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

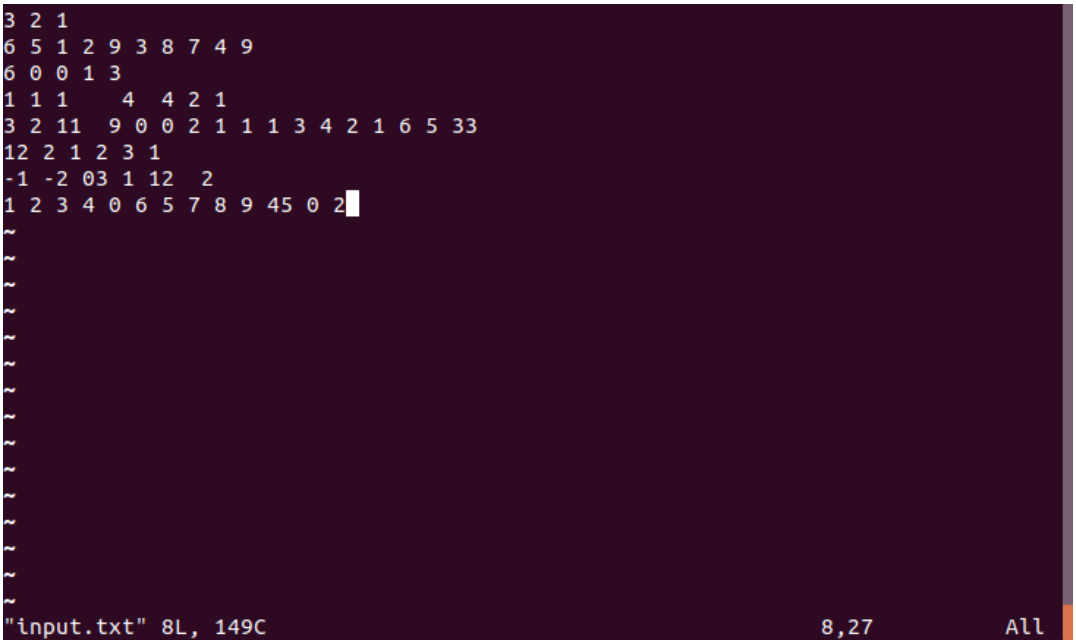
Para realizar las pruebas del algoritmo de ordenamiento Merge Sort, creamos una archivo de texto con la siguiente estructura:

```
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'**.

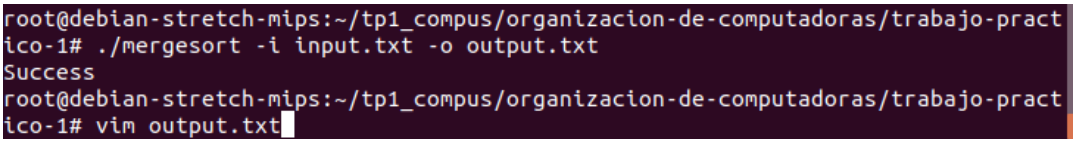
A continuación mostramos el archivo ya creado en el sistema y lo abrimos con el editor de texto en terminal VIM.

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

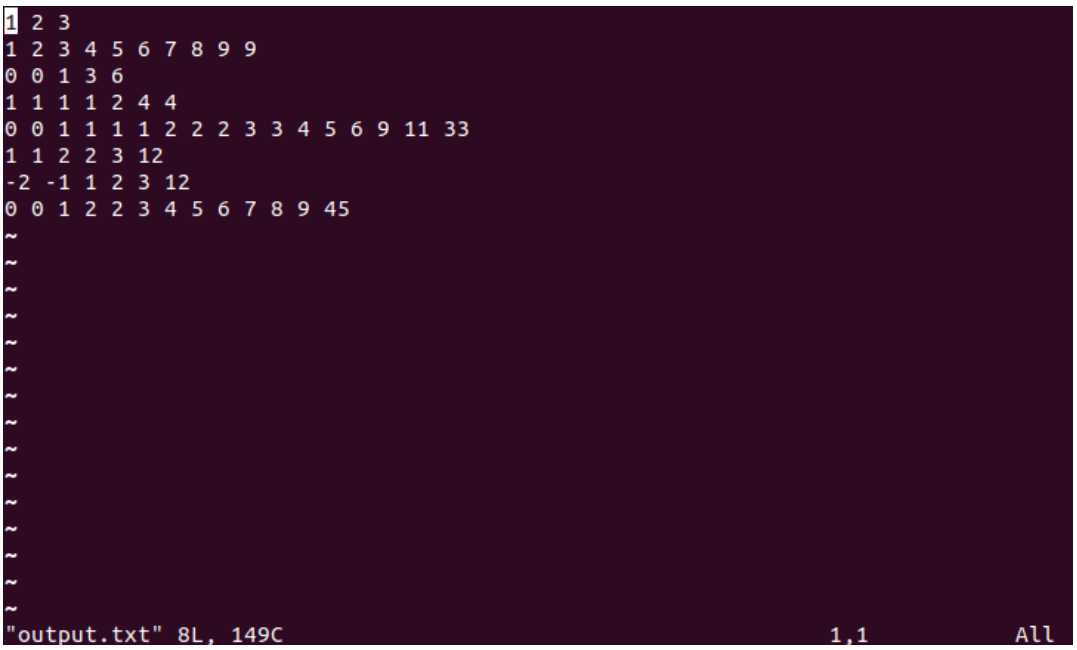


4.3.1. Prueba 1 - Ordenamiento de vectores de un archivo inicial, guardado en otro archivo

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'

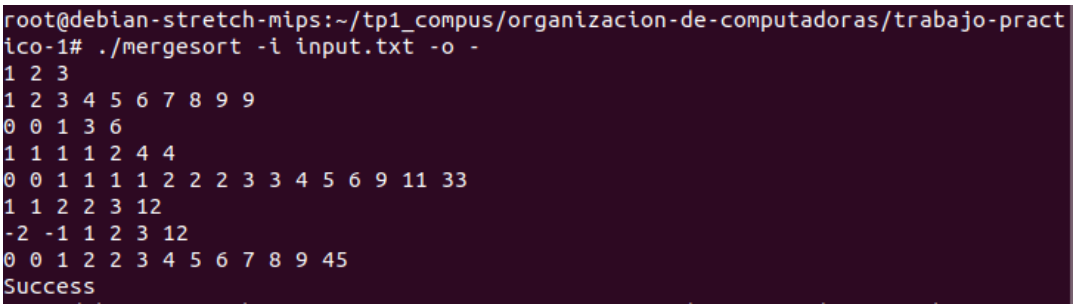


Lo que podemos observar al abrir el archivo 'output.txt' con VIM nos muestra lo siguiente:



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;
```

```c
    size_t size_buff = 0;

    ssize_t bytes_read = getline(&buffer, &size_buff, i_file);

    if (bytes_read == ERROR || bytes_read == 0) {
        if (bytes_read == ERROR) perror(NULL);
        free(buffer);
        return ERROR;
    }

    parse_vec_buffer(buffer, vector);

    free(buffer);
    return SUCCESS;
}

void print_sorted_vec(FILE* o_file, vector_t* vector) {
    if (vector) {
        for (size_t i = 0; i < vector->size; i++) {
            if (i < vector->size - 1)
                fprintf(o_file, "%d ", vector->array[i]);
            else // Last element avoid space
                fprintf(o_file, "%d", vector->array[i]);
        }
    }

    fprintf(o_file,"\n");
}
/*----------------------------------------------------------------*/


/*-----------------------exectuion-modes--------------------------*/
int help() {
    printf("Usage:\n");
    printf("\ttp1 -h\n");
    printf("\ttp1 -V\n");
    printf("\ttp1 -i in_file -o out_file\n");
    printf("Options:\n");
    printf("\t-V, --version Print version and quit.\n");
    printf("\t-h, --help Print this information and quit.\n");
    printf("\t-i, --input Specify input stream/file, '-' for stdin\n");
    printf("\t-o, --output Specify output stream/file, '-' for stdout.\n")
    ;
    printf("Examples:\n");
    printf("\ttp1 < in.txt > out.txt\n");
    printf("\tcat in.txt | tp1 -i - > out.txt\n");
    return SUCCESS;
}

int version() {
    printf("vsorter version: 1.0.0\n");
    return SUCCESS;
}

```

```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  #define MS_A1_OFF 44
7  #define MS_A0_OFF 40
8  #define MS_STACK_SZ 40
9  #define MS_RA_OFF 32
10 #define MS_FP_OFF 28
11 #define MS_GP_OFF 24
12 #define MS_LTA_OFF 16
13
14 .data
15 err_msg: .asciiz "Mymalloc error\n"
16
17 .text
18 .align 2
19
20 .ent merge_sort
21 .globl merge_sort
22 #-------------------------------MERGE SORT-------------------------------#
23 merge_sort:
24     # a0 -> array pointer
25     # a1 -> array size
26
27     # fp -> stack begin
28     # 40 -> stack size
29     # ra -> retunr addr
30     .frame fp, MS_STACK_SZ, ra  # 16 -> SRA | 8 -> LTA | 16 -> ABA
31
32     # stack frame creation
33     subu sp, sp, MS_STACK_SZ    # callee stack -> [sp, sp + 40]
34
35     # saved register area (SRA)
36                                 # padding in (36 + sp) | callee stack
37     sw ra, MS_RA_OFF(sp)        # save ra in (32 + sp) | callee stack
38     sw fp, MS_FP_OFF(sp)        # save fp in (28 + sp) | callee stack
39     .cprestore MS_GP_OFF        # save gp in (MS_GP_OFF + sp) | callee
       stack
40
41     move fp, sp                 # fp = sp
42
43     # fill ABA caller
44                                 # a3 reserved area     | caller stack
45                                 # a2 reserved area     | caller stack
46     sw a1, MS_A1_OFF(fp)        # save a1 in (44 + sp) | caller stack
```

```
47      sw a0, MS_A0_OFF(fp)          # save a1 in (40 + sp) | caller stack
48
49      # local and temporary area (LTA)
50                                    # padding stack[sp + 20]
51      sll a0, a1, 2                 # a0 = array size * 4 (bytes)
52      jal mymalloc                  # call void* mymalloc(size_t)
53                                    # return value stored in v0
54
55      beqz v0, _exit                # if mymalloc failed then v0=NULL and we
        call exit syscall
56
57      sw v0, MS_LTA_OFF(fp)         # aux_array saved in LTA
58
59      lw a0, MS_A0_OFF(fp)          # retrieve a0 from ABA - a0 = stack[40 +
        fp]
60      lw a1, MS_A1_OFF(fp)          # retrieve a1 from ABA - a1 = stack[44 +
        fp]
61
62      subu t1, a1, 1                # calculate right index (t1 = size - 1)
63
64      # a0 = array pointer (P)
65      move a1, v0                   # a1 = v0 = merge aux array pointer (T)
66      move a2, zero                 # a2 = left index (L) = 0
67      move a3, t1                   # a3 = t1 = right index (R) = size - 1
68
69      # argument building area (ABA)
70      # reserved MS_LTA_OFF bytes for callee
71
72      jal _merge_sort_rec
73
74      lw a0, MS_LTA_OFF(fp)         # load aux_array in a0
75      jal myfree                    # free memory requested by aux_array
76
77  #----------------------RETURN-----------------------------#
78  _return_ms:
79      # retreive registers from SRA
80      lw  gp, MS_GP_OFF(sp)
81      lw  fp, MS_FP_OFF(sp)
82      lw  ra, MS_RA_OFF(sp)
83      addiu sp, sp, MS_STACK_SZ     # increment stack pointer
84      jr  ra                        # jump to return address
85  #---------------------------------------------------------#
86
87  #----------------------EXIT_SYS---------------------------#
88  _exit:
89      #Write err_msg to stderr
90      li a0, 2                      # a0 = stderr fd=2
91      li a1, err_msg                # a1 = err_msg
92      li a2, 15                     # a2 = strlen(err_msg)
93      li v0, SYS_write              # Syscall write(2, err_msg, strlen(err_msg
        )
94      syscall
95
96      addiu sp, sp, MS_STACK_SZ     # increment stack pointer
```

```
97      li   a0, -1                    # exit value = -1
98      li   v0, 17                    # Syscall exit2 = 17
99      syscall
100 #-------------------------------------------------------------#
101
102 .end merge_sort
103 #-------------------------------------------------------------------------------#
```

