FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Organización de Computadoras 66.20

Trabajo práctico 1: Programación MIPS

Primer Cuatrimestre 2020
Grupo:
Integrantes:

Alumno	Padron	Email
Álvaro Iribarren	101049	airibarren@fi.uba.ar
Dario Hernan Markarian	98684	dmarkarian@fi.uba.ar
Joaquin Parodi	100752	jparodi@fi.uba.ar

Índice

1. Introducción

El objetivo principal de este trabajo práctico es el de familiarizarse con el conjunto de instrucciones ***MIPS*** y el concepto de ***ABI***, extendiendo un programa que deberá procesar un stream de vectores de números enteros ordenando cada vector en forma creciente, e imprimir inmediatamente el resultado por el stream de salida.

2. Documentación relevante en el diseño del programa

En el diseño del programa se intento separar en archivos las distintas funcionalidades. Ya se arrancó con una ventaja que fueron las funciones de alocar memoria dinamica y liberarla. Estas se encuentran en mymalloc.S. Una vez dentro de nuestra implementación se utilizó esta función para crear un array temporal donde en cada iteración se iban intercambiando de posición los números según el diseño del algoritmo merge-sort. Una vez terminadas todas las iteraciones se escribe el array temporal en el original y se libera la memoria alocada. Se usaron los archivos merge.S y functions.S.

2.1. Merge.S

Este solo contiene el cuerpo principal de la función merge llamada desde C. Internamente se hace una llamada a la función "merge_sort"presente en el archivo a continuación.

Section	Comentarios	Dirección	Merge
ABA(CALL		36	size(a1)
	sp (antes)	32	array(a0)
SRA		28	padding
		24	ra
		20	fp
		16	gp
ABA(callee)		12	size(a3)
		8	right index (a2)
		4	left index (a1)
	sp (despues)	0	array(a0)

Figura 1: Stack frame de la función merge.

2.2. Functions.S:

En este archivo se encuentra la lógica cruda del algoritmo. Como se dijo en el item anterior, se comienza con una llamada a la función merge_sort la cual realiza llamadas recursivas para generar los sub arrays y luego se utiliza la función merge_sub_arrays para realizar los ordenamientos correspondientes sobre el array temporal.

2.2.1. Merge_sort

Esta función recibe cuatro parámetros.

1. Dirección de array original
2. Left_index: Índice de comienzo del array o subarray dependiendo de si es la primera llamada o unas de las siguientes.
3. Right_index: Índice de fin del array o sub array. En la primera llamada este valor es igual al size del array menos uno.
4. Size: Tamaño del array original.

Section	Comentarios	Dirección	Merge-sort
ABA (caller)		60	max size (a3)
		56	right index (a2)
		52	left index (a1)
	sp (antes)	48	array (a0)
SRA		44	padding
		40	ra
		36	fp
		32	gp
LTA		28	padding
		24	medio
ABA(callee)		20	size (stack)
		16	right index (stack)
		12	left index 2 (a3)
		8	right index 1 (a2)
		4	left index 1 (a1)
	sp (despues)	0	array (a0)

Figura 2: Stack frame de la función merge-sort.

2.2.2. Merge_sub_array

Como se dijo previamente, es la función usada para ir ordenando los sub arrays en el array temporal. Recibe seis parámetros.

1. Dirección de comienzo del array original.
2. Índice izquierdo 1 de array o sub array.
3. Índice derecho 1 de array o sub array.
4. Índice izquierdo 2 de array o sub array.
5. Índice derecho 2 de array o sub array.
6. Tamaño del array original.

Dentro de la función fue necesario utilizar 3 variables como iteradores que fueron llamados i,j y k para realizar los accesos. Estos fueron ubicados en la LTA.

Section		Dirección	Merge-sub-arrays
ABA (caller		68	max size (stack)
		64	right index 2 (stack)
		60	left index 2 (a3)
		56	right index 1 (a2)
		52	left index 1 (a1)
	sp (antes)	48	array (a0)
SRA		44	padding
		40	ra
		36	fp
		32	gp
LTA		28	k
		24	j
		20	i
		16	dir array temporal
ABA(callee)		12	
		8	
		4	
	sp (despues)	0	

Figura 3: Stack frame de la función merge-sub-arrays.

3. Implementación del programa

El programa a desarrollar consta 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 es:

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

Se usa el algoritmo de ***merge sort*** y el modo 1 del sistema operativo para manejo de acceso no alineado a memoria. En cuanto al manejo de memoria dinámica realizado por ***merge sort***, está realizado

íntegramente en ***MIPS***, usando la implementación de referencia ***mymalloc/myfree***

4. Comandos para compilar el programa

```
gcc -Wall -g -o tp1 merge.c merge.S functions.S mymalloc.S
```

5. Corridas de prueba

```
Dado el siguiente flujo de entrada:
$ cat input.txt
3 2 1
6 5 1 2 9 3 8 7 4
6 0 0 1 3
-1
9

Al ejecutar el programa:
$ ./tp1 -i input.txt -o -
1 2 3
1 2 3 4 5 6 7 8 9
0 0 1 3 6
-1

Dado el siguiente flujo de entrada:
$ cat input.txt
1 2 4 6 8
0 7 3 8 3
4 7 3 8    2
1 2 4 6 8
0 7 3 8 3
4 7 3 8    2
2 5 1 0 3 3 3 3 3 3 3 5 5  55 6 6 7  78 8 8 97 9 0 0 0 2  2

Al ejecutar el programa:
$ cat prueba.txt | ./tp1
1 2 4 6 8
```

```
0 3 3 7 8
2 3 4 7 8
0 0 0 0 1 2 2 2 3 3 3 3 3 3 3 5 5 5 6 6 7 8 8 9 55 78 97
```

