .set   noreorder
927   .cpload  $25
928   .set   nomacro
929   addiu  $sp,$sp,-40
930   sw   $31,36($sp)
931   sw   $fp,32($sp)
932   move   $fp,$sp
933   .cprestore   16
934   sw   $4,40($fp)
935   sw   $5,44($fp)
936   lw   $4,44($fp)
937   lw   $2,%got(vector_clear)($28)
938   move   $25,$2
939   .reloc   1f,R_MIPS_JALR,vector_clear
940 1:   jalr   $25
941   nop
942
943   lw   $28,16($fp)
944   lw   $4,40($fp)
945   lw   $2,%got(remove_endline)($28)
946   move   $25,$2
947   .reloc   1f,R_MIPS_JALR,remove_endline
948 1:   jalr   $25
949   nop
950
951   lw   $28,16($fp)
952   lw   $2,%got($LC8)($28)
953   addiu  $5,$2,%lo($LC8)
954   lw   $4,40($fp)
955   lw   $2,%call16(strtok)($28)
956   move   $25,$2
957   .reloc   1f,R_MIPS_JALR,strtok
958 1:   jalr   $25
```

```
959   nop
960
961   lw   $28,16($fp)
962   sw   $2,24($fp)
963   b $L52
964   nop
965
966 $L53:
967   li   $6,10       # 0xa
968   move   $5,$0
969   lw   $4,24($fp)
970   lw   $2,%call16(strtol)($28)
971   move   $25,$2
972   .reloc   1f,R_MIPS_JALR,strtol
973 1:   jalr   $25
974   nop
975
976   lw   $28,16($fp)
977   sw   $2,28($fp)
978   lw   $5,28($fp)
979   lw   $4,44($fp)
980   lw   $2,%got(vector_push)($28)
981   move   $25,$2
982   .reloc   1f,R_MIPS_JALR,vector_push
983 1:   jalr   $25
984   nop
985
986   lw   $28,16($fp)
987   lw   $2,%got($LC8)($28)
988   addiu  $5,$2,%lo($LC8)
989   move   $4,$0
990   lw   $2,%call16(strtok)($28)
991   move   $25,$2
992   .reloc   1f,R_MIPS_JALR,strtok
993 1:   jalr   $25
994   nop
995
996   lw   $28,16($fp)
997   sw   $2,24($fp)
998 $L52:
999   lw   $2,24($fp)
1000  bne  $2,$0,$L53
1001  nop
1002
1003  move   $2,$0
1004  move   $sp,$fp
1005  lw   $31,36($sp)
1006  lw   $fp,32($sp)
1007  addiu  $sp,$sp,40
1008  jr   $31
1009  nop
1010
1011  .set   macro
1012  .set   reorder
```

```
1013   .end  parse_vec_buffer
1014   .size  parse_vec_buffer, .-parse_vec_buffer
1015   .align  2
1016   .globl  read_vector
1017   .set  nomips16
1018   .set  nomicromips
1019   .ent  read_vector
1020   .type  read_vector, @function
1021 read_vector:
1022   .frame  $fp,48,$31    # vars= 16, regs= 2/0, args= 16, gp= 8
1023   .mask 0xc0000000,-4
1024   .fmask  0x00000000,0
1025   .set  noreorder
1026   .cpload $25
1027   .set  nomacro
1028   addiu $sp,$sp,-48
1029   sw  $31,44($sp)
1030   sw  $fp,40($sp)
1031   move  $fp,$sp
1032   .cprestore  16
1033   sw  $4,48($fp)
1034   sw  $5,52($fp)
1035   sw  $0,28($fp)
1036   sw  $0,32($fp)
1037   addiu $3,$fp,32
1038   addiu $2,$fp,28
1039   lw  $6,48($fp)
1040   move  $5,$3
1041   move  $4,$2
1042   lw  $2,%call16(getline)($28)
1043   move  $25,$2
1044   .reloc  1f,R_MIPS_JALR,getline
1045 1:  jalr  $25
1046   nop
1047
1048   lw  $28,16($fp)
1049   sw  $2,24($fp)
1050   lw  $3,24($fp)
1051   li  $2,-1     # 0xffffffffffffffff
1052   beq $3,$2,$L56
1053   nop
1054
1055   lw  $2,24($fp)
1056   bne $2,$0,$L57
1057   nop
1058
1059 $L56:
1060   lw  $3,24($fp)
1061   li  $2,-1     # 0xffffffffffffffff
1062   bne $3,$2,$L58
1063   nop
1064
1065   move  $4,$0
1066   lw  $2,%call16(perror)($28)
```