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.

```
1  #include <sys/regdef.h>
2  #include "merge.S"
3
4  #define MSR_A3_OFF 52
5  #define MSR_A2_OFF 48
6  #define MSR_A1_OFF 44
7  #define MSR_A0_OFF 40
8  #define MSR_STACK_SZ 40
9  #define MSR_RA_OFF 32
10 #define MSR_FP_OFF 28
11 #define MSR_GP_OFF 24
12 #define MSR_LTA_OFF 16
13
14 .text
15 .align 2
16
17 .ent _merge_sort_rec
18 .globl _merge_sort_rec
19 #-------------------------MERGE-SORT-RECURSIVE-----------------------------#
20 _merge_sort_rec:
21     # a0 -> array pointer (P)
22     # a1 -> merge aux array pointer (T)
23     # a2 -> left index (L)
24     # a3 -> right index (R)
25
26     # fp -> stack begin
27     # 40 -> stack size
28     # ra -> return addr
29     .frame fp, MSR_STACK_SZ, ra      # 16 -> SRA | 8 -> LTA | 16 -> ABA
30
31     # stack frame creation
32     subu sp, sp, MSR_STACK_SZ        # callee stack -> [sp, sp + 40]
33
34     # saved register area (SRA)
35                                      # padding in (36 + sp) | callee stack
36     sw ra, MSR_RA_OFF(sp)            # save ra in (32 + sp) | callee stack
37     sw fp, MSR_FP_OFF(sp)            # save fp in (28 + sp) | callee stack
38     .cprestore MSR_GP_OFF            # save gp in (24 + sp) | callee stack
39
40     move fp, sp                      # fp = sp
```

```
41
42      # fill ABA caller
43      sw a3, MSR_A3_OFF(fp)           # save a3 in (52 + sp) | caller stack
44      sw a2, MSR_A2_OFF(fp)           # save a2 in (48 + sp) | caller stack
45      sw a1, MSR_A1_OFF(fp)           # save a1 in (44 + sp) | caller stack
46      sw a0, MSR_A0_OFF(fp)           # save a0 in (40 + sp) | caller stack
47
48      bge a2, a3, _return_ms_rec      # if (L >= R) return
49      nop
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      # local and temporary area (LTA)
58      # padding stack[sp + 20]
59      sw t0, MSR_LTA_OFF(fp)
60
61      # a0 = array pointer (P)
62      # a1 = merge aux array pointer (T)
63      # a2 = left index (L)
64      move a3, t0                     # a3 = t0 = right index (R) = m
65
66      jal _merge_sort_rec
67
68      lw a3, MSR_A3_OFF(fp)           # retrieve a3 from (52 + sp) | caller
        stack (a3 = right index (R))
69      lw a2, MSR_A2_OFF(fp)           # retrieve a2 from (48 + sp) | caller
        stack (a2 = left index (L))
70      lw a1, MSR_A1_OFF(fp)           # retrieve a1 from (44 + sp) | caller
        stack (a1 = merge aux array pointer (T))
71      lw a0, MSR_A0_OFF(fp)           # retrieve a0 from (40 + sp) | caller
        stack (a0 = array pointer (P))
72
73      lw t0, MSR_LTA_OFF(fp)          # retrieve m
74      addiu t0, t0, 1                 # m = m + 1
75
76      # a0 = array pointer (P)
77      # a1 = merge aux array pointer (T)
78      move a2, t0                     # a2 = left index (L) = m + 1
79      # a3 = right index (R)
80
81      jal _merge_sort_rec
82
83      lw a3, MSR_A3_OFF(fp)           # retrieve a3 from (52 + sp) | caller
        stack (a3 = right index (R))
84      lw a2, MSR_A2_OFF(fp)           # retrieve a2 from (48 + sp) | caller
        stack (a2 = left index (L))
85      lw a1, MSR_A1_OFF(fp)           # retrieve a1 from (44 + sp) | caller
        stack (a1 = merge aux array pointer (T))
86      lw a0, MSR_A0_OFF(fp)           # retrieve a0 from (40 + sp) | caller
        stack (a0 = array pointer (P))
```

```
87
88      jal merge
89
90  #------------------------RETURN-------------------------------#
91  _return_ms_rec:
92      # retreive registers from SRA
93      lw   gp, MSR_GP_OFF(sp)
94      lw   fp, MSR_FP_OFF(sp)
95      lw   ra, MSR_RA_OFF(sp)
96      addiu sp, sp, MSR_STACK_SZ      # increment stack pointer
97      jr   ra                         # jump to return address
98  #--------------------------------------------------------------#
99
100 .end _merge_sort_rec
101 #-------------------------------------------------------------------------------#
```

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   #define MER_A3_OFF 44
5   #define MER_A2_OFF 40
6   #define MER_A1_OFF 36
7   #define MER_A0_OFF 32
8   #define MER_STACK_SZ 32
9   #define MER_RA_OFF 24
10  #define MER_FP_OFF 20
11  #define MER_GP_OFF 16
12
13  .text
14  .align 2
15
16  .ent merge
17  .globl merge
18  #-------------------------------MERGE-----------------------------------#
19  merge:
20      # a0 -> array pointer (P)
21      # a1 -> merge aux array pointer (T)
22      # a2 -> left index (L)
23      # a3 -> right index (R)
24
25      # fp -> stack begin
26      # 32 -> stack size
27      # ra -> retunr addr
28      .frame fp, MER_STACK_SZ, ra      # 16 -> SRA | 16 -> LTA
29
30      #.set noreorder
31      #.cpload t9
32      #.set reorder
```

```
33
34      # leaf-function -> ABA is not created
35
36      # stack frame creation
37      subu sp, sp, MER_STACK_SZ       # callee stack -> [sp, sp + 32]
38
39      # fill ABA caller
40      sw a3, MER_A3_OFF(sp)           # save a3 in (44 + sp) | caller stack
41      sw a2, MER_A2_OFF(sp)           # save a2 in (40 + sp) | caller stack
42      sw a1, MER_A1_OFF(sp)           # save a1 in (36 + sp) | caller stack
43      sw a0, MER_A0_OFF(sp)           # save a0 in (32 + sp) | caller stack
44
45      # saved register area (SRA)
46                                      # padding in (28 + sp) | callee stack
47      sw ra, MER_RA_OFF(sp)           # save ra in (24 + sp) | callee stack
48      sw fp, MER_FP_OFF(sp)           # save fp in (20 + sp) | callee stack
49      .cprestore MER_GP_OFF           # save gp in (16 + sp) | callee stack
50
51      move fp, sp                     # fp = sp
52
53      # temps regs that dont change values during the execution
54      # t0 -> array middle index  (m)
55      # t1 -> left subarray size  (l_size)
56      # t2 -> right subarray size (r_size)
57
58      # t3, t4, t5, t6, t7 -> free to use
59
60      #calculates the middle of the array
61      subu t0, a3, a2                 # t0 <-- R - L
62      srl t0, t0, 1                   # t0 <-- t0 / 2
63      addu t0, t0, a2                 # t0 <-- t0 + L
64      # t0 <-- L (R - L) / 2 = m
65
66      # calculate l_size
67      subu t1, t0, a2                 # t1 <-- m - L
68      addiu t1, t1, 1                 # t1 <-- t1 + 1 = l_size
69
70      # calculate r_size
71      subu t2, a3, t0                 #t2 <-- R - m (r_size)
72
73      sw zero, 0(fp)                  # save i = 0 in (0 + fp)
74      sw zero, 4(fp)                  # save j = 0 in (4 + fp)
75      sw a2, 8(fp)                    # save k = L in (8 + fp)
76
77  #-------------------WHILE LEFT AND RIGHT---------------------#
78  while_lr:                           # while (i < l_size && j < r_size)
79      lw t3, 0(fp)                    # t3 <-- i
80      lw t4, 4(fp)                    # t4 <-- j
81
82      bge t3, t1, end_lr              # if (i >= l_size) jump to end_lr
83      bge t4, t2, end_lr              # if (j >= r_size) jump to end_lr
84
85      addu t3, a2, t3                 # t3 <-- L + i | index
86      addu t4, t0, t4                 # t4 <-- m + j
```