6. Conclusiones

Se realizó un programa capaz de recibir, tanto por entrada estandar como por un archivo determinado, una serie de números separada por espacios y ordenar dichos números. La forma en lo que se hace esto es parseando cada linea de numeros a un array y utilizando el algoritmo merge-sort implementado en lenguaje ensamblador. La salida por default es la salida estandar, de otra forma podríamos indicarle al programa una salida determinada, como puede ser un archivo de texto. Como comentario, es posible la ejecución a traves de pipes y redirecciones implementadas en el sistema operativo ya que estas insertan o toman datos del programa a través de los canales estándares.

7. Código fuente en lenguaje C

MAIN.c

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include "string.h"
#include <ctype.h>
#include <assert.h>

#define HELP "-h"
#define VERSION "-v"
#define INPUT "-i"
#define OUTPUT "-o"
```

```
#define STD "-"
#define BASE 10

int* get_numbers(char* line, size_t capacity, int* amountOfNumbers);
void write_vector(FILE* out_stream, int* vector, int length);
void help();
bool strings_are_equal(char* str1, char* str2);
bool check_if_there_are_letters(char* buffer);
bool endOfLine(char* str);
int merge(int* array, int size);



void help(){
    printf("Usage:\n");
    printf(" tp1 -h\n");
    printf(" tp1 -V\n");
    printf(" tp1 -i in_file -o out_file\n");
    printf("Options:\n");
    printf(" -V, --version Print version and quit.\n");
    printf(" -h, --help Print this information and quit.\n");
    printf(" -i, --input Specify input stream/file, '-' for stdin.\n")
    printf(" -o, --output Specify output stream/file, '-' for stdout.\
    printf("Examples:\n");
    printf(" tp1 < in.txt > out.txt\n");
    printf(" cat in.txt | tp1 -i - > out.txt\n");
}

bool check_if_there_are_letters(char* buffer){
    char* endptr = NULL;
    strtol(buffer, &endptr, BASE);

    //chequeo el resultado de strtol, que detecta los caracteres
    bool lettersFound = *endptr != '\0' || endptr == buffer;
    return lettersFound;
}
```

```
bool strings_are_equal(char* str1, char* str2){
    if (str1 != NULL && str2 != NULL)
        return strcmp(str1, str2) == 0;

    return false;
}

void check_malloc(int size, void* ptr){
    if (size != 0 && ptr == NULL){
        fprintf(stderr, "Fallo al alocar memoria");
        exit(-1);
    }
}

bool endOfLine(char* str){
    return *str == '\n' || strings_are_equal(str, "\\n");
}

int* get_numbers(char* line, size_t capacity, int* amountOfNumbers) {
    bool lettersFound = false;
    char* buffer = malloc(capacity * sizeof(char));
    check_malloc(capacity, buffer);

    strncpy(buffer, line, capacity);
    char* ptr;
    ptr = strtok (buffer, " \n");

    int* vector = malloc(capacity * sizeof(int));
    check_malloc(capacity, vector);

    int i = 0;
    for (; ptr != NULL && !endOfLine(ptr) && !lettersFound; i++) {
        if (i == capacity){
            capacity += 10;
            vector = (int*)realloc(vector, capacity*sizeof(int));
```

```
        }
        lettersFound = check_if_there_are_letters(ptr);
        if (!lettersFound){
            vector[i] = atoi(ptr);
            ptr = strtok (NULL, " \n");
        }
    }

    if (lettersFound){
        free(vector);
        vector = NULL;
    }

    *amountOfNumbers = i;
    free(buffer);
    return vector;
}

void write_vector(FILE* out_stream, int* vector, int length){
    for (int i=0; i < length; i++) {
        fprintf(out_stream,"%d ", vector[i]);
    }
    fprintf(out_stream,"\n");
}

void read_file(FILE* in_stream, FILE* out_stream){
    char* line = NULL;
    size_t bufSize = 0;
    bool letterFound = false;
    bool error = false;

    while(!letterFound && !error && (bufSize = getline(&line, &bufSize
        if (bufSize != 1 || line[0] != ' '){ //si es un espacio vacio
            int amountOfNumbers = 0;
            int *vector = get_numbers(line, bufSize + 1,        //sumo
                                      &amountOfNumbers);
```

```
            letterFound = (vector == NULL);
            if (!letterFound) {
                int return_value = merge(vector, amountOfNumbers);
                if (return_value == 0) {
                    write_vector(out_stream, vector, amountOfNumbers);
                    free(vector);
                } else {
                    error = true;
                    fprintf(stderr, "Ocurrió un error en la función de
                }
            } else {
                fprintf(stderr, "Se encontro una letra en el programa
            }
        }
    }
    free(line);
}


int main(int argc, char* argv[]) {
    FILE* in = stdin;
    FILE* out = stdout;

    char* execOption = argv[1];
    if (execOption != NULL) {
        if (strings_are_equal(execOption, HELP)) {
            help();
            return 0;
        } else if (strings_are_equal(execOption, VERSION)) {
            printf("versión TP1 Organización de computadoras\n");
            return 0;
        } else if (strings_are_equal(execOption, INPUT)) {
            char *in_file = argv[2];
            assert(in_file);
```