```
1067   move   $25,$2
1068   .reloc   1f,R_MIPS_JALR,perror
1069 1:   jalr   $25
1070   nop
1071
1072   lw   $28,16($fp)
1073 $L58:
1074   lw   $2,28($fp)
1075   move   $4,$2
1076   lw   $2,%call16(free)($28)
1077   move   $25,$2
1078   .reloc   1f,R_MIPS_JALR,free
1079 1:   jalr   $25
1080   nop
1081
1082   lw   $28,16($fp)
1083   li   $2,-1      # 0xffffffffffffffff
1084   b $L60
1085   nop
1086
1087 $L57:
1088   lw   $2,28($fp)
1089   lw   $5,52($fp)
1090   move   $4,$2
1091   lw   $2,%got(parse_vec_buffer)($28)
1092   move   $25,$2
1093   .reloc   1f,R_MIPS_JALR,parse_vec_buffer
1094 1:   jalr   $25
1095   nop
1096
1097   lw   $28,16($fp)
1098   lw   $2,28($fp)
1099   move   $4,$2
1100   lw   $2,%call16(free)($28)
1101   move   $25,$2
1102   .reloc   1f,R_MIPS_JALR,free
1103 1:   jalr   $25
1104   nop
1105
1106   lw   $28,16($fp)
1107   move   $2,$0
1108 $L60:
1109   move   $sp,$fp
1110   lw   $31,44($sp)
1111   lw   $fp,40($sp)
1112   addiu $sp,$sp,48
1113   jr   $31
1114   nop
1115
1116   .set   macro
1117   .set   reorder
1118   .end   read_vector
1119   .size read_vector, .-read_vector
1120   .rdata
```

```
1121   .align  2
1122 $LC9:
1123   .ascii  "%i \000"
1124   .text
1125   .align  2
1126   .globl  print_sorted_vec
1127   .set  nomips16
1128   .set  nomicromips
1129   .ent  print_sorted_vec
1130   .type  print_sorted_vec, @function
1131 print_sorted_vec:
1132   .frame  $fp,40,$31    # vars= 8, regs= 2/0, args= 16, gp= 8
1133   .mask 0xc0000000,-4
1134   .fmask  0x00000000,0
1135   .set  noreorder
1136   .cpload $25
1137   .set  nomacro
1138   addiu $sp,$sp,-40
1139   sw  $31,36($sp)
1140   sw  $fp,32($sp)
1141   move  $fp,$sp
1142   .cprestore  16
1143   sw  $4,40($fp)
1144   sw  $5,44($fp)
1145   lw  $2,44($fp)
1146   beq $2,$0,$L62
1147   nop
1148
1149   sw  $0,24($fp)
1150   b $L63
1151   nop
1152
1153 $L64:
1154   lw  $2,44($fp)
1155   lw  $3,0($2)
1156   lw  $2,24($fp)
1157   sll $2,$2,2
1158   addu  $2,$3,$2
1159   lw  $2,0($2)
1160   move  $6,$2
1161   lw  $2,%got($LC9)($28)
1162   addiu $5,$2,%lo($LC9)
1163   lw  $4,40($fp)
1164   lw  $2,%call16(fprintf)($28)
1165   move  $25,$2
1166   .reloc  1f,R_MIPS_JALR,fprintf
1167 1:  jalr  $25
1168   nop
1169
1170   lw  $28,16($fp)
1171   lw  $2,24($fp)
1172   addiu $2,$2,1
1173   sw  $2,24($fp)
1174 $L63:
```