```
87      addiu t4, t4, 1                 # t4 <-- m + j + 1 | index
88
89      sll t3, t3, 2                   # t3 <-- (t3 * 4) | memory pos
90      sll t4, t4, 2                   # t4 <-- (t4 * 4) | memory pos
91
92      addu t5, t3, a0                 # t5 <-- &p[l + i] | memory pos
93      lw t5, 0(t5)                    # t5 <-- p[l + i]  | value
94
95      addu t6, t4, a0                 # t6 <-- &p[m + 1 + j] | memory pos
96      lw t6, 0(t6)                    # t6 <-- p[m + 1 + j]  | value
97
98      blt t6, t5, else_lr             # if (p[m + 1 + j] < p[l + i] ) jump
        to else_lr
99
100     lw t7, 8(fp)                    # t7 <-- k
101     sll t7, t7, 2                   # t7 <-- (k*4)
102     addu t7, a1, t7                 # t7 <-- t + (k*4) = &t[k] | memory
        pos
103     sw t5, 0(t7)                    # t[k] = p[l + i]
104
105     lw t7, 0(fp)                    # t7 <-- i
106     addiu t7, t7, 1                 # t7 <-- i + 1
107     sw t7, 0(fp)                    # i  <-- i + 1
108
109     la t7, inc_k_lr                 # t7 <-- address of inc_k_lr
110     jr t7                           # jump to inc_k_lr
111
112 else_lr:
113
114     lw t7, 8(fp)                    # t7 <-- k
115     sll t7, t7, 2                   # t7 <-- (k*4)
116     addu t7, a1, t7                 # t7 <-- t + (k*4) = &t[k] | memory
        pos
117     sw t6, 0(t7)                    # t[k] = p[m + 1 + j]
118
119     lw t7, 4(fp)                    # t7 <-- j
120     addiu t7, t7, 1                 # t7 <-- j + 1
121     sw t7, 4(fp)                    # j  <-- j + 1
122
123     # continue to inc_k_lr
124
125 inc_k_lr:
126     lw t7, 8(fp)                    # t7 <-- k
127     addiu t7, t7, 1                 # t7 <-- k + 1
128     sw t7, 8(fp)                    # k  <-- k + 1
129
130     la t7, while_lr                 # t7 <-- address of while_lr
131     jr t7                           # jump to while_lr
132
133 end_lr:
134     nop                             # continue to while_l
135 #-------------------------------------------------------------#
136
137 #-------------------------WHILE LEFT--------------------------#
```

```
# copy the remaining elements of left side, if there are any.
while_l:                            # while (i < l_size)

    lw t3, 0(fp)                    # t3 <-- i
    bge t3, t1, end_l               # if (i >= l_size) jump to end_l

    addu t3, t3, a2                 # t3 <-- L + i
    sll t3, t3, 2                   # t3 <-- (t3 * 4)  | memory pos
    addu t5, t3, a0                 # t5 <-- &p[l + i] | memory pos
    lw t5, 0(t5)                    # t5 <-- p[l + i]  | value

    lw t7, 8(fp)                    # t7 <-- k
    sll t7, t7, 2                   # t7 <-- (k*4)
    addu t7, a1, t7                 # t7 <-- t + (k*4) = &t[k] | mem_pos
    sw t5, 0(t7)                    # t[k] = p[l + i]

    lw t3, 0(fp)                    # t3 <-- i
    addiu t3, t3, 1                 # t3 <-- i + 1
    sw t3, 0(fp)                    # i  <-- i + 1

    lw t7, 8(fp)                    # t7 <-- k
    addiu t7, t7, 1                 # t7 <-- k + 1
    sw t7, 8(fp)                    # k  <-- k + 1

    la t7, while_l                  # t7 <-- address of while_l
    jr t7                           # jump to while_l

end_l:
    nop                             # continue to while_r
#-------------------------------------------------------------#

#-------------------------WHILE RIGHT-------------------------#
# copy the remaining elements of right side, if there are any.
while_r:                            # while (j < r_size)

    lw t4, 4(fp)                    # t4 <-- j
    bge t4, t2, end_r               # if (j >= r_size) jump to end_r

    addu t4, t0, t4                 # t4 <--  m + j
    addiu t4, t4, 1                 # t4 <--  m + j + 1 | index
    sll t4, t4, 2                   # t4 <-- (t4 * 4) | mem_pos
    addu t6, t4, a0                 # t6 <-- &p[m + j + 1] | mem_pos
    lw t6, 0(t6)                    # t6 <-- p[m + j + 1] | value

    lw t7, 8(fp)                    # t7 <-- k
    sll t7, t7, 2                   # t7 <-- (k*4)
    addu t7, a1, t7                 # t7 <-- t + (k*4) = &t[k] | mem_pos
    sw t6, 0(t7)                    # t[k] = p[m + j + 1]

    lw t4, 4(fp)                    # t7 <-- j
    addiu t4, t4, 1                 # t7 <-- j + 1
    sw t4, 4(fp)                    # j  <-- j + 1