```
            //abro infile
            if (!strings_are_equal(in_file, STD) && (in = fopen(in_fil
                fprintf(stderr, "Error al abrir el archivo de entrada'
                return -1;
            }

            char *out_command = argv[3];
            char *out_file = argv[4];

            bool presentOutput = strings_are_equal(out_command, OUTPUT
            bool stdOut = strings_are_equal(out_file, STD);
            if (presentOutput && !stdOut) {
                if ((out = fopen(out_file, "w+")) == NULL) {
                    fprintf(stderr, "Error al abrir el archivo de sali
                    return -1;
                }
            } else if (!presentOutput || out_file == NULL) {
                out = stdout;
            } else {
                fprintf(stderr,"Las opciones para el out command son
                return -1;
            }
        } else {
            fprintf(stderr, "Parametro de entrada invalido\n");
            return -1;
        }
    }


    read_file(in, out);

    if (in != stdin)
        fclose(in);

    if (out != stdout){
        fclose(out);
```

```
    }

    return 0;
}
```

8. Código fuente en lenguaje MIPS

MERGE.S

```
#include <sys/regdef.h>
#include <sys/syscall.h>
#include <asm/mman.h>

.abicalls
.text
.align  2
.globl merge
.ent  merge

#define MERGE_STACK_SIZE  40
#define MERGE_RA_OFFSET 32
#define MERGE_FP_OFFSET  28
#define MERGE_GP_OFFSET  24

merge:
.frame fp,MERGE_STACK_SIZE,ra
subu sp,sp,MERGE_STACK_SIZE
.cprestore  MERGE_GP_OFFSET
sw ra,MERGE_RA_OFFSET(sp)
sw fp,MERGE_FP_OFFSET(sp)
move fp,sp

sw a0,40(fp) #direccion del array original
sw a1,44(fp) #size del array original
```

```
call_merge:
lw a0,40(fp) #a0: array dir
li a1,0 #a1: left_index =  0
lw a2,44(fp) #a2: right_index
lw a3,44(fp) #a3: max size
addiu a2,a2,-1 #a2: right index-1
jal merge_sort

fin:
li v0,0 #si llegue acá es porque no hubo errores.
lw ra,MERGE_RA_OFFSET(sp)
lw fp,MERGE_FP_OFFSET(sp)
addu sp,sp,MERGE_STACK_SIZE

jr ra
.end merge
```

FUNCTIONS.S

```
#include <sys/regdef.h>
#include <sys/syscall.h>
#include <asm/mman.h>

.abicalls
.text
.align 2
.globl merge_sub_arrays
.ent merge_sub_arrays

#a0: array
#a1: left index 1
#a2: right index 1
#a3: left index 2
#stack: right index 2
#stack: max array size
```

```
#define SUB_ARRAY_STACK_SIZE 48
#define SUB_ARRAY_RA_OFFSET 40
#define SUB_ARRAY_FP_OFFSET 36
#define SUB_ARRAY_GP_OFFSET 32
#define TEMPORAL_OFFSET 16
#define I_OFFSET 20
#define J_OFFSET 24
#define K_OFFSET 28

merge_sub_arrays:
.frame fp,SUB_ARRAY_STACK_SIZE,ra
subu sp,sp,SUB_ARRAY_STACK_SIZE
sw ra,SUB_ARRAY_RA_OFFSET(sp)
sw fp,SUB_ARRAY_FP_OFFSET(sp)
.cprestore  SUB_ARRAY_GP_OFFSET
move fp,sp

#Parametros
sw a0,48(fp) #array
sw a1,52(fp) #left index 1
sw a2,56(fp) #right index 1
sw a3,60(fp) #left index 2
lw t3,64(sp) #right index 2, viene por stack en la
#posicion 16 del mismo. Luego hago 40 del actual stack para llegar a
#al stack del caller. Esta en la 16 porque en las 16 anteriores estan
#los primeros 4 parametros.
lw t4,68(sp) #max size: Hago lo mismo. 40+20

sw t3,64(fp) #guardo right index
sw t4,68(fp) #guardo max size


create_temporal_array:
lw a0,68(fp) #a0: max size
sll a0,a0,2 #sizex4
```

```
jal mymalloc
bgtz v0,continue

exit: li      a0,-2 #codigo de error para fallo de mymalloc
        li      v0,SYS_exit
        syscall

continue:
sw v0,TEMPORAL_OFFSET(fp) #temporal array

load_variables:
lw a0,48(fp) #array original
lw a1,TEMPORAL_OFFSET(fp) #array temporal
lw t0,52(fp) #left index 1
lw t1,56(fp) #right index 1
lw t2,60(fp) #left index 2
lw t3,64(fp) #right index  2

sll t0,t0,2
sll t1,t1,2 #multiplico por 4 los indices.
sll t2,t2,2
sll t3,t3,2


#iteradores
move t5,t0 #i = l1
move t6,t2 #j = l2
li t7,0 #k = 0;
li t8,0 #auxiliar acceso a array original
li t4,0 #auxiliar acceso a array temporal
sw t5,I_OFFSET(fp)
sw t6,J_OFFSET(fp) #guardo en LTA
sw t7,K_OFFSET(fp)


while_elements_in_both_arrays: #while(l<= l1 && r<=r2)
```

```
bgt t5,t1,while_i_less_than_right_index_1 #condicion 1
bgt t6,t3,while_i_less_than_right_index_1 #condicion 2

lw a0,48(fp) #a0: dir array
addu a0,a0,t5 #s0: dir de array[i]
lw t8,0(a0) #t8: array[i]