```
1175   lw   $2,44($fp)
1176   lw   $3,4($2)
1177   lw   $2,24($fp)
1178   sltu   $2,$2,$3
1179   bne  $2,$0,$L64
1180   nop
1181
1182 $L62:
1183   lw   $5,40($fp)
1184   li   $4,10      # 0xa
1185   lw   $2,%call16(fputc)($28)
1186   move   $25,$2
1187   .reloc   1f,R_MIPS_JALR,fputc
1188 1:   jalr   $25
1189   nop
1190
1191   lw   $28,16($fp)
1192   nop
1193   move   $sp,$fp
1194   lw   $31,36($sp)
1195   lw   $fp,32($sp)
1196   addiu  $sp,$sp,40
1197   jr   $31
1198   nop
1199
1200   .set   macro
1201   .set   reorder
1202   .end   print_sorted_vec
1203   .size  print_sorted_vec, .-print_sorted_vec
1204   .rdata
1205   .align   2
1206 $LC10:
1207   .ascii   "Usage:\000"
1208   .align   2
1209 $LC11:
1210   .ascii   "\011tp1 -h\000"
1211   .align   2
1212 $LC12:
1213   .ascii   "\011tp1 -V\000"
1214   .align   2
1215 $LC13:
1216   .ascii   "\011tp1 -i in_file -o out_file\000"
1217   .align   2
1218 $LC14:
1219   .ascii   "Options:\000"
1220   .align   2
1221 $LC15:
1222   .ascii   "\011-V, --version Print version and quit.\000"
1223   .align   2
1224 $LC16:
1225   .ascii   "\011-h, --help Print this information and quit.\000"
1226   .align   2
1227 $LC17:
1228   .ascii   "\011-i, --input Specify input stream/file, '-' for stdin"
```

```
1229    .ascii  "\000"
1230    .align  2
1231 $LC18:
1232    .ascii  "\011-o, --output Specify output stream/file, '-' for std"
1233    .ascii  "out.\000"
1234    .align  2
1235 $LC19:
1236    .ascii  "Examples:\000"
1237    .align  2
1238 $LC20:
1239    .ascii  "\011tp1 < in.txt > out.txt\000"
1240    .align  2
1241 $LC21:
1242    .ascii  "\011cat in.txt | tp1 -i - > out.txt\000"
1243    .text
1244    .align  2
1245    .globl  help
1246    .set    nomips16
1247    .set    nomicromips
1248    .ent    help
1249    .type   help, @function
1250 help:
1251    .frame  $fp,32,$31      # vars= 0, regs= 2/0, args= 16, gp= 8
1252    .mask   0xc0000000,-4
1253    .fmask  0x00000000,0
1254    .set    noreorder
1255    .cpload $25
1256    .set    nomacro
1257    addiu   $sp,$sp,-32
1258    sw      $31,28($sp)
1259    sw      $fp,24($sp)
1260    move    $fp,$sp
1261    .cprestore  16
1262    lw      $2,%got($LC10)($28)
1263    addiu   $4,$2,%lo($LC10)
1264    lw      $2,%call16(puts)($28)
1265    move    $25,$2
1266    .reloc  1f,R_MIPS_JALR,puts
1267 1:  jalr   $25
1268    nop
1269
1270    lw      $28,16($fp)
1271    lw      $2,%got($LC11)($28)
1272    addiu   $4,$2,%lo($LC11)
1273    lw      $2,%call16(puts)($28)
1274    move    $25,$2
1275    .reloc  1f,R_MIPS_JALR,puts
1276 1:  jalr   $25
1277    nop
1278
1279    lw      $28,16($fp)
1280    lw      $2,%got($LC12)($28)
1281    addiu   $4,$2,%lo($LC12)
1282    lw      $2,%call16(puts)($28)
```