    lw t7, 8(fp)                    # t7 <-- k
```

```
192     addiu t7, t7, 1                 # t7 <-- k + 1
193     sw t7, 8(fp)                    # k  <-- k + 1
194
195     la t7, while_r                  # t7 <-- address of while_r
196     jr t7                           # jump to while_r
197
198 end_r:
199     nop                             # continue to for
200 #-------------------------------------------------------------#
201
202 #-------------------------------FOR-------------------------------#
203     # Copy auxiliary elements on the original array.
204
205     lw t3, 8(fp)                    # t3 <-- k
206     move t4, a2                     # t4 <-- i = L
207
208 for:
209     ble t3, t4, _return             # for (i = 1; i < k; i++)
210     sll t5, t4, 2                   # t5 <-- (i*4)
211     addu t6, t5, a1                 # t6 <-- &t[i] | mem_pos
212     lw t6, 0(t6)                    # t6 <-- t[i] | value
213
214     addu t7, t5, a0                 # t7 <-- &p[i] | mem_pos
215     sw  t6, 0(t7)                   # p[i] = t[i];
216
217     addiu t4, t4, 1                 # t4 <-- t4 + 1
218
219     la t7, for                      # t7 <-- address of for
220     jr t7                           # jump to for
221 #-------------------------------------------------------------#
222
223 #-------------------------------RETURN-------------------------------#
224 _return:
225     # retreive registers from SRA
226     lw  gp, MER_GP_OFF(sp)
227     lw  fp, MER_FP_OFF(sp)
228     lw  ra, MER_RA_OFF(sp)
229     addiu sp, sp, MER_STACK_SZ      # increment stack pointer
230     jr  ra                          # jump to return address
231 #-------------------------------------------------------------#
232
233 .end merge
234 #-----------------------------------------------------------------------------#
```

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
```

```
  .nan  legacy
  .module fp=xx
  .module nooddspreg
  .abicalls
  .text
  .align  2
  .globl  vector_init
  .set  nomips16
  .set  nomicromips
  .ent  vector_init
  .type vector_init, @function
vector_init:
  .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)
  li  $5,4      # 0x4
  li  $4,4096     # 0x1000
  lw  $2,%call16(calloc)($28)
  move  $25,$2
  .reloc  1f,R_MIPS_JALR,calloc
1:  jalr  $25
  nop

  lw  $28,16($fp)
  move  $3,$2
  lw  $2,32($fp)
  sw  $3,0($2)
  lw  $2,32($fp)
  li  $3,4096     # 0x1000
  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
```

```
58   .globl  vector_empty
59   .set  nomips16
60   .set  nomicromips
61   .ent  vector_empty
62   .type vector_empty, @function
63 vector_empty:
64   .frame  $fp,8,$31   # vars= 0, regs= 1/0, args= 0, gp= 0
65   .mask 0x40000000,-4
66   .fmask  0x00000000,0
67   .set  noreorder
68   .set  nomacro
69   addiu $sp,$sp,-8
70   sw  $fp,4($sp)
71   move  $fp,$sp
72   sw  $4,8($fp)
73   lw  $2,8($fp)
74   lw  $2,4($2)
75   sltu  $2,$2,1
76   andi  $2,$2,0x00ff
77   move  $sp,$fp
78   lw  $fp,4($sp)
79   addiu $sp,$sp,8
80   jr  $31
81   nop
82
83   .set  macro
84   .set  reorder
85   .end  vector_empty
86   .size vector_empty, .-vector_empty
87   .align  2
88   .globl  vector_push
89   .set  nomips16
90   .set  nomicromips
91   .ent  vector_push
92   .type vector_push, @function
93 vector_push:
94   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
95   .mask 0xc0000000,-4
96   .fmask  0x00000000,0
97   .set  noreorder
98   .cpload $25
99   .set  nomacro
100  addiu $sp,$sp,-32
101  sw  $31,28($sp)
102  sw  $fp,24($sp)
103  move  $fp,$sp
104  .cprestore  16
105  sw  $4,32($fp)
106  sw  $5,36($fp)
107  lw  $2,32($fp)
108  lw  $3,4($2)
109  lw  $2,32($fp)
110  lw  $2,8($2)
111  addiu $2,$2,-1
```

```
112   bne  $3,$2,$L6
113   nop
114
115   lw   $2,32($fp)
116   lw   $2,8($2)
117   sll  $3,$2,1
118   lw   $2,32($fp)
119   sw   $3,8($2)
120   lw   $2,32($fp)
121   lw   $3,0($2)
122   lw   $2,32($fp)
123   lw   $2,8($2)
124   sll  $2,$2,2
125   move   $5,$2
126   move   $4,$3
127   lw   $2,%call16(realloc)($28)
128   move   $25,$2
129   .reloc   1f,R_MIPS_JALR,realloc
130 1:   jalr   $25
131   nop
132
133   lw   $28,16($fp)
134   move   $3,$2
135   lw   $2,32($fp)
136   sw   $3,0($2)
137 $L6:
138   lw   $2,32($fp)
139   lw   $3,0($2)
140   lw   $2,32($fp)
141   lw   $2,4($2)
142   sll  $2,$2,2
143   addu   $2,$3,$2
144   lw   $3,36($fp)
145   sw   $3,0($2)
146   lw   $2,32($fp)
147   lw   $2,4($2)
148   addiu  $3,$2,1
149   lw   $2,32($fp)
150   sw   $3,4($2)
151   nop
152   move   $sp,$fp
153   lw   $31,28($sp)
154   lw   $fp,24($sp)
155   addiu  $sp,$sp,32
156   jr   $31
157   nop
158
159   .set   macro
160   .set   reorder
161   .end   vector_push
162   .size  vector_push, .-vector_push
163   .align   2
164   .globl   vector_clear
165   .set   nomips16
```

```
166   .set   nomicromips
167   .ent   vector_clear
168   .type  vector_clear, @function
169 vector_clear:
170   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
171   .mask 0xc0000000,-4
172   .fmask   0x00000000,0
173   .set   noreorder
174   .cpload $25
175   .set   nomacro
176   addiu $sp,$sp,-32
177   sw   $31,28($sp)
178   sw   $fp,24($sp)
179   move   $fp,$sp
180   .cprestore   16
181   sw   $4,32($fp)
182   lw   $2,32($fp)
183   lw   $3,0($2)
184   lw   $2,32($fp)
185   lw   $2,8($2)
186   move   $6,$2
187   move   $5,$0
188   move   $4,$3
189   lw   $2,%call16(memset)($28)
190   move   $25,$2
191   .reloc   1f,R_MIPS_JALR,memset
192 1:   jalr   $25
193   nop
194
195   lw   $28,16($fp)
196   lw   $2,32($fp)
197   sw   $0,4($2)
198   nop
199   move   $sp,$fp
200   lw   $31,28($sp)
201   lw   $fp,24($sp)
202   addiu $sp,$sp,32
203   jr   $31
204   nop
205
206   .set   macro
207   .set   reorder
208   .end   vector_clear
209   .size  vector_clear, .-vector_clear
210   .align   2
211   .globl   vector_destroy
212   .set   nomips16
213   .set   nomicromips
214   .ent   vector_destroy
215   .type  vector_destroy, @function
216 vector_destroy:
217   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
218   .mask 0xc0000000,-4
219   .fmask   0x00000000,0
```

```
220   .set   noreorder
221   .cpload $25
222   .set   nomacro
223   addiu $sp,$sp,-32
224   sw   $31,28($sp)
225   sw   $fp,24($sp)
226   move   $fp,$sp
227   .cprestore   16
228   sw   $4,32($fp)
229   lw   $2,32($fp)
230   lw   $2,0($2)
231   move   $4,$2
232   lw   $2,%call16(free)($28)
233   move   $25,$2
234   .reloc   1f,R_MIPS_JALR,free
235 1:   jalr   $25
236   nop
237
238   lw   $28,16($fp)
239   lw   $2,32($fp)
240   li   $3,4096      # 0x1000
241   sw   $3,8($2)
242   lw   $2,32($fp)
243   sw   $0,0($2)
244   lw   $2,32($fp)
245   sw   $0,4($2)
246   nop
247   move   $sp,$fp
248   lw   $31,28($sp)
249   lw   $fp,24($sp)
250   addiu $sp,$sp,32
251   jr   $31
252   nop
253
254   .set   macro
255   .set   reorder
256   .end   vector_destroy
257   .size vector_destroy, .-vector_destroy
258   .rdata
259   .align   2
260 $LC0:
261   .ascii   "-h\000"
262   .align   2
263 $LC1:
264   .ascii   "--help\000"
265   .text
266   .align   2
267   .globl   is_help_flag
268   .set   nomips16
269   .set   nomicromips
270   .ent   is_help_flag
271   .type is_help_flag, @function
272 is_help_flag:
273   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
```

```
274   .mask 0xc0000000,-4
275   .fmask  0x00000000,0
276   .set  noreorder
277   .cpload $25
278   .set  nomacro
279   addiu $sp,$sp,-32
280   sw  $31,28($sp)
281   sw  $fp,24($sp)
282   move  $fp,$sp
283   .cprestore  16
284   sw  $4,32($fp)
285   lw  $2,%got($LC0)($28)
286   addiu $5,$2,%lo($LC0)
287   lw  $4,32($fp)
288   lw  $2,%call16(strcmp)($28)
289   move  $25,$2
290   .reloc  1f,R_MIPS_JALR,strcmp
291 1:  jalr  $25
292   nop
293
294   lw  $28,16($fp)
295   beq $2,$0,$L10
296   nop
297
298   lw  $2,%got($LC1)($28)
299   addiu $5,$2,%lo($LC1)
300   lw  $4,32($fp)
301   lw  $2,%call16(strcmp)($28)
302   move  $25,$2
303   .reloc  1f,R_MIPS_JALR,strcmp
304 1:  jalr  $25
305   nop
306
307   lw  $28,16($fp)
308   bne $2,$0,$L11
309   nop
310
311 $L10:
312   li  $2,1      # 0x1
313   b $L12
314   nop
315
316 $L11:
317   move  $2,$0
318 $L12:
319   andi  $2,$2,0x1
320   andi  $2,$2,0x00ff
321   move  $sp,$fp
322   lw  $31,28($sp)
323   lw  $fp,24($sp)
324   addiu $sp,$sp,32
325   jr  $31
326   nop
327
```

```
328   .set   macro
329   .set   reorder
330   .end   is_help_flag
331   .size  is_help_flag, .-is_help_flag
332   .rdata
333   .align   2