lw a0,48(fp) #s0: posicion inicial
addu a0,a0,t6 #s0: dir de array[j]
lw t4,0(a0) #t9: array[j]
lw a0,48(fp) #s0: posicion inicial
bge t8,t4,else_element_is_less_than  #condicion

if_element_is_less_than: #if(a[i] < a[j])
lw a1,TEMPORAL_OFFSET(fp)
addu a1,a1,t7 #s1: dir de temp[k]
sw t8,0(a1) #temp[k] = array[i]
addiu t7,t7,4
addiu t5,t5,4
lw a1,TEMPORAL_OFFSET(fp) #s1: posicion inicial
b while_elements_in_both_arrays

else_element_is_less_than:
lw a1,TEMPORAL_OFFSET(fp)
addu a1,a1,t7 #s1:dir temp[k]
sw t4,0(a1) #temp[k] = array[j]
addiu t7,t7,4
addiu t6,t6,4
lw a1,TEMPORAL_OFFSET(fp)
b while_elements_in_both_arrays


while_i_less_than_right_index_1: #while(i<=r1)
bgt t5,t1,while_j_less_than_right_index_2
lw a1,TEMPORAL_OFFSET(fp) #cargo temporal
addu a1,a1,t7 #s1: dir de temp[k]
```

```
lw a0,48(fp) #cargo array
addu a0,a0,t5 #dir de array[i]
lw t8,0(a0) #t8: array[i]
sw t8,0(a1) #temp[k] = array[i]

#aumento iteradores
addiu t7,t7,4 #k++
addiu t5,t5,4 #i++
b while_i_less_than_right_index_1

while_j_less_than_right_index_2:
bgt t6,t3,transfer_elements_from_temp_to_array
lw a1,TEMPORAL_OFFSET(fp) #cargo temporal
addu a1,a1,t7 #s1: dir de temp[k]

lw a0,48(fp) #s0: array
addu a0,a0,t6 #s0: dir de array[j]
lw t8,0(a0) #t8: array[j]
sw t8,0(a1) #temp[k] = array[j]

#aumento iteradores
addiu t6,t6,4 #j++
addiu t7,t7,4 #k++
b while_j_less_than_right_index_2

transfer_elements_from_temp_to_array:
move t5,t0 #i=i1
li t6,0 #j=0
while_i_less_or_equal_than_R2:
bgt t5,t3,free_temporal
lw a0,48(fp) #cargo array original
lw a1,TEMPORAL_OFFSET(fp) #cargo temporal
addu a0,a0,t5 #dir de array[i]
addu a1,a1,t6 #dir de temp[j]
lw t4,0(a1) #temp[j]
```

```
sw t4,0(a0) #array[i] = temp[j]
addiu t5,t5,4
addiu t6,t6,4
b while_i_less_or_equal_than_R2

free_temporal:
lw a0,TEMPORAL_OFFSET(fp)
jal myfree

return_merge_sub_arrays:
lw ra,SUB_ARRAY_RA_OFFSET(sp)
lw fp,SUB_ARRAY_FP_OFFSET(sp)
addiu sp,sp,SUB_ARRAY_STACK_SIZE
j ra
.end merge_sub_arrays


#define SORT_STACK_SIZE 48
#define SORT_SP_OFFSET 40
#define SORT_FP_OFFSET 36
#define SORT_GP_OFFSET 32
#define MIDDLE_OFFSET 24
.globl merge_sort
.ent merge_sort

merge_sort:  #(a0:array[size], a1:left index, a2:right index, a3:max s
.frame fp,SORT_STACK_SIZE,ra
subu  sp,sp,SORT_STACK_SIZE
sw ra,SORT_SP_OFFSET(sp)
sw fp,SORT_FP_OFFSET(sp)
.cprestore  SORT_GP_OFFSET
move fp,sp

sw a0,48(fp) #array
sw a1,52(fp) #left index
sw a2,56(fp) #right index
```

```
sw a3,60(fp) #max size

lw t0,48(fp)
lw t1,52(fp)
lw t2,56(fp)
lw t3,60(fp)

li t4,0 #medio
bge t1,t2,return_merge_sort #if(l<r)
addu t4,t1,t2 #l+r
srl t4,t4,1 #(l+r)/2
sw t4,MIDDLE_OFFSET(fp) #medio

left_recursion:
lw a0,48(fp) #array
lw a1,52(fp) #left
lw a2,MIDDLE_OFFSET(fp) #medio
lw a3,60(fp) #size
jal merge_sort

right_recursion:
lw a0,48(fp) #array
lw a1,MIDDLE_OFFSET(fp) #medio
addiu a1,a1,1 #medio+1
lw a2,56(fp) #right index
lw a3,60(fp) #size
jal merge_sort

merging_sub_arrays:
lw a0,48(fp) #array
lw a1,52(fp) #left index
lw  a2,MIDDLE_OFFSET(fp) #medio
addiu a3,a2,1 #medio +1

lw t0,56(fp) #j por stack
sw t0,16(sp) #Lo mando por stack
```

```
lw t1,60(fp) #t1: max_size
sw t1,20(sp) #lo mando por stack

jal  merge_sub_arrays

return_merge_sort:
lw ra,SORT_SP_OFFSET(sp)
lw fp,SORT_FP_OFFSET(sp)
addiu sp,sp,SORT_STACK_SIZE
j ra