```
1283   move   $25,$2
1284   .reloc   1f,R_MIPS_JALR,puts
1285 1:   jalr   $25
1286   nop
1287
1288   lw   $28,16($fp)
1289   lw   $2,%got($LC13)($28)
1290   addiu  $4,$2,%lo($LC13)
1291   lw   $2,%call16(puts)($28)
1292   move   $25,$2
1293   .reloc   1f,R_MIPS_JALR,puts
1294 1:   jalr   $25
1295   nop
1296
1297   lw   $28,16($fp)
1298   lw   $2,%got($LC14)($28)
1299   addiu  $4,$2,%lo($LC14)
1300   lw   $2,%call16(puts)($28)
1301   move   $25,$2
1302   .reloc   1f,R_MIPS_JALR,puts
1303 1:   jalr   $25
1304   nop
1305
1306   lw   $28,16($fp)
1307   lw   $2,%got($LC15)($28)
1308   addiu  $4,$2,%lo($LC15)
1309   lw   $2,%call16(puts)($28)
1310   move   $25,$2
1311   .reloc   1f,R_MIPS_JALR,puts
1312 1:   jalr   $25
1313   nop
1314
1315   lw   $28,16($fp)
1316   lw   $2,%got($LC16)($28)
1317   addiu  $4,$2,%lo($LC16)
1318   lw   $2,%call16(puts)($28)
1319   move   $25,$2
1320   .reloc   1f,R_MIPS_JALR,puts
1321 1:   jalr   $25
1322   nop
1323
1324   lw   $28,16($fp)
1325   lw   $2,%got($LC17)($28)
1326   addiu  $4,$2,%lo($LC17)
1327   lw   $2,%call16(puts)($28)
1328   move   $25,$2
1329   .reloc   1f,R_MIPS_JALR,puts
1330 1:   jalr   $25
1331   nop
1332
1333   lw   $28,16($fp)
1334   lw   $2,%got($LC18)($28)
1335   addiu  $4,$2,%lo($LC18)
1336   lw   $2,%call16(puts)($28)
```

```
1337   move   $25,$2
1338   .reloc   1f,R_MIPS_JALR,puts
1339 1:   jalr   $25
1340   nop
1341
1342   lw   $28,16($fp)
1343   lw   $2,%got($LC19)($28)
1344   addiu  $4,$2,%lo($LC19)
1345   lw   $2,%call16(puts)($28)
1346   move   $25,$2
1347   .reloc   1f,R_MIPS_JALR,puts
1348 1:   jalr   $25
1349   nop
1350
1351   lw   $28,16($fp)
1352   lw   $2,%got($LC20)($28)
1353   addiu  $4,$2,%lo($LC20)
1354   lw   $2,%call16(puts)($28)
1355   move   $25,$2
1356   .reloc   1f,R_MIPS_JALR,puts
1357 1:   jalr   $25
1358   nop
1359
1360   lw   $28,16($fp)
1361   lw   $2,%got($LC21)($28)
1362   addiu  $4,$2,%lo($LC21)
1363   lw   $2,%call16(puts)($28)
1364   move   $25,$2
1365   .reloc   1f,R_MIPS_JALR,puts
1366 1:   jalr   $25
1367   nop
1368
1369   lw   $28,16($fp)
1370   move   $2,$0
1371   move   $sp,$fp
1372   lw   $31,28($sp)
1373   lw   $fp,24($sp)
1374   addiu  $sp,$sp,32
1375   jr   $31
1376   nop
1377
1378   .set   macro
1379   .set   reorder
1380   .end   help
1381   .size  help, .-help
1382   .rdata
1383   .align   2
1384 $LC22:
1385   .ascii   "vsorter version: 1.0.0\000"
1386   .text
1387   .align   2
1388   .globl   version
1389   .set   nomips16
1390   .set   nomicromips
```