334 $LC2:
335   .ascii   "-V\000"
336   .align   2
337 $LC3:
338   .ascii   "--version\000"
339   .text
340   .align   2
341   .globl   is_version_flag
342   .set   nomips16
343   .set   nomicromips
344   .ent   is_version_flag
345   .type  is_version_flag, @function
346 is_version_flag:
347   .frame   $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
348   .mask  0xc0000000,-4
349   .fmask   0x00000000,0
350   .set   noreorder
351   .cpload  $25
352   .set   nomacro
353   addiu  $sp,$sp,-32
354   sw   $31,28($sp)
355   sw   $fp,24($sp)
356   move   $fp,$sp
357   .cprestore   16
358   sw   $4,32($fp)
359   lw   $2,%got($LC2)($28)
360   addiu  $5,$2,%lo($LC2)
361   lw   $4,32($fp)
362   lw   $2,%call16(strcmp)($28)
363   move   $25,$2
364   .reloc   1f,R_MIPS_JALR,strcmp
365 1:   jalr   $25
366   nop
367
368   lw   $28,16($fp)
369   beq  $2,$0,$L15
370   nop
371
372   lw   $2,%got($LC3)($28)
373   addiu  $5,$2,%lo($LC3)
374   lw   $4,32($fp)
375   lw   $2,%call16(strcmp)($28)
376   move   $25,$2
377   .reloc   1f,R_MIPS_JALR,strcmp
378 1:   jalr   $25
379   nop
380
381   lw   $28,16($fp)
```

```
382   bne $2,$0,$L16
383   nop
384
385 $L15:
386   li  $2,1        # 0x1
387   b $L17
388   nop
389
390 $L16:
391   move  $2,$0
392 $L17:
393   andi  $2,$2,0x1
394   andi  $2,$2,0x00ff
395   move  $sp,$fp
396   lw  $31,28($sp)
397   lw  $fp,24($sp)
398   addiu $sp,$sp,32
399   jr  $31
400   nop
401
402   .set  macro
403   .set  reorder
404   .end  is_version_flag
405   .size is_version_flag, .-is_version_flag
406   .rdata
407   .align  2
408 $LC4:
409   .ascii  "-i\000"
410   .align  2
411 $LC5:
412   .ascii  "--input\000"
413   .text
414   .align  2
415   .globl  is_input_flag
416   .set  nomips16
417   .set  nomicromips
418   .ent  is_input_flag
419   .type is_input_flag, @function
420 is_input_flag:
421   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
422   .mask 0xc0000000,-4
423   .fmask  0x00000000,0
424   .set  noreorder
425   .cpload $25
426   .set  nomacro
427   addiu $sp,$sp,-32
428   sw  $31,28($sp)
429   sw  $fp,24($sp)
430   move  $fp,$sp
431   .cprestore  16
432   sw  $4,32($fp)
433   lw  $2,%got($LC4)($28)
434   addiu $5,$2,%lo($LC4)
435   lw  $4,32($fp)
```

```
436   lw   $2,%call16(strcmp)($28)
437   move   $25,$2
438   .reloc   1f,R_MIPS_JALR,strcmp
439 1:   jalr   $25
440   nop
441
442   lw   $28,16($fp)
443   beq  $2,$0,$L20
444   nop
445
446   lw   $2,%got($LC5)($28)
447   addiu  $5,$2,%lo($LC5)
448   lw   $4,32($fp)
449   lw   $2,%call16(strcmp)($28)
450   move   $25,$2
451   .reloc   1f,R_MIPS_JALR,strcmp
452 1:   jalr   $25
453   nop
454
455   lw   $28,16($fp)
456   bne  $2,$0,$L21
457   nop
458
459 $L20:
460   li   $2,1        # 0x1
461   b $L22
462   nop
463
464 $L21:
465   move   $2,$0
466 $L22:
467   andi   $2,$2,0x1
468   andi   $2,$2,0x00ff
469   move   $sp,$fp
470   lw   $31,28($sp)
471   lw   $fp,24($sp)
472   addiu  $sp,$sp,32
473   jr   $31
474   nop
475
476   .set   macro
477   .set   reorder
478   .end   is_input_flag
479   .size  is_input_flag, .-is_input_flag
480   .rdata
481   .align   2
482 $LC6:
483   .ascii   "-o\000"
484   .align   2
485 $LC7:
486   .ascii   "--output\000"
487   .text
488   .align   2
489   .globl   is_output_flag
```

```
490   .set   nomips16
491   .set   nomicromips
492   .ent   is_output_flag
493   .type is_output_flag, @function
494 is_output_flag:
495   .frame   $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
496   .mask 0xc0000000,-4
497   .fmask   0x00000000,0
498   .set   noreorder
499   .cpload $25
500   .set   nomacro
501   addiu $sp,$sp,-32
502   sw   $31,28($sp)
503   sw   $fp,24($sp)
504   move   $fp,$sp
505   .cprestore   16
506   sw   $4,32($fp)
507   lw   $2,%got($LC6)($28)
508   addiu $5,$2,%lo($LC6)
509   lw   $4,32($fp)
510   lw   $2,%call16(strcmp)($28)
511   move   $25,$2
512   .reloc   1f,R_MIPS_JALR,strcmp
513 1:   jalr   $25
514   nop
515
516   lw   $28,16($fp)
517   beq $2,$0,$L25
518   nop
519
520   lw   $2,%got($LC7)($28)
521   addiu $5,$2,%lo($LC7)
522   lw   $4,32($fp)
523   lw   $2,%call16(strcmp)($28)
524   move   $25,$2
525   .reloc   1f,R_MIPS_JALR,strcmp
526 1:   jalr   $25
527   nop
528
529   lw   $28,16($fp)
530   bne $2,$0,$L26
531   nop
532
533 $L25:
534   li   $2,1        # 0x1
535   b $L27
536   nop
537
538 $L26:
539   move   $2,$0
540 $L27:
541   andi   $2,$2,0x1
542   andi   $2,$2,0x00ff
543   move   $sp,$fp
```

```
544   lw   $31,28($sp)
545   lw   $fp,24($sp)
546   addiu $sp,$sp,32
547   jr   $31
548   nop
549
550   .set   macro
551   .set   reorder
552   .end   is_output_flag
553   .size  is_output_flag, .-is_output_flag
554   .align   2
555   .globl   get_2args_mode
556   .set   nomips16
557   .set   nomicromips
558   .ent   get_2args_mode
559   .type  get_2args_mode, @function
560 get_2args_mode:
561   .frame   $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
562   .mask 0xc0000000,-4
563   .fmask   0x00000000,0
564   .set   noreorder
565   .cpload $25
566   .set   nomacro
567   addiu $sp,$sp,-40
568   sw   $31,36($sp)
569   sw   $fp,32($sp)
570   move   $fp,$sp
571   .cprestore   16
572   sw   $4,40($fp)
573   lw   $2,40($fp)
574   lw   $2,4($2)
575   sw   $2,24($fp)
576   lw   $4,24($fp)
577   lw   $2,%got(is_help_flag)($28)
578   move   $25,$2
579   .reloc   1f,R_MIPS_JALR,is_help_flag
580 1:   jalr   $25
581   nop
582
583   lw   $28,16($fp)
584   beq $2,$0,$L30
585   nop
586
587   li   $2,1       # 0x1
588   b $L31
589   nop
590
591 $L30:
592   lw   $4,24($fp)
593   lw   $2,%got(is_version_flag)($28)
594   move   $25,$2
595   .reloc   1f,R_MIPS_JALR,is_version_flag
596 1:   jalr   $25
597   nop
```

```
598
599   lw   $28,16($fp)
600   beq  $2,$0,$L32
601   nop
602
603   li   $2,2        # 0x2
604   b $L31
605   nop
606
607 $L32:
608   li   $2,-1       # 0xffffffffffffffff
609 $L31:
610   move   $sp,$fp
611   lw   $31,36($sp)
612   lw   $fp,32($sp)
613   addiu  $sp,$sp,40
614   jr   $31
615   nop
616
617   .set   macro
618   .set   reorder
619   .end   get_2args_mode
620   .size  get_2args_mode, .-get_2args_mode
621   .align   2
622   .globl   get_3args_mode
623   .set   nomips16
624   .set   nomicromips
625   .ent   get_3args_mode
626   .type  get_3args_mode, @function
627 get_3args_mode:
628   .frame   $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
629   .mask  0xc0000000,-4
630   .fmask   0x00000000,0
631   .set   noreorder
632   .cpload  $25
633   .set   nomacro
634   addiu  $sp,$sp,-40
635   sw   $31,36($sp)
636   sw   $fp,32($sp)
637   move   $fp,$sp
638   .cprestore   16
639   sw   $4,40($fp)
640   lw   $2,40($fp)
641   lw   $2,4($2)
642   sw   $2,24($fp)
643   lw   $4,24($fp)
644   lw   $2,%got(is_input_flag)($28)
645   move   $25,$2
646   .reloc   1f,R_MIPS_JALR,is_input_flag
647 1:   jalr   $25
648   nop
649
650   lw   $28,16($fp)
651   beq  $2,$0,$L34
```

```
652   nop
653
654   li   $2,3        # 0x3
655   b $L35
656   nop
657
658 $L34:
659   lw   $4,24($fp)
660   lw   $2,%got(is_output_flag)($28)
661   move   $25,$2
662   .reloc   1f,R_MIPS_JALR,is_output_flag
663 1:   jalr   $25
664   nop
665
666   lw   $28,16($fp)
667   beq  $2,$0,$L36
668   nop
669
670   li   $2,4        # 0x4
671   b $L35
672   nop
673
674 $L36:
675   li   $2,-1       # 0xffffffffffffffff
676 $L35:
677   move   $sp,$fp
678   lw   $31,36($sp)
679   lw   $fp,32($sp)
680   addiu  $sp,$sp,40
681   jr   $31
682   nop
683
684   .set   macro
685   .set   reorder
686   .end   get_3args_mode
687   .size  get_3args_mode, .-get_3args_mode
688   .align   2
689   .globl   get_5args_mode
690   .set   nomips16
691   .set   nomicromips
692   .ent   get_5args_mode
693   .type  get_5args_mode, @function
694 get_5args_mode:
695   .frame   $fp,48,$31     # vars= 16, regs= 2/0, args= 16, gp= 8
696   .mask  0xc0000000,-4
697   .fmask   0x00000000,0
698   .set   noreorder
699   .cpload  $25
700   .set   nomacro
701   addiu  $sp,$sp,-48
702   sw   $31,44($sp)
703   sw   $fp,40($sp)
704   move   $fp,$sp
705   .cprestore   16
```

```
706   sw   $4,48($fp)
707   lw   $2,48($fp)
708   lw   $2,4($2)
709   sw   $2,24($fp)
710   lw   $2,48($fp)
711   lw   $2,12($2)
712   sw   $2,28($fp)
713   lw   $4,24($fp)
714   lw   $2,%got(is_input_flag)($28)
715   move   $25,$2
716   .reloc   1f,R_MIPS_JALR,is_input_flag
717 1:   jalr   $25
718   nop
719
720   lw   $28,16($fp)
721   sb   $2,32($fp)
722   lw   $4,28($fp)
723   lw   $2,%got(is_input_flag)($28)
724   move   $25,$2