.end merge_sort
```

9. Código MIPS32 generado por el compilador

```
.file 1 "main.c"
.section .mdebug.abi32
.previous
.nan legacy
.module fp=xx
.module nooddspreg
.abicalls
.rdata
.align 2
$LC0:
.ascii "Usage:\000"
.align 2
$LC1:
.ascii "\011tp1 -h\000"
.align 2
$LC2:
.ascii "\011tp1 -V\000"
.align 2
$LC3:
```

```
.ascii "\011tp1 -i in_file -o out_file\000"
.align 2
$LC4:
.ascii "Options:\000"
.align 2
$LC5:
.ascii "\011-V, --version\011Print version and quit.\000"
.align 2
$LC6:
.ascii "\011-h, --help\011Print this information and quit.\000"
.align 2
$LC7:
.ascii "\011-i, --input\011Specify input stream/file, '-' for st"
.ascii "din.\000"
.align 2
$LC8:
.ascii "\011-o, --output\011Specify output stream/file, '-' for "
.ascii "stdout.\000"
.align 2
$LC9:
.ascii "Examples:\000"
.align 2
$LC10:
.ascii "\011tp1 < in.txt > out.txt\000"
.align 2
$LC11:
.ascii "\011cat in.txt | tp1 -i - > out.txt\000"
.text
.align 2
.globl help
.set nomips16
.set nomicromips
.ent help
.type help, @function
help:
.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
lw $2,%got($LC0)($28)
addiu $4,$2,%lo($LC0)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC1)($28)
addiu $4,$2,%lo($LC1)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC2)($28)
addiu $4,$2,%lo($LC2)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop
```

```
lw $28,16($fp)
lw $2,%got($LC3)($28)
addiu $4,$2,%lo($LC3)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC4)($28)
addiu $4,$2,%lo($LC4)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC5)($28)
addiu $4,$2,%lo($LC5)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC6)($28)
addiu $4,$2,%lo($LC6)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop
```

```
lw $28,16($fp)
lw $2,%got($LC7)($28)
addiu $4,$2,%lo($LC7)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC8)($28)
addiu $4,$2,%lo($LC8)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC9)($28)
addiu $4,$2,%lo($LC9)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC10)($28)
addiu $4,$2,%lo($LC10)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop
```

```
lw $28,16($fp)
lw $2,%got($LC11)($28)
addiu $4,$2,%lo($LC11)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
nop
move $sp,$fp
lw $31,28($sp)
lw $fp,24($sp)
addiu $sp,$sp,32
jr $31
nop

.set macro
.set reorder
.end help
.size help, .-help
.align 2
.globl check_if_there_are_letters
.set nomips16
.set nomicromips
.ent check_if_there_are_letters
.type check_if_there_are_letters, @function
check_if_there_are_letters:
.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 $0,28($fp)
addiu $2,$fp,28
li $6,10 # 0xa
move $5,$2
lw $4,40($fp)
lw $2,%call16(strtol)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strtol
1: jalr $25
nop

lw $28,16($fp)
lw $2,28($fp)
lb $2,0($2)
bne $2,$0,$L3
nop

lw $3,28($fp)
lw $2,40($fp)
bne $3,$2,$L4
nop

$L3:
li $2,1 # 0x1
b $L5
nop

$L4:
move $2,$0
$L5:
sb $2,24($fp)
```

```
lbu $2,24($fp)
andi $2,$2,0x1
sb $2,24($fp)
lbu $2,24($fp)
move $sp,$fp
lw $31,36($sp)
lw $fp,32($sp)
addiu $sp,$sp,40
jr $31
nop

.set macro
.set reorder
.end check_if_there_are_letters
.size check_if_there_are_letters, .-check_if_there_are_letters
.align 2
.globl strings_are_equal
.set nomips16
.set nomicromips
.ent strings_are_equal
.type strings_are_equal, @function
strings_are_equal:
.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)
sw $5,36($fp)
lw $2,32($fp)
```

```
beq $2,$0,$L8
nop

lw $2,36($fp)
beq $2,$0,$L8
nop

lw $5,36($fp)
lw $4,32($fp)
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
sltu $2,$2,1
andi $2,$2,0x00ff
b $L9
nop

$L8:
move $2,$0
$L9:
move $sp,$fp
lw $31,28($sp)
lw $fp,24($sp)
addiu $sp,$sp,32
jr $31
nop

.set macro
.set reorder
.end strings_are_equal
.size strings_are_equal, .-strings_are_equal
.rdata
```

```
.align 2
$LC12:
.ascii "Fallo al alocar memoria\000"
.text
.align 2
.globl check_malloc
.set nomips16
.set nomicromips
.ent check_malloc
.type check_malloc, @function
check_malloc:
.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)
sw $5,36($fp)
lw $2,32($fp)
beq $2,$0,$L12
nop

lw $2,36($fp)
bne $2,$0,$L12
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,23 # 0x17
```

```
li $5,1 # 0x1
lw $2,%got($LC12)($28)
addiu $4,$2,%lo($LC12)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
li $4,-1 # 0xffffffffffffffff
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L12:
nop
move $sp,$fp
lw $31,28($sp)
lw $fp,24($sp)
addiu $sp,$sp,32
jr $31
nop

.set macro
.set reorder
.end check_malloc
.size check_malloc, .-check_malloc
.rdata
.align 2
$LC13:
.ascii "\\n\000"
.text
.align 2
```

```
.globl endOfLine
.set nomips16
.set nomicromips
.ent endOfLine
.type endOfLine, @function
endOfLine:
.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 $2,32($fp)
lb $3,0($2)
li $2,10 # 0xa
beq $3,$2,$L14
nop

lw $2,%got($LC13)($28)
addiu $5,$2,%lo($LC13)
lw $4,32($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L15
nop
```