```
1391   .ent   version
1392   .type  version, @function
1393 version:
1394   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
1395   .mask  0xc0000000,-4
1396   .fmask   0x00000000,0
1397   .set   noreorder
1398   .cpload $25
1399   .set   nomacro
1400   addiu $sp,$sp,-32
1401   sw   $31,28($sp)
1402   sw   $fp,24($sp)
1403   move   $fp,$sp
1404   .cprestore   16
1405   lw   $2,%got($LC22)($28)
1406   addiu $4,$2,%lo($LC22)
1407   lw   $2,%call16(puts)($28)
1408   move   $25,$2
1409   .reloc   1f,R_MIPS_JALR,puts
1410 1:   jalr   $25
1411   nop
1412
1413   lw   $28,16($fp)
1414   move   $2,$0
1415   move   $sp,$fp
1416   lw   $31,28($sp)
1417   lw   $fp,24($sp)
1418   addiu $sp,$sp,32
1419   jr   $31
1420   nop
1421
1422   .set   macro
1423   .set   reorder
1424   .end   version
1425   .size  version, .-version
1426   .align   2
1427   .globl   sort
1428   .set   nomips16
1429   .set   nomicromips
1430   .ent   sort
1431   .type  sort, @function
1432 sort:
1433   .frame   $fp,48,$31     # vars= 16, regs= 2/0, args= 16, gp= 8
1434   .mask  0xc0000000,-4
1435   .fmask   0x00000000,0
1436   .set   noreorder
1437   .cpload $25
1438   .set   nomacro
1439   addiu $sp,$sp,-48
1440   sw   $31,44($sp)
1441   sw   $fp,40($sp)
1442   move   $fp,$sp
1443   .cprestore   16
1444   sw   $4,48($fp)
```

```
1445   sw   $5,52($fp)
1446   addiu  $2,$fp,24
1447   move   $4,$2
1448   lw   $2,%got(vector_init)($28)
1449   move   $25,$2
1450   .reloc   1f,R_MIPS_JALR,vector_init
1451 1:   jalr   $25
1452   nop
1453
1454   lw   $28,16($fp)
1455   b $L70
1456   nop
1457
1458 $L72:
1459   addiu  $2,$fp,24
1460   move   $4,$2
1461   lw   $2,%got(vector_empty)($28)
1462   move   $25,$2
1463   .reloc   1f,R_MIPS_JALR,vector_empty
1464 1:   jalr   $25
1465   nop
1466
1467   lw   $28,16($fp)
1468   xori   $2,$2,0x1
1469   andi   $2,$2,0x00ff
1470   beq  $2,$0,$L71
1471   nop
1472
1473   lw   $2,24($fp)
1474   lw   $3,28($fp)
1475   move   $5,$3
1476   move   $4,$2
1477   lw   $2,%call16(merge_sort)($28)
1478   move   $25,$2
1479   .reloc   1f,R_MIPS_JALR,merge_sort
1480 1:   jalr   $25
1481   nop
1482
1483   lw   $28,16($fp)
1484   addiu  $2,$fp,24
1485   move   $5,$2
1486   lw   $4,52($fp)
1487   lw   $2,%got(print_sorted_vec)($28)
1488   move   $25,$2
1489   .reloc   1f,R_MIPS_JALR,print_sorted_vec
1490 1:   jalr   $25
1491   nop
1492
1493   lw   $28,16($fp)
1494   b $L70
1495   nop
1496
1497 $L71:
1498   move   $5,$0
```

```
1499   lw   $4,52($fp)
1500   lw   $2,%got(print_sorted_vec)($28)
1501   move   $25,$2
1502   .reloc   1f,R_MIPS_JALR,print_sorted_vec
1503 1:   jalr   $25
1504   nop
1505
1506   lw   $28,16($fp)
1507 $L70:
1508   addiu  $2,$fp,24
1509   move   $5,$2
1510   lw   $4,48($fp)
1511   lw   $2,%got(read_vector)($28)
1512   move   $25,$2
1513   .reloc   1f,R_MIPS_JALR,read_vector
1514 1:   jalr   $25
1515   nop
1516
1517   lw   $28,16($fp)
1518   beq  $2,$0,$L72
1519   nop
1520
1521   addiu  $2,$fp,24
1522   move   $4,$2
1523   lw   $2,%got(vector_destroy)($28)
1524   move   $25,$2
1525   .reloc   1f,R_MIPS_JALR,vector_destroy
1526 1:   jalr   $25
1527   nop
1528
1529   lw   $28,16($fp)
1530   nop
1531   move   $sp,$fp
1532   lw   $31,44($sp)
1533   lw   $fp,40($sp)
1534   addiu  $sp,$sp,48
1535   jr   $31
1536   nop
1537
1538   .set   macro
1539   .set   reorder
1540   .end   sort
1541   .size  sort, .-sort
1542   .rdata
1543   .align   2
1544 $LC23:
1545   .ascii   "-\000"
1546   .align   2
1547 $LC24:
1548   .ascii   "unrecognized command line option\000"
1549   .align   2
1550 $LC25:
1551   .ascii   "r+\000"
1552   .align   2
```