725   .reloc   1f,R_MIPS_JALR,is_input_flag
726 1:   jalr   $25
727   nop
728
729   lw   $28,16($fp)
730   sb   $2,33($fp)
731   lw   $4,24($fp)
732   lw   $2,%got(is_output_flag)($28)
733   move   $25,$2
734   .reloc   1f,R_MIPS_JALR,is_output_flag
735 1:   jalr   $25
736   nop
737
738   lw   $28,16($fp)
739   sb   $2,34($fp)
740   lw   $4,28($fp)
741   lw   $2,%got(is_output_flag)($28)
742   move   $25,$2
743   .reloc   1f,R_MIPS_JALR,is_output_flag
744 1:   jalr   $25
745   nop
746
747   lw   $28,16($fp)
748   sb   $2,35($fp)
749   lbu  $2,32($fp)
750   beq  $2,$0,$L38
751   nop
752
753   lbu  $2,35($fp)
754   beq  $2,$0,$L38
755   nop
756
757   li   $2,5       # 0x5
758   b $L39
759   nop
```

```
760
761 $L38:
762   lbu $2,34($fp)
763   beq $2,$0,$L40
764   nop
765
766   lbu $2,33($fp)
767   beq $2,$0,$L40
768   nop
769
770   li  $2,6        # 0x6
771   b $L39
772   nop
773
774 $L40:
775   li  $2,-1       # 0xffffffffffffffff
776 $L39:
777   move  $sp,$fp
778   lw  $31,44($sp)
779   lw  $fp,40($sp)
780   addiu $sp,$sp,48
781   jr  $31
782   nop
783
784   .set  macro
785   .set  reorder
786   .end  get_5args_mode
787   .size get_5args_mode, .-get_5args_mode
788   .align  2
789   .globl  get_exec_mode
790   .set  nomips16
791   .set  nomicromips
792   .ent  get_exec_mode
793   .type get_exec_mode, @function
794 get_exec_mode:
795   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
796   .mask 0xc0000000,-4
797   .fmask  0x00000000,0
798   .set  noreorder
799   .cpload $25
800   .set  nomacro
801   addiu $sp,$sp,-32
802   sw  $31,28($sp)
803   sw  $fp,24($sp)
804   move  $fp,$sp
805   .cprestore  16
806   sw  $4,32($fp)
807   sw  $5,36($fp)
808   lw  $2,32($fp)
809   li  $3,2        # 0x2
810   beq $2,$3,$L43
811   nop
812
813   slt $3,$2,3
```

```
814   beq $3,$0,$L44
815   nop
816
817   li  $3,1        # 0x1
818   beq $2,$3,$L45
819   nop
820
821   b $L42
822   nop
823
824 $L44:
825   li  $3,3        # 0x3
826   beq $2,$3,$L46
827   nop
828
829   li  $3,5        # 0x5
830   beq $2,$3,$L47
831   nop
832
833   b $L42
834   nop
835
836 $L45:
837   li  $2,7        # 0x7
838   b $L48
839   nop
840
841 $L43:
842   lw  $4,36($fp)
843   lw  $2,%got(get_2args_mode)($28)
844   move  $25,$2
845   .reloc  1f,R_MIPS_JALR,get_2args_mode
846 1:  jalr  $25
847   nop
848
849   lw  $28,16($fp)
850   b $L48
851   nop
852
853 $L46:
854   lw  $4,36($fp)
855   lw  $2,%got(get_3args_mode)($28)
856   move  $25,$2
857   .reloc  1f,R_MIPS_JALR,get_3args_mode
858 1:  jalr  $25
859   nop
860
861   lw  $28,16($fp)
862   b $L48
863   nop
864
865 $L47:
866   lw  $4,36($fp)
867   lw  $2,%got(get_5args_mode)($28)
```

```
868   move  $25,$2
869   .reloc  1f,R_MIPS_JALR,get_5args_mode
870 1:  jalr  $25
871   nop
872
873   lw  $28,16($fp)
874   b $L48
875   nop
876
877 $L42:
878   li  $2,-1     # 0xffffffffffffffff
879 $L48:
880   move  $sp,$fp
881   lw  $31,28($sp)
882   lw  $fp,24($sp)
883   addiu $sp,$sp,32
884   jr  $31
885   nop
886
887   .set  macro
888   .set  reorder
889   .end  get_exec_mode
890   .size get_exec_mode, .-get_exec_mode
891   .align  2
892   .globl  remove_endline
893   .set  nomips16
894   .set  nomicromips
895   .ent  remove_endline
896   .type remove_endline, @function
897 remove_endline:
898   .frame  $fp,40,$31    # vars= 8, regs= 2/0, args= 16, gp= 8
899   .mask 0xc0000000,-4
900   .fmask  0x00000000,0
901   .set  noreorder
902   .cpload $25
903   .set  nomacro
904   addiu $sp,$sp,-40
905   sw  $31,36($sp)
906   sw  $fp,32($sp)
907   move  $fp,$sp
908   .cprestore  16
909   sw  $4,40($fp)
910   lw  $4,40($fp)
911   lw  $2,%call16(strlen)($28)
912   move  $25,$2
913   .reloc  1f,R_MIPS_JALR,strlen
914 1:  jalr  $25
915   nop
916
917   lw  $28,16($fp)
918   sw  $2,24($fp)
919   lw  $2,24($fp)
920   addiu $2,$2,-1
921   lw  $3,40($fp)
```

```
922   addu  $2,$3,$2
923   lb  $3,0($2)
924   li  $2,10     # 0xa
925   bne $3,$2,$L51
926   nop
927
928   lw  $2,24($fp)
929   addiu $2,$2,-1
930   lw  $3,40($fp)
931   addu  $2,$3,$2
932   sb  $0,0($2)
933 $L51:
934   nop
935   move  $sp,$fp
936   lw  $31,36($sp)
937   lw  $fp,32($sp)
938   addiu $sp,$sp,40
939   jr  $31
940   nop
941
942   .set  macro
943   .set  reorder
944   .end  remove_endline
945   .size remove_endline, .-remove_endline
946   .rdata
947   .align  2
948 $LC8:
949   .ascii  " \000"
950   .text
951   .align  2
952   .globl  parse_vec_buffer
953   .set  nomips16
954   .set  nomicromips
955   .ent  parse_vec_buffer
956   .type parse_vec_buffer, @function
957 parse_vec_buffer:
958   .frame  $fp,40,$31    # vars= 8, regs= 2/0, args= 16, gp= 8
959   .mask 0xc0000000,-4
960   .fmask  0x00000000,0
961   .set  noreorder
962   .cpload $25
963   .set  nomacro
964   addiu $sp,$sp,-40
965   sw  $31,36($sp)
966   sw  $fp,32($sp)
967   move  $fp,$sp
968   .cprestore  16
969   sw  $4,40($fp)
970   sw  $5,44($fp)
971   lw  $4,44($fp)
972   lw  $2,%got(vector_clear)($28)
973   move  $25,$2
974   .reloc  1f,R_MIPS_JALR,vector_clear
975 1:  jalr  $25
```

```
976   nop
977
978   lw   $28,16($fp)
979   lw   $4,40($fp)
980   lw   $2,%got(remove_endline)($28)
981   move   $25,$2
982   .reloc   1f,R_MIPS_JALR,remove_endline
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   lw   $4,40($fp)
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   b  $L53
999   nop
1000
1001 $L54:
1002   li   $6,10       # 0xa
1003   move   $5,$0
1004   lw   $4,24($fp)
1005   lw   $2,%call16(strtol)($28)
1006   move   $25,$2
1007   .reloc   1f,R_MIPS_JALR,strtol
1008 1:   jalr   $25
1009   nop
1010
1011   lw   $28,16($fp)
1012   sw   $2,28($fp)
1013   lw   $5,28($fp)
1014   lw   $4,44($fp)
1015   lw   $2,%got(vector_push)($28)
1016   move   $25,$2
1017   .reloc   1f,R_MIPS_JALR,vector_push
1018 1:   jalr   $25
1019   nop
1020
1021   lw   $28,16($fp)
1022   lw   $2,%got($LC8)($28)
1023   addiu  $5,$2,%lo($LC8)
1024   move   $4,$0
1025   lw   $2,%call16(strtok)($28)
1026   move   $25,$2
1027   .reloc   1f,R_MIPS_JALR,strtok
1028 1:   jalr   $25
1029   nop
```

```
1030
1031    lw  $28,16($fp)
1032    sw  $2,24($fp)
1033  $L53:
1034    lw  $2,24($fp)
1035    bne $2,$0,$L54
1036    nop
1037
1038    move  $2,$0
1039    move  $sp,$fp
1040    lw  $31,36($sp)
1041    lw  $fp,32($sp)
1042    addiu $sp,$sp,40
1043    jr  $31
1044    nop
1045
1046    .set  macro
1047    .set  reorder
1048    .end  parse_vec_buffer
1049    .size parse_vec_buffer, .-parse_vec_buffer
1050    .align  2
1051    .globl  read_vector
1052    .set  nomips16
1053    .set  nomicromips
1054    .ent  read_vector
1055    .type read_vector, @function
1056  read_vector:
1057    .frame  $fp,48,$31    # vars= 16, regs= 2/0, args= 16, gp= 8
1058    .mask 0xc0000000,-4
1059    .fmask  0x00000000,0
1060    .set  noreorder
1061    .cpload $25
1062    .set  nomacro
1063    addiu $sp,$sp,-48
1064    sw  $31,44($sp)
1065    sw  $fp,40($sp)
1066    move  $fp,$sp
1067    .cprestore  16
1068    sw  $4,48($fp)
1069    sw  $5,52($fp)
1070    sw  $0,28($fp)
1071    sw  $0,32($fp)
1072    addiu $3,$fp,32
1073    addiu $2,$fp,28
1074    lw  $6,48($fp)
1075    move  $5,$3
1076    move  $4,$2
1077    lw  $2,%call16(getline)($28)
1078    move  $25,$2
1079    .reloc  1f,R_MIPS_JALR,getline
1080  1:  jalr  $25
1081    nop
1082
1083    lw  $28,16($fp)
```

```
1084   sw   $2,24($fp)
1085   lw   $3,24($fp)
1086   li   $2,-1      # 0xffffffffffffffff
1087   beq  $3,$2,$L57
1088   nop
1089
1090   lw   $2,24($fp)
1091   bne  $2,$0,$L58
1092   nop
1093
1094 $L57:
1095   lw   $3,24($fp)
1096   li   $2,-1      # 0xffffffffffffffff
1097   bne  $3,$2,$L59
1098   nop
1099
1100   move   $4,$0
1101   lw   $2,%call16(perror)($28)
1102   move   $25,$2
1103   .reloc   1f,R_MIPS_JALR,perror
1104 1:   jalr   $25
1105   nop
1106
1107   lw   $28,16($fp)
1108 $L59:
1109   lw   $2,28($fp)
1110   move   $4,$2
1111   lw   $2,%call16(free)($28)
1112   move   $25,$2
1113   .reloc   1f,R_MIPS_JALR,free
1114 1:   jalr   $25
1115   nop
1116
1117   lw   $28,16($fp)
1118   li   $2,-1      # 0xffffffffffffffff
1119   b $L61
1120   nop
1121
1122 $L58:
1123   lw   $2,28($fp)
1124   lw   $5,52($fp)
1125   move   $4,$2
1126   lw   $2,%got(parse_vec_buffer)($28)
1127   move   $25,$2
1128   .reloc   1f,R_MIPS_JALR,parse_vec_buffer
1129 1:   jalr   $25
1130   nop
1131
1132   lw   $28,16($fp)
1133   lw   $2,28($fp)
1134   move   $4,$2
1135   lw   $2,%call16(free)($28)
1136   move   $25,$2
1137   .reloc   1f,R_MIPS_JALR,free
```

```
1:  jalr  $25