```
$L14:
li $2,1 # 0x1
b $L16
nop

$L15:
move $2,$0
$L16:
andi $2,$2,0x1
andi $2,$2,0x00ff
move $sp,$fp
lw $31,28($sp)
lw $fp,24($sp)
addiu $sp,$sp,32
jr $31
nop

.set macro
.set reorder
.end endOfLine
.size endOfLine, .-endOfLine
.rdata
.align 2
$LC14:
.ascii " \012\000"
.text
.align 2
.globl get_numbers
.set nomips16
.set nomicromips
.ent get_numbers
.type get_numbers, @function
get_numbers:
.frame $fp,64,$31 # vars= 24, regs= 3/0, args= 16, gp= 8
.mask 0xc0010000,-4
```

```
.fmask 0x00000000,0
.set noreorder
.cpload $25
.set nomacro
addiu $sp,$sp,-64
sw $31,60($sp)
sw $fp,56($sp)
sw $16,52($sp)
move $fp,$sp
.cprestore 16
sw $4,64($fp)
sw $5,68($fp)
sw $6,72($fp)
sb $0,24($fp)
lw $4,68($fp)
lw $2,%call16(malloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,malloc
1: jalr $25
nop

lw $28,16($fp)
sw $2,40($fp)
lw $2,68($fp)
lw $5,40($fp)
move $4,$2
lw $2,%got(check_malloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,check_malloc
1: jalr $25
nop

lw $28,16($fp)
lw $6,68($fp)
lw $5,64($fp)
lw $4,40($fp)
```

```
lw $2,%call16(strncpy)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strncpy
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got($LC14)($28)
addiu $5,$2,%lo($LC14)
lw $4,40($fp)
lw $2,%call16(strtok)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strtok
1: jalr $25
nop

lw $28,16($fp)
sw $2,28($fp)
lw $2,68($fp)
sll $2,$2,2
move $4,$2
lw $2,%call16(malloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,malloc
1: jalr $25
nop

lw $28,16($fp)
sw $2,32($fp)
lw $2,68($fp)
lw $5,32($fp)
move $4,$2
lw $2,%got(check_malloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,check_malloc
1: jalr $25
```

```
nop

lw $28,16($fp)
sw $0,36($fp)
b $L19
nop

$L23:
lw $3,36($fp)
lw $2,68($fp)
bne $3,$2,$L20
nop

lw $2,68($fp)
addiu $2,$2,10
sw $2,68($fp)
lw $2,68($fp)
sll $2,$2,2
move $5,$2
lw $4,32($fp)
lw $2,%call16(realloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,realloc
1: jalr $25
nop

lw $28,16($fp)
sw $2,32($fp)
$L20:
lw $4,28($fp)
lw $2,%got(check_if_there_are_letters)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,check_if_there_are_letters
1: jalr $25
nop
```

```
lw $28,16($fp)
sb $2,24($fp)
lbu $2,24($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L21
nop

lw $2,36($fp)
sll $2,$2,2
lw $3,32($fp)
addu $16,$3,$2
lw $4,28($fp)
lw $2,%call16(atoi)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,atoi
1: jalr $25
nop

lw $28,16($fp)
sw $2,0($16)
lw $2,%got($LC14)($28)
addiu $5,$2,%lo($LC14)
move $4,$0
lw $2,%call16(strtok)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strtok
1: jalr $25
nop

lw $28,16($fp)
sw $2,28($fp)
$L21:
lw $2,36($fp)
addiu $2,$2,1
sw $2,36($fp)
```

```
$L19:
lw $2,28($fp)
beq $2,$0,$L22
nop

lw $4,28($fp)
lw $2,%got(endOfLine)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,endOfLine
1: jalr $25
nop

lw $28,16($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L22
nop

lbu $2,24($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
bne $2,$0,$L23
nop

$L22:
lbu $2,24($fp)
beq $2,$0,$L24
nop

lw $4,32($fp)
lw $2,%call16(free)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,free
1: jalr $25
nop
```

```
lw $28,16($fp)
sw $0,32($fp)
$L24:
lw $2,72($fp)
lw $3,36($fp)
sw $3,0($2)
lw $4,40($fp)
lw $2,%call16(free)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,free
1: jalr $25
nop

lw $28,16($fp)
lw $2,32($fp)
move $sp,$fp
lw $31,60($sp)
lw $fp,56($sp)
lw $16,52($sp)
addiu $sp,$sp,64
jr $31
nop

.set macro
.set reorder
.end get_numbers
.size get_numbers, .-get_numbers
.rdata
.align 2
$LC15:
.ascii "%d \000"
.text
.align 2
.globl write_vector
.set nomips16
.set nomicromips
```

```
.ent write_vector
.type write_vector, @function
write_vector:
.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)
sw $6,48($fp)
sw $0,24($fp)
b $L27
nop

$L28:
lw $2,24($fp)
sll $2,$2,2
lw $3,44($fp)
addu $2,$3,$2
lw $2,0($2)
move $6,$2
lw $2,%got($LC15)($28)
addiu $5,$2,%lo($LC15)
lw $4,40($fp)
lw $2,%call16(fprintf)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fprintf
1: jalr $25
nop
```