```
$LC26:
  .ascii  "could not open input file\012\000"
  .align  2
$LC27:
  .ascii  "w+\000"
  .align  2
$LC28:
  .ascii  "could not open output file\012\000"
  .text
  .align  2
  .globl  main
  .set  nomips16
  .set  nomicromips
  .ent  main
  .type  main, @function
main:
  .frame  $fp,56,$31    # vars= 24, regs= 2/0, args= 16, gp= 8
  .mask  0xc0000000,-4
  .fmask  0x00000000,0
  .set  noreorder
  .cpload $25
  .set  nomacro
  addiu $sp,$sp,-56
  sw  $31,52($sp)
  sw  $fp,48($sp)
  move  $fp,$sp
  .cprestore  16
  sw  $4,56($fp)
  sw  $5,60($fp)
  lw  $2,%got($LC23)($28)
  addiu $2,$2,%lo($LC23)
  sw  $2,24($fp)
  lw  $2,%got(stdin)($28)
  lw  $2,0($2)
  sw  $2,28($fp)
  lw  $2,%got($LC23)($28)
  addiu $2,$2,%lo($LC23)
  sw  $2,32($fp)
  lw  $2,%got(stdout)($28)
  lw  $2,0($2)
  sw  $2,36($fp)
  lw  $5,60($fp)
  lw  $4,56($fp)
  lw  $2,%got(get_exec_mode)($28)
  move  $25,$2
  .reloc  1f,R_MIPS_JALR,get_exec_mode
1:  jalr  $25
  nop

  lw  $28,16($fp)
  sw  $2,40($fp)
  lw  $2,40($fp)
  sltu  $2,$2,7
  beq $2,$0,$L74
```

```
1607    nop
1608
1609    lw   $2,40($fp)
1610    sll  $3,$2,2
1611    lw   $2,%got($L76)($28)
1612    addiu  $2,$2,%lo($L76)
1613    addu   $2,$3,$2
1614    lw   $2,0($2)
1615    addu   $2,$2,$28
1616    jr   $2
1617    nop
1618
1619    .rdata
1620    .align   2
1621    .align   2
1622 $L76:
1623    .gpword  $L74
1624    .gpword  $L75
1625    .gpword  $L77
1626    .gpword  $L78
1627    .gpword  $L79
1628    .gpword  $L80
1629    .gpword  $L81
1630    .text
1631 $L77:
1632    lw   $2,%got(version)($28)
1633    move   $25,$2
1634    .reloc   1f,R_MIPS_JALR,version
1635 1:   jalr   $25
1636    nop
1637
1638    lw   $28,16($fp)
1639    b  $L82
1640    nop
1641
1642 $L75:
1643    lw   $2,%got(help)($28)
1644    move   $25,$2
1645    .reloc   1f,R_MIPS_JALR,help
1646 1:   jalr   $25
1647    nop
1648
1649    lw   $28,16($fp)
1650    b  $L82
1651    nop
1652
1653 $L78:
1654    lw   $2,60($fp)
1655    lw   $2,8($2)
1656    sw   $2,24($fp)
1657    b  $L83
1658    nop
1659
1660 $L79:
```