  nop

  lw  $28,16($fp)
  move  $2,$0
$L61:
  move  $sp,$fp
  lw  $31,44($sp)
  lw  $fp,40($sp)
  addiu $sp,$sp,48
  jr  $31
  nop

  .set  macro
  .set  reorder
  .end  read_vector
  .size read_vector, .-read_vector
  .rdata
  .align  2
$LC9:
  .ascii  "%i \000"
  .align  2
$LC10:
  .ascii  "%i\000"
  .text
  .align  2
  .globl  print_sorted_vec
  .set  nomips16
  .set  nomicromips
  .ent  print_sorted_vec
  .type print_sorted_vec, @function
print_sorted_vec:
  .frame  $fp,40,$31    # vars= 8, regs= 2/0, args= 16, gp= 8
  .mask 0xc0000000,-4
  .fmask  0x00000000,0
  .set  noreorder
  .cpload $25
  .set  nomacro
  addiu $sp,$sp,-40
  sw  $31,36($sp)
  sw  $fp,32($sp)
  move  $fp,$sp
  .cprestore  16
  sw  $4,40($fp)
  sw  $5,44($fp)
  lw  $2,44($fp)
  beq $2,$0,$L63
  nop

  sw  $0,24($fp)
  b $L64
  nop

$L67:
```

```
1192   lw   $2,44($fp)
1193   lw   $2,4($2)
1194   addiu $3,$2,-1
1195   lw   $2,24($fp)
1196   sltu   $2,$2,$3
1197   beq $2,$0,$L65
1198   nop
1199
1200   lw   $2,44($fp)
1201   lw   $3,0($2)
1202   lw   $2,24($fp)
1203   sll $2,$2,2
1204   addu   $2,$3,$2
1205   lw   $2,0($2)
1206   move   $6,$2
1207   lw   $2,%got($LC9)($28)
1208   addiu $5,$2,%lo($LC9)
1209   lw   $4,40($fp)
1210   lw   $2,%call16(fprintf)($28)
1211   move   $25,$2
1212   .reloc   1f,R_MIPS_JALR,fprintf
1213 1:   jalr   $25
1214   nop
1215
1216   lw   $28,16($fp)
1217   b $L66
1218   nop
1219
1220 $L65:
1221   lw   $2,44($fp)
1222   lw   $3,0($2)
1223   lw   $2,24($fp)
1224   sll $2,$2,2
1225   addu   $2,$3,$2
1226   lw   $2,0($2)
1227   move   $6,$2
1228   lw   $2,%got($LC10)($28)
1229   addiu $5,$2,%lo($LC10)
1230   lw   $4,40($fp)
1231   lw   $2,%call16(fprintf)($28)
1232   move   $25,$2
1233   .reloc   1f,R_MIPS_JALR,fprintf
1234 1:   jalr   $25
1235   nop
1236
1237   lw   $28,16($fp)
1238 $L66:
1239   lw   $2,24($fp)
1240   addiu $2,$2,1
1241   sw   $2,24($fp)
1242 $L64:
1243   lw   $2,44($fp)
1244   lw   $3,4($2)
1245   lw   $2,24($fp)
```

```
1246   sltu   $2,$2,$3
1247   bne $2,$0,$L67
1248   nop
1249
1250 $L63:
1251   lw   $5,40($fp)
1252   li   $4,10       # 0xa
1253   lw   $2,%call16(fputc)($28)
1254   move   $25,$2
1255   .reloc   1f,R_MIPS_JALR,fputc
1256 1:   jalr   $25
1257   nop
1258
1259   lw   $28,16($fp)
1260   nop
1261   move   $sp,$fp
1262   lw   $31,36($sp)
1263   lw   $fp,32($sp)
1264   addiu $sp,$sp,40
1265   jr   $31
1266   nop
1267
1268   .set   macro
1269   .set   reorder
1270   .end   print_sorted_vec
1271   .size print_sorted_vec, .-print_sorted_vec
1272   .rdata
1273   .align   2
1274 $LC11:
1275   .ascii   "Usage:\000"
1276   .align   2
1277 $LC12:
1278   .ascii   "\011tp1 -h\000"
1279   .align   2
1280 $LC13:
1281   .ascii   "\011tp1 -V\000"
1282   .align   2
1283 $LC14:
1284   .ascii   "\011tp1 -i in_file -o out_file\000"
1285   .align   2
1286 $LC15:
1287   .ascii   "Options:\000"
1288   .align   2
1289 $LC16:
1290   .ascii   "\011-V, --version Print version and quit.\000"
1291   .align   2
1292 $LC17:
1293   .ascii   "\011-h, --help Print this information and quit.\000"
1294   .align   2
1295 $LC18:
1296   .ascii   "\011-i, --input Specify input stream/file, '-' for stdin"
1297   .ascii   "\000"
1298   .align   2
1299 $LC19:
```

```
1300    .ascii  "\011-o, --output Specify output stream/file, '-' for std"
1301    .ascii  "out.\000"
1302    .align  2
1303 $LC20:
1304    .ascii  "Examples:\000"
1305    .align  2
1306 $LC21:
1307    .ascii  "\011tp1 < in.txt > out.txt\000"
1308    .align  2
1309 $LC22:
1310    .ascii  "\011cat in.txt | tp1 -i - > out.txt\000"
1311    .text
1312    .align  2
1313    .globl  help
1314    .set    nomips16
1315    .set    nomicromips
1316    .ent    help
1317    .type   help, @function
1318 help:
1319    .frame  $fp,32,$31      # vars= 0, regs= 2/0, args= 16, gp= 8
1320    .mask   0xc0000000,-4
1321    .fmask  0x00000000,0
1322    .set    noreorder
1323    .cpload $25
1324    .set    nomacro
1325    addiu   $sp,$sp,-32
1326    sw      $31,28($sp)
1327    sw      $fp,24($sp)
1328    move    $fp,$sp
1329    .cprestore  16
1330    lw      $2,%got($LC11)($28)
1331    addiu   $4,$2,%lo($LC11)
1332    lw      $2,%call16(puts)($28)
1333    move    $25,$2
1334    .reloc  1f,R_MIPS_JALR,puts
1335 1:  jalr    $25
1336    nop
1337
1338    lw      $28,16($fp)
1339    lw      $2,%got($LC12)($28)
1340    addiu   $4,$2,%lo($LC12)
1341    lw      $2,%call16(puts)($28)
1342    move    $25,$2
1343    .reloc  1f,R_MIPS_JALR,puts
1344 1:  jalr    $25
1345    nop
1346
1347    lw      $28,16($fp)
1348    lw      $2,%got($LC13)($28)
1349    addiu   $4,$2,%lo($LC13)
1350    lw      $2,%call16(puts)($28)
1351    move    $25,$2
1352    .reloc  1f,R_MIPS_JALR,puts
1353 1:  jalr    $25
```

```
1354   nop
1355
1356   lw   $28,16($fp)
1357   lw   $2,%got($LC14)($28)
1358   addiu  $4,$2,%lo($LC14)
1359   lw   $2,%call16(puts)($28)
1360   move   $25,$2
1361   .reloc   1f,R_MIPS_JALR,puts
1362 1:   jalr   $25
1363   nop
1364
1365   lw   $28,16($fp)
1366   lw   $2,%got($LC15)($28)
1367   addiu  $4,$2,%lo($LC15)
1368   lw   $2,%call16(puts)($28)
1369   move   $25,$2
1370   .reloc   1f,R_MIPS_JALR,puts
1371 1:   jalr   $25
1372   nop
1373
1374   lw   $28,16($fp)
1375   lw   $2,%got($LC16)($28)
1376   addiu  $4,$2,%lo($LC16)
1377   lw   $2,%call16(puts)($28)
1378   move   $25,$2
1379   .reloc   1f,R_MIPS_JALR,puts
1380 1:   jalr   $25
1381   nop
1382
1383   lw   $28,16($fp)
1384   lw   $2,%got($LC17)($28)
1385   addiu  $4,$2,%lo($LC17)
1386   lw   $2,%call16(puts)($28)
1387   move   $25,$2
1388   .reloc   1f,R_MIPS_JALR,puts
1389 1:   jalr   $25
1390   nop
1391
1392   lw   $28,16($fp)
1393   lw   $2,%got($LC18)($28)
1394   addiu  $4,$2,%lo($LC18)
1395   lw   $2,%call16(puts)($28)
1396   move   $25,$2
1397   .reloc   1f,R_MIPS_JALR,puts
1398 1:   jalr   $25
1399   nop
1400
1401   lw   $28,16($fp)
1402   lw   $2,%got($LC19)($28)
1403   addiu  $4,$2,%lo($LC19)
1404   lw   $2,%call16(puts)($28)
1405   move   $25,$2
1406   .reloc   1f,R_MIPS_JALR,puts
1407 1:   jalr   $25
```

```
1408   nop
1409
1410   lw   $28,16($fp)
1411   lw   $2,%got($LC20)($28)
1412   addiu  $4,$2,%lo($LC20)
1413   lw   $2,%call16(puts)($28)
1414   move   $25,$2
1415   .reloc   1f,R_MIPS_JALR,puts
1416 1:   jalr   $25
1417   nop
1418
1419   lw   $28,16($fp)
1420   lw   $2,%got($LC21)($28)
1421   addiu  $4,$2,%lo($LC21)
1422   lw   $2,%call16(puts)($28)
1423   move   $25,$2
1424   .reloc   1f,R_MIPS_JALR,puts
1425 1:   jalr   $25
1426   nop
1427
1428   lw   $28,16($fp)
1429   lw   $2,%got($LC22)($28)
1430   addiu  $4,$2,%lo($LC22)
1431   lw   $2,%call16(puts)($28)
1432   move   $25,$2
1433   .reloc   1f,R_MIPS_JALR,puts
1434 1:   jalr   $25
1435   nop
1436
1437   lw   $28,16($fp)
1438   move   $2,$0
1439   move   $sp,$fp
1440   lw   $31,28($sp)
1441   lw   $fp,24($sp)
1442   addiu  $sp,$sp,32
1443   jr   $31
1444   nop
1445
1446   .set   macro
1447   .set   reorder
1448   .end   help
1449   .size  help, .-help
1450   .rdata
1451   .align   2
1452 $LC23:
1453   .ascii   "vsorter version: 1.0.0\000"
1454   .text
1455   .align   2
1456   .globl   version
1457   .set   nomips16
1458   .set   nomicromips
1459   .ent   version
1460   .type  version, @function
1461 version:
```

```
1462   .frame  $fp,32,$31     # vars= 0, regs= 2/0, args= 16, gp= 8
1463   .mask 0xc0000000,-4
1464   .fmask  0x00000000,0
1465   .set  noreorder
1466   .cpload $25
1467   .set  nomacro
1468   addiu $sp,$sp,-32
1469   sw  $31,28($sp)
1470   sw  $fp,24($sp)
1471   move  $fp,$sp
1472   .cprestore  16
1473   lw  $2,%got($LC23)($28)
1474   addiu $4,$2,%lo($LC23)
1475   lw  $2,%call16(puts)($28)
1476   move  $25,$2
1477   .reloc  1f,R_MIPS_JALR,puts
1478 1:  jalr  $25
1479   nop
1480
1481   lw  $28,16($fp)
1482   move  $2,$0
1483   move  $sp,$fp
1484   lw  $31,28($sp)
1485   lw  $fp,24($sp)
1486   addiu $sp,$sp,32
1487   jr  $31
1488   nop
1489
1490   .set  macro
1491   .set  reorder
1492   .end  version
1493   .size version, .-version
1494   .align  2
1495   .globl  sort
1496   .set  nomips16
1497   .set  nomicromips
1498   .ent  sort
1499   .type sort, @function
1500 sort:
1501   .frame  $fp,48,$31     # vars= 16, regs= 2/0, args= 16, gp= 8
1502   .mask 0xc0000000,-4
1503   .fmask  0x00000000,0
1504   .set  noreorder
1505   .cpload $25
1506   .set  nomacro
1507   addiu $sp,$sp,-48
1508   sw  $31,44($sp)
1509   sw  $fp,40($sp)
1510   move  $fp,$sp
1511   .cprestore  16
1512   sw  $4,48($fp)