```
lw $28,16($fp)
lw $2,24($fp)
addiu $2,$2,1
sw $2,24($fp)
$L27:
lw $3,24($fp)
lw $2,48($fp)
slt $2,$3,$2
bne $2,$0,$L28
nop

lw $5,40($fp)
li $4,10 # 0xa
lw $2,%call16(fputc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fputc
1: jalr $25
nop

lw $28,16($fp)
nop
move $sp,$fp
lw $31,36($sp)
lw $fp,32($sp)
addiu $sp,$sp,40
jr $31
nop

.set macro
.set reorder
.end write_vector
.size write_vector, .-write_vector
.rdata
.align 2
$LC16:
```

```
.ascii "Ocurri\303\263 un error en la funci\303\263n de ordenami"
.ascii "ento, se procede al cierre\000"
.align 2
$LC17:
.ascii "Se encontro una letra en el programa y se procede al cie"
.ascii "rre ordenado\012\000"
.text
.align 2
.globl read_file
.set nomips16
.set nomicromips
.ent read_file
.type read_file, @function
read_file:
.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)
sw $0,36($fp)
sw $0,40($fp)
sb $0,24($fp)
sb $0,25($fp)
b $L30
nop

$L37:
lw $3,40($fp)
```

```
li $2,1 # 0x1
bne $3,$2,$L31
nop

lw $2,36($fp)
lb $3,0($2)
li $2,32 # 0x20
beq $3,$2,$L30
nop

$L31:
sw $0,44($fp)
lw $3,36($fp)
lw $2,40($fp)
addiu $2,$2,1
addiu $4,$fp,44
move $6,$4
move $5,$2
move $4,$3
lw $2,%got(get_numbers)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,get_numbers
1: jalr $25
nop

lw $28,16($fp)
sw $2,28($fp)
lw $2,28($fp)
sltu $2,$2,1
sb $2,24($fp)
lbu $2,24($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L32
nop
```

```
lw $2,44($fp)
move $5,$2
lw $4,28($fp)
lw $2,%call16(merge)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,merge
1: jalr $25
nop

lw $28,16($fp)
sw $2,32($fp)
lw $2,32($fp)
bne $2,$0,$L33
nop

lw $2,44($fp)
move $6,$2
lw $5,28($fp)
lw $4,60($fp)
lw $2,%got(write_vector)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,write_vector
1: jalr $25
nop

lw $28,16($fp)
lw $4,28($fp)
lw $2,%call16(free)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,free
1: jalr $25
nop

lw $28,16($fp)
b $L30
nop
```

```
$L33:
li $2,1 # 0x1
sb $2,25($fp)
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,70 # 0x46
li $5,1 # 0x1
lw $2,%got($LC16)($28)
addiu $4,$2,%lo($LC16)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
b $L30
nop

$L32:
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,69 # 0x45
li $5,1 # 0x1
lw $2,%got($LC17)($28)
addiu $4,$2,%lo($LC17)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
```

```
$L30:
lbu $2,24($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L36
nop

lbu $2,25($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L36
nop

addiu $3,$fp,40
addiu $2,$fp,36
lw $6,56($fp)
move $5,$3
move $4,$2
lw $2,%call16(getline)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,getline
1: jalr $25
nop

lw $28,16($fp)
sw $2,40($fp)
lw $3,40($fp)
li $2,-1 # 0xffffffffffffffff
bne $3,$2,$L37
nop

$L36:
lw $2,36($fp)
move $4,$2
lw $2,%call16(free)($28)
move $25,$2
```

```
.reloc 1f,R_MIPS_JALR,free
1: jalr $25
nop

lw $28,16($fp)
nop
move $sp,$fp
lw $31,52($sp)
lw $fp,48($sp)
addiu $sp,$sp,56
jr $31
nop

.set macro
.set reorder
.end read_file
.size read_file, .-read_file
.rdata
.align 2
$LC18:
.ascii "-h\000"
.align 2
$LC19:
.ascii "-v\000"
.align 2
$LC20:
.ascii "versi\303\263n TP1 Organizaci\303\263n de computadoras\000"
.align 2
$LC21:
.ascii "-i\000"
.align 2
$LC22:
.ascii "main.c\000"
.align 2
$LC23:
.ascii "in_file\000"
```

```
.align 2
$LC24:
.ascii "-\000"
.align 2
$LC25:
.ascii "r\000"
.align 2
$LC26:
.ascii "Error al abrir el archivo de entrada\000"
.align 2
$LC27:
.ascii "-o\000"
.align 2
$LC28:
.ascii "w+\000"
.align 2
$LC29:
.ascii "Error al abrir el archivo de salida\012\000"
.align 2
$LC30:
.ascii "Las opciones para el out command son '-o' o nada\012\000"
.align 2
$LC31:
.ascii "Parametro de entrada invalido\012\000"
.text
.align 2
.globl main
.set nomips16
.set nomicromips
.ent main
.type main, @function
main:
.frame $fp,64,$31 # vars= 32, regs= 2/0, args= 16, gp= 8
.mask 0xc0000000,-4
.fmask 0x00000000,0
.set noreorder
```