```
1661   lw   $2,60($fp)
1662   lw   $2,8($2)
1663   sw   $2,32($fp)
1664   b $L83
1665   nop
1666
1667 $L80:
1668   lw   $2,60($fp)
1669   lw   $2,8($2)
1670   sw   $2,24($fp)
1671   lw   $2,60($fp)
1672   lw   $2,16($2)
1673   sw   $2,32($fp)
1674   b $L83
1675   nop
1676
1677 $L81:
1678   lw   $2,60($fp)
1679   lw   $2,16($2)
1680   sw   $2,24($fp)
1681   lw   $2,60($fp)
1682   lw   $2,8($2)
1683   sw   $2,32($fp)
1684   b $L83
1685   nop
1686
1687 $L74:
1688   lw   $2,%got(stderr)($28)
1689   lw   $2,0($2)
1690   move   $7,$2
1691   li   $6,32       # 0x20
1692   li   $5,1        # 0x1
1693   lw   $2,%got($LC24)($28)
1694   addiu  $4,$2,%lo($LC24)
1695   lw   $2,%call16(fwrite)($28)
1696   move   $25,$2
1697   .reloc   1f,R_MIPS_JALR,fwrite
1698 1:   jalr   $25
1699   nop
1700
1701   lw   $28,16($fp)
1702   li   $2,-1       # 0xffffffffffffffff
1703   b $L82
1704   nop
1705
1706 $L83:
1707   lw   $2,%got($LC23)($28)
1708   addiu  $5,$2,%lo($LC23)
1709   lw   $4,24($fp)
1710   lw   $2,%call16(strcmp)($28)
1711   move   $25,$2
1712   .reloc   1f,R_MIPS_JALR,strcmp
1713 1:   jalr   $25
1714   nop
```

```
1715
1716    lw   $28,16($fp)
1717    sltu   $2,$2,1
1718    sb   $2,44($fp)
1719    lw   $2,%got($LC23)($28)
1720    addiu  $5,$2,%lo($LC23)
1721    lw   $4,32($fp)
1722    lw   $2,%call16(strcmp)($28)
1723    move   $25,$2
1724    .reloc   1f,R_MIPS_JALR,strcmp
1725 1:   jalr   $25
1726    nop
1727
1728    lw   $28,16($fp)
1729    sltu   $2,$2,1
1730    sb   $2,45($fp)
1731    lbu  $2,44($fp)
1732    xori   $2,$2,0x1
1733    andi   $2,$2,0x00ff
1734    beq  $2,$0,$L84
1735    nop
1736
1737    lw   $2,%got($LC25)($28)
1738    addiu  $5,$2,%lo($LC25)
1739    lw   $4,24($fp)
1740    lw   $2,%call16(fopen)($28)
1741    move   $25,$2
1742    .reloc   1f,R_MIPS_JALR,fopen
1743 1:   jalr   $25
1744    nop
1745
1746    lw   $28,16($fp)
1747    sw   $2,28($fp)
1748    lw   $2,28($fp)
1749    bne  $2,$0,$L84
1750    nop
1751
1752    lw   $2,%got(stderr)($28)
1753    lw   $2,0($2)
1754    move   $7,$2
1755    li   $6,26      # 0x1a
1756    li   $5,1       # 0x1
1757    lw   $2,%got($LC26)($28)
1758    addiu  $4,$2,%lo($LC26)
1759    lw   $2,%call16(fwrite)($28)
1760    move   $25,$2
1761    .reloc   1f,R_MIPS_JALR,fwrite
1762 1:   jalr   $25
1763    nop
1764
1765    lw   $28,16($fp)
1766    li   $2,-1      # 0xffffffffffffffff
1767    b $L82
1768    nop
```