1513   sw  $5,52($fp)
1514   addiu $2,$fp,24
1515   move  $4,$2
```

```
1516   lw   $2,%got(vector_init)($28)
1517   move   $25,$2
1518   .reloc   1f,R_MIPS_JALR,vector_init
1519 1:   jalr   $25
1520   nop
1521
1522   lw   $28,16($fp)
1523   b $L73
1524   nop
1525
1526 $L75:
1527   addiu $2,$fp,24
1528   move   $4,$2
1529   lw   $2,%got(vector_empty)($28)
1530   move   $25,$2
1531   .reloc   1f,R_MIPS_JALR,vector_empty
1532 1:   jalr   $25
1533   nop
1534
1535   lw   $28,16($fp)
1536   xori   $2,$2,0x1
1537   andi   $2,$2,0x00ff
1538   beq $2,$0,$L74
1539   nop
1540
1541   lw   $2,24($fp)
1542   lw   $3,28($fp)
1543   move   $5,$3
1544   move   $4,$2
1545   lw   $2,%call16(merge_sort)($28)
1546   move   $25,$2
1547   .reloc   1f,R_MIPS_JALR,merge_sort
1548 1:   jalr   $25
1549   nop
1550
1551   lw   $28,16($fp)
1552   addiu $2,$fp,24
1553   move   $5,$2
1554   lw   $4,52($fp)
1555   lw   $2,%got(print_sorted_vec)($28)
1556   move   $25,$2
1557   .reloc   1f,R_MIPS_JALR,print_sorted_vec
1558 1:   jalr   $25
1559   nop
1560
1561   lw   $28,16($fp)
1562   b $L73
1563   nop
1564
1565 $L74:
1566   move   $5,$0
1567   lw   $4,52($fp)
1568   lw   $2,%got(print_sorted_vec)($28)
1569   move   $25,$2
```

```
1570    .reloc  1f,R_MIPS_JALR,print_sorted_vec
1571 1:  jalr   $25
1572    nop
1573
1574    lw   $28,16($fp)
1575 $L73:
1576    addiu $2,$fp,24
1577    move   $5,$2
1578    lw   $4,48($fp)
1579    lw   $2,%got(read_vector)($28)
1580    move   $25,$2
1581    .reloc  1f,R_MIPS_JALR,read_vector
1582 1:  jalr   $25
1583    nop
1584
1585    lw   $28,16($fp)
1586    beq  $2,$0,$L75
1587    nop
1588
1589    addiu $2,$fp,24
1590    move   $4,$2
1591    lw   $2,%got(vector_destroy)($28)
1592    move   $25,$2
1593    .reloc  1f,R_MIPS_JALR,vector_destroy
1594 1:  jalr   $25
1595    nop
1596
1597    lw   $28,16($fp)
1598    nop
1599    move   $sp,$fp
1600    lw   $31,44($sp)
1601    lw   $fp,40($sp)
1602    addiu $sp,$sp,48
1603    jr   $31
1604    nop
1605
1606    .set   macro
1607    .set   reorder
1608    .end   sort
1609    .size  sort, .-sort
1610    .rdata
1611    .align  2
1612 $LC24:
1613    .ascii  "-\000"
1614    .align  2
1615 $LC25:
1616    .ascii  "unrecognized command line option\012\000"
1617    .align  2
1618 $LC26:
1619    .ascii  "r+\000"
1620    .align  2
1621 $LC27:
1622    .ascii  "could not open input file\012\000"
1623    .align  2
```

```
$LC28:
  .ascii  "w+\000"
  .align  2
$LC29:
  .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  $2,%got($LC24)($28)
  addiu $2,$2,%lo($LC24)
  sw  $2,24($fp)
  lw  $2,%got(stdin)($28)
  lw  $2,0($2)
  sw  $2,28($fp)
  lw  $2,%got($LC24)($28)
  addiu $2,$2,%lo($LC24)
  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,$2,8
  beq $2,$0,$L77
  nop

  lw  $2,40($fp)
```

```
1678    sll $3,$2,2
1679    lw  $2,%got($L79)($28)
1680    addiu $2,$2,%lo($L79)
1681    addu  $2,$3,$2
1682    lw  $2,0($2)
1683    addu  $2,$2,$28
1684    jr  $2
1685    nop
1686
1687    .rdata
1688    .align  2
1689    .align  2
1690 $L79:
1691    .gpword $L77
1692    .gpword $L78
1693    .gpword $L80
1694    .gpword $L81
1695    .gpword $L82
1696    .gpword $L83
1697    .gpword $L84
1698    .gpword $L93
1699    .text
1700 $L80:
1701    lw  $2,%got(version)($28)
1702    move  $25,$2
1703    .reloc  1f,R_MIPS_JALR,version
1704 1:  jalr  $25
1705    nop
1706
1707    lw  $28,16($fp)
1708    b $L86
1709    nop
1710
1711 $L78:
1712    lw  $2,%got(help)($28)
1713    move  $25,$2
1714    .reloc  1f,R_MIPS_JALR,help
1715 1:  jalr  $25
1716    nop
1717
1718    lw  $28,16($fp)
1719    b $L86
1720    nop
1721
1722 $L81:
1723    lw  $2,60($fp)
1724    lw  $2,8($2)
1725    sw  $2,24($fp)
1726    b $L87
1727    nop
1728
1729 $L82:
1730    lw  $2,60($fp)
1731    lw  $2,8($2)
```

```
1732   sw   $2,32($fp)
1733   b $L87
1734   nop
1735
1736 $L83:
1737   lw   $2,60($fp)
1738   lw   $2,8($2)
1739   sw   $2,24($fp)
1740   lw   $2,60($fp)
1741   lw   $2,16($2)
1742   sw   $2,32($fp)
1743   b $L87
1744   nop
1745
1746 $L84:
1747   lw   $2,60($fp)
1748   lw   $2,16($2)
1749   sw   $2,24($fp)
1750   lw   $2,60($fp)
1751   lw   $2,8($2)
1752   sw   $2,32($fp)
1753   b $L87
1754   nop
1755
1756 $L77:
1757   lw   $2,%got(stderr)($28)
1758   lw   $2,0($2)
1759   move  $7,$2
1760   li   $6,33       # 0x21
1761   li   $5,1        # 0x1
1762   lw   $2,%got($LC25)($28)
1763   addiu $4,$2,%lo($LC25)
1764   lw   $2,%call16(fwrite)($28)
1765   move  $25,$2
1766   .reloc   1f,R_MIPS_JALR,fwrite
1767 1:   jalr   $25
1768   nop
1769
1770   lw   $28,16($fp)
1771   li   $2,-1       # 0xffffffffffffffff
1772   b $L86
1773   nop
1774
1775 $L93:
1776   nop
1777 $L87:
1778   lw   $2,%got($LC24)($28)
1779   addiu $5,$2,%lo($LC24)
1780   lw   $4,24($fp)
1781   lw   $2,%call16(strcmp)($28)
1782   move  $25,$2
1783   .reloc   1f,R_MIPS_JALR,strcmp
1784 1:   jalr   $25
1785   nop
```

```
1786
1787    lw   $28,16($fp)
1788    sltu   $2,$2,1
1789    sb   $2,44($fp)
1790    lw   $2,%got($LC24)($28)
1791    addiu  $5,$2,%lo($LC24)
1792    lw   $4,32($fp)
1793    lw   $2,%call16(strcmp)($28)
1794    move   $25,$2
1795    .reloc   1f,R_MIPS_JALR,strcmp
1796 1:   jalr   $25
1797    nop
1798
1799    lw   $28,16($fp)
1800    sltu   $2,$2,1
1801    sb   $2,45($fp)
1802    lbu  $2,44($fp)
1803    xori   $2,$2,0x1
1804    andi   $2,$2,0x00ff
1805    beq  $2,$0,$L88
1806    nop
1807
1808    lw   $2,%got($LC26)($28)
1809    addiu  $5,$2,%lo($LC26)
1810    lw   $4,24($fp)
1811    lw   $2,%call16(fopen)($28)
1812    move   $25,$2
1813    .reloc   1f,R_MIPS_JALR,fopen
1814 1:   jalr   $25
1815    nop
1816
1817    lw   $28,16($fp)
1818    sw   $2,28($fp)
1819    lw   $2,28($fp)
1820    bne  $2,$0,$L88
1821    nop
1822
1823    lw   $2,%got(stderr)($28)
1824    lw   $2,0($2)
1825    move   $7,$2
1826    li   $6,26       # 0x1a
1827    li   $5,1        # 0x1
1828    lw   $2,%got($LC27)($28)
1829    addiu  $4,$2,%lo($LC27)
1830    lw   $2,%call16(fwrite)($28)
1831    move   $25,$2
1832    .reloc   1f,R_MIPS_JALR,fwrite
1833 1:   jalr   $25
1834    nop
1835
1836    lw   $28,16($fp)
1837    li   $2,-1       # 0xffffffffffffffff
1838    b  $L86
1839    nop
```

```
1840
1841 $L88:
1842   lbu  $2,45($fp)
1843   xori   $2,$2,0x1
1844   andi   $2,$2,0x00ff
1845   beq  $2,$0,$L89
1846   nop
1847
1848   lw   $2,%got($LC28)($28)
1849   addiu  $5,$2,%lo($LC28)
1850   lw   $4,32($fp)
1851   lw   $2,%call16(fopen)($28)
1852   move   $25,$2
1853   .reloc   1f,R_MIPS_JALR,fopen
1854 1:   jalr   $25
1855   nop
1856
1857   lw   $28,16($fp)
1858   sw   $2,36($fp)
1859   lw   $2,36($fp)
1860   bne  $2,$0,$L89
1861   nop
1862
1863   lw   $2,%got($LC29)($28)
1864   addiu  $4,$2,%lo($LC29)
1865   lw   $2,%call16(perror)($28)
1866   move   $25,$2
1867   .reloc   1f,R_MIPS_JALR,perror
1868 1:   jalr   $25
1869   nop
1870
1871   lw   $28,16($fp)
1872   lbu  $2,44($fp)
1873   xori   $2,$2,0x1
1874   andi   $2,$2,0x00ff
1875   beq  $2,$0,$L90
1876   nop
1877
1878   lw   $4,28($fp)
1879   lw   $2,%call16(fclose)($28)
1880   move   $25,$2
1881   .reloc   1f,R_MIPS_JALR,fclose
1882 1:   jalr   $25
1883   nop
1884
1885   lw   $28,16($fp)
1886 $L90:
1887   li   $2,-1       # 0xffffffffffffffff
1888   b $L86
1889   nop
1890
1891 $L89:
1892   lw   $5,36($fp)
1893   lw   $4,28($fp)
```

```
1894    lw   $2,%got(sort)($28)
1895    move   $25,$2
1896    .reloc   1f,R_MIPS_JALR,sort
1897 1:   jalr   $25
1898    nop
1899
1900    lw   $28,16($fp)
1901    lbu  $2,44($fp)
1902    xori   $2,$2,0x1
1903    andi   $2,$2,0x00ff
1904    beq  $2,$0,$L91
1905    nop
1906
1907    lw   $4,28($fp)
1908    lw   $2,%call16(fclose)($28)
1909    move   $25,$2
1910    .reloc   1f,R_MIPS_JALR,fclose
1911 1:   jalr   $25
1912    nop
1913
1914    lw   $28,16($fp)
1915 $L91:
1916    lbu  $2,45($fp)
1917    xori   $2,$2,0x1
1918    andi   $2,$2,0x00ff
1919    beq  $2,$0,$L92
1920    nop
1921
1922    lw   $4,36($fp)
1923    lw   $2,%call16(fclose)($28)
1924    move   $25,$2
1925    .reloc   1f,R_MIPS_JALR,fclose
1926 1:   jalr   $25
1927    nop
1928
1929    lw   $28,16($fp)
1930 $L92:
1931    move   $2,$0
1932 $L86:
1933    move   $sp,$fp
1934    lw   $31,52($sp)
1935    lw   $fp,48($sp)
1936    addiu  $sp,$sp,56
1937    jr   $31
1938    nop
1939
1940    .set   macro
1941    .set   reorder
1942    .end   main
1943    .size  main, .-main
1944    .ident   "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
1945
```

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).