```
.cpload $25
.set nomacro
addiu $sp,$sp,-64
sw $31,60($sp)
sw $fp,56($sp)
move $fp,$sp
.cprestore 16
sw $4,64($fp)
sw $5,68($fp)
lw $2,%got(stdin)($28)
lw $2,0($2)
sw $2,24($fp)
lw $2,%got(stdout)($28)
lw $2,0($2)
sw $2,28($fp)
lw $2,68($fp)
lw $2,4($2)
sw $2,32($fp)
lw $2,32($fp)
beq $2,$0,$L39
nop

lw $2,%got($LC18)($28)
addiu $5,$2,%lo($LC18)
lw $4,32($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L40
nop

lw $2,%got(help)($28)
```

```
move $25,$2
.reloc 1f,R_MIPS_JALR,help
1: jalr $25
nop

lw $28,16($fp)
move $2,$0
b $L41
nop

$L40:
lw $2,%got($LC19)($28)
addiu $5,$2,%lo($LC19)
lw $4,32($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L42
nop

lw $2,%got($LC20)($28)
addiu $4,$2,%lo($LC20)
lw $2,%call16(puts)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,puts
1: jalr $25
nop

lw $28,16($fp)
move $2,$0
b $L41
nop
```

```
$L42:
lw $2,%got($LC21)($28)
addiu $5,$2,%lo($LC21)
lw $4,32($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L43
nop

lw $2,68($fp)
lw $2,8($2)
sw $2,36($fp)
lw $2,36($fp)
bne $2,$0,$L44
nop

lw $2,%got(__PRETTY_FUNCTION__.2719)($28)
addiu $7,$2,%lo(__PRETTY_FUNCTION__.2719)
li $6,154 # 0x9a
lw $2,%got($LC22)($28)
addiu $5,$2,%lo($LC22)
lw $2,%got($LC23)($28)
addiu $4,$2,%lo($LC23)
lw $2,%call16(__assert_fail)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,__assert_fail
1: jalr $25
nop

$L44:
```

```
lw $2,%got($LC24)($28)
addiu $5,$2,%lo($LC24)
lw $4,36($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L45
nop

lw $2,%got($LC25)($28)
addiu $5,$2,%lo($LC25)
lw $4,36($fp)
lw $2,%call16(fopen)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fopen
1: jalr $25
nop

lw $28,16($fp)
sw $2,24($fp)
lw $2,24($fp)
bne $2,$0,$L45
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,36 # 0x24
li $5,1 # 0x1
lw $2,%got($LC26)($28)
```

```
addiu $4,$2,%lo($LC26)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
li $2,-1 # 0xffffffffffffffff
b $L41
nop

$L45:
lw $2,68($fp)
lw $2,12($2)
sw $2,40($fp)
lw $2,68($fp)
lw $2,16($2)
sw $2,44($fp)
lw $2,%got($LC27)($28)
addiu $5,$2,%lo($LC27)
lw $4,40($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
1: jalr $25
nop

lw $28,16($fp)
sb $2,48($fp)
lw $2,%got($LC24)($28)
addiu $5,$2,%lo($LC24)
lw $4,44($fp)
lw $2,%got(strings_are_equal)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strings_are_equal
```

```
1: jalr $25
nop

lw $28,16($fp)
sb $2,49($fp)
lbu $2,48($fp)
beq $2,$0,$L46
nop

lbu $2,49($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
beq $2,$0,$L46
nop

lw $2,%got($LC28)($28)
addiu $5,$2,%lo($LC28)
lw $4,44($fp)
lw $2,%call16(fopen)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fopen
1: jalr $25
nop

lw $28,16($fp)
sw $2,28($fp)
lw $2,28($fp)
bne $2,$0,$L39
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,36 # 0x24
li $5,1 # 0x1
lw $2,%got($LC29)($28)
```

```
addiu $4,$2,%lo($LC29)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
li $2,-1 # 0xffffffffffffffff
b $L41
nop

$L46:
lbu $2,48($fp)
xori $2,$2,0x1
andi $2,$2,0x00ff
bne $2,$0,$L49
nop

lw $2,44($fp)
bne $2,$0,$L50
nop

$L49:
lw $2,%got(stdout)($28)
lw $2,0($2)
sw $2,28($fp)
b $L39
nop

$L50:
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,49 # 0x31
li $5,1 # 0x1
```

```
lw $2,%got($LC30)($28)
addiu $4,$2,%lo($LC30)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
li $2,-1 # 0xffffffffffffffff
b $L41
nop

$L43:
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,30 # 0x1e
li $5,1 # 0x1
lw $2,%got($LC31)($28)
addiu $4,$2,%lo($LC31)
lw $2,%call16(fwrite)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fwrite
1: jalr $25
nop

lw $28,16($fp)
li $2,-1 # 0xffffffffffffffff
b $L41
nop

$L39:
lw $5,28($fp)
lw $4,24($fp)
lw $2,%got(read_file)($28)
```

```
move $25,$2
.reloc 1f,R_MIPS_JALR,read_file
1: jalr $25
nop

lw $28,16($fp)
lw $2,%got(stdin)($28)
lw $2,0($2)
lw $3,24($fp)
beq $3,$2,$L51
nop

lw $4,24($fp)
lw $2,%call16(fclose)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fclose
1: jalr $25
nop

lw $28,16($fp)
$L51:
lw $2,%got(stdout)($28)
lw $2,0($2)
lw $3,28($fp)
beq $3,$2,$L52
nop

lw $4,28($fp)
lw $2,%call16(fclose)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fclose
1: jalr $25
nop

lw $28,16($fp)
$L52:
```

```
move $2,$0
$L41:
move $sp,$fp
lw $31,60($sp)
lw $fp,56($sp)
addiu $sp,$sp,64
jr $31
nop

.set macro
.set reorder
.end main
.size main, .-main
.rdata
.align 2
.type __PRETTY_FUNCTION__.2719, @object
.size __PRETTY_FUNCTION__.2719, 5
__PRETTY_FUNCTION__.2719:
.ascii "main\000"
.ident "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```