```
1769
1770 $L84:
1771   lbu $2,45($fp)
1772   xori  $2,$2,0x1
1773   andi  $2,$2,0x00ff
1774   beq $2,$0,$L85
1775   nop
1776
1777   lw  $2,%got($LC27)($28)
1778   addiu $5,$2,%lo($LC27)
1779   lw  $4,32($fp)
1780   lw  $2,%call16(fopen)($28)
1781   move  $25,$2
1782   .reloc  1f,R_MIPS_JALR,fopen
1783 1:  jalr  $25
1784   nop
1785
1786   lw  $28,16($fp)
1787   sw  $2,36($fp)
1788   lw  $2,36($fp)
1789   bne $2,$0,$L85
1790   nop
1791
1792   lw  $2,%got(stderr)($28)
1793   lw  $2,0($2)
1794   move  $7,$2
1795   li  $6,27       # 0x1b
1796   li  $5,1        # 0x1
1797   lw  $2,%got($LC28)($28)
1798   addiu $4,$2,%lo($LC28)
1799   lw  $2,%call16(fwrite)($28)
1800   move  $25,$2
1801   .reloc  1f,R_MIPS_JALR,fwrite
1802 1:  jalr  $25
1803   nop
1804
1805   lw  $28,16($fp)
1806   lbu $2,44($fp)
1807   xori  $2,$2,0x1
1808   andi  $2,$2,0x00ff
1809   beq $2,$0,$L86
1810   nop
1811
1812   lw  $4,28($fp)
1813   lw  $2,%call16(fclose)($28)
1814   move  $25,$2
1815   .reloc  1f,R_MIPS_JALR,fclose
1816 1:  jalr  $25
1817   nop
1818
1819   lw  $28,16($fp)
1820 $L86:
1821   li  $2,-1       # 0xffffffffffffffff
1822   b $L82
```

```
1823   nop
1824
1825 $L85:
1826   lw   $5,36($fp)
1827   lw   $4,28($fp)
1828   lw   $2,%got(sort)($28)
1829   move   $25,$2
1830   .reloc   1f,R_MIPS_JALR,sort
1831 1:   jalr   $25
1832   nop
1833
1834   lw   $28,16($fp)
1835   lbu  $2,44($fp)
1836   xori   $2,$2,0x1
1837   andi   $2,$2,0x00ff
1838   beq  $2,$0,$L87
1839   nop
1840
1841   lw   $4,28($fp)
1842   lw   $2,%call16(fclose)($28)
1843   move   $25,$2
1844   .reloc   1f,R_MIPS_JALR,fclose
1845 1:   jalr   $25
1846   nop
1847
1848   lw   $28,16($fp)
1849 $L87:
1850   lbu  $2,45($fp)
1851   xori   $2,$2,0x1
1852   andi   $2,$2,0x00ff
1853   beq  $2,$0,$L88
1854   nop
1855
1856   lw   $4,36($fp)
1857   lw   $2,%call16(fclose)($28)
1858   move   $25,$2
1859   .reloc   1f,R_MIPS_JALR,fclose
1860 1:   jalr   $25
1861   nop
1862
1863   lw   $28,16($fp)
1864 $L88:
1865   move   $2,$0
1866 $L82:
1867   move   $sp,$fp
1868   lw   $31,52($sp)
1869   lw   $fp,48($sp)
1870   addiu  $sp,$sp,56
1871   jr   $31
1872   nop
1873
1874   .set   macro
1875   .set   reorder
1876   .end   main
```

```
1877   .size  main, .-main
1878   .ident  "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
1879
```

7. Conclusión

Pudimos cumplir con la finalidad de este trabajo, escribir en el lenguaje C un programa que permite leer vectores numéricos a partir de archivos escritos en el sistema o bien ingresado como una o varias lineas de texto por entrada estándar. Además, logramos escribir en el lenguaje Assembly para MIPS-32 el código que le permite al programa escrito en C realizar un ordenamiento a dichos vectores mediante el algoritmo de **Mergesort**.

Podemos concluir que a pesar de que escribir código en MIPS 32 resulte complicado, debido a que hay que ser mucho más especifico en lo que refiere al uso de los recursos del sistema y los registros del procesador. Es esta misma complejidad la que nos da la ventaja de tener un control prácticamente absoluto de lo que sucede a nivel bit y por lo tanto es posible generar código con una performance mucho mejor en cuanto a velocidad y memoria. Sin embargo, si el código no es lo suficientemente verboso puede resultar muy complicado de seguir y por lo tanto de encontrar errores (además de que es mucho mas sencillo introducirlos por un descuido) así que por este motivo es preferible escribir código en lenguajes de más alto nivel como C siempre que sea posible y dejar la programación en Assembly para ciertas tareas específicas que requieran la mayor performance posible (en este caso un algoritmo de ordenamiento es un gran ejemplo ya que el tiempo que tarda en realizarlo escala tanto como de grande sea el vector a ordenar, por lo que una buena performance es muy deseable).