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. Esta función devuelve la dirección donde comienza en array temporal para luego liberarlo.

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. Max_size: Tamaño del array original.

Y devuelve uno solo, que es la dirección al array temporal para luego liberar la memoria.

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, 2 como auxiliares para los distintos valores y 2 para manejar las direcciones de los arrays. Todo esto se suma a los valores recibidos de la llamada a la función. La siguiente tabla sirvió de guia para poner orden, evitar pisar valores y finalmente lograr el correcto funcionamiento de la función.

Finalmente esta función devuelve la dirección del array temporal, que es la misma que devuelve merge_sort para luego poder liberar la memoria.

t0	L1
t1	R1
t2	L2
t3	R2
t4	Max size
t5	i
t6	j
t7	k
t8	Auxiliar de array original
t9	Auxiliar de array temporal
s0	Dirección de comienzo de array original
s1	Dirección de comienzo de array temporal

Cuadro 1: Tabla de seguimiento de registros utilizados.

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. Código fuente en lenguaje C

MERGE.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>


void merge(int* vector, int len_vector);
```

```
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");
}
```

```
int* to_vector(char* string,int* len){
int len_string = 0;
for(int j = 0; (string[j] != '\n');j++){
if(string[j] == ' '){
            len_string++;
        }
else if( (isdigit(string[j])==0 && (string[j]!='-'))  || ((string[j]==
            fprintf( stderr, "ERROR: Uno o más datos del archivo no so
            exit(EXIT_FAILURE);
        }
}
int* vector = malloc (sizeof(int)*(len_string));
if (!vector){
        fprintf(stderr, "ERROR: No se pudo reservar la memoria suficie
        exit(EXIT_FAILURE);
}
const char sep[2] = " ";
char *temp;
```

```
temp = strtok(string, sep);
int i = 0;
while(strcmp(temp,"\n")!=0){
vector[i] = atoi(temp);
i++;
temp = strtok(NULL, sep);
if (!temp){
            break;
        }
}
    *len=i;
return vector;
}


void print_vector(int* vector,int* len){
    int i=0;
    while (i<*len){
        printf("%i ", vector[i]);
        i++;
    }
    printf("\n");
}


void write_file(int* vector,FILE* dfile,int*len){
    int i=0;
    while(i<*len){
        char numero [10];
        sprintf(numero, "%i", vector[i]);
        fputs(numero,dfile);
        fputs(" ",dfile);
        i++;
    }
    fputs("\n",dfile);
}
```

```
void process_line(char* line,FILE* dfile){
    int* len = malloc(sizeof(int));
    if(!len){
        fprintf( stderr, "ERROR: no se pudo reservar memoria para alma
        exit(EXIT_FAILURE);
        }
    int* vector = to_vector(line,len);
    //verifica si la linea esta vacía
    if(*len == 1 && vector[0] == 0){
     fprintf( stderr, "ERROR: se insertó una línea vacía\n");
        exit(EXIT_FAILURE);
    }
    if (*len>0){
     merge(vector, *len);
    }
    if (dfile==NULL){
        print_vector(vector,len);
    }
    else{
        write_file(vector,dfile,len);
    }
    free(vector);
    free(len);
}



char* read_line(FILE* file){
    int tam=100;
    char* line= malloc(sizeof(char)*tam);
    if (!line){
        fprintf( stderr, "ERROR: No se pudo reservar memoria suficente
        exit(EXIT_FAILURE);
    }
```

```
    char c;
    int pos=0;
    int count=1;
    c = fgetc(file);
    if (c==EOF){
        return NULL;
    }
    while (c!='\n'){
        if (count==100){
            tam=tam+100;
            line=realloc(line,tam*sizeof(char));
            if (!line){
                fprintf( stderr, "ERROR: No se pudo almacenar la linea
                exit(EXIT_FAILURE);
            }
            count=1;
        }
        line[pos]=c;
        pos++;
        count++;
        c = fgetc(file);
    }
    line[pos]=c;
    return line;
}



void process_file(char* filename, char* destination){
    FILE* dfile=NULL;
    if (strcmp(destination,"Unspecified") != 0){
        dfile= fopen(destination,"w+");
        if (!dfile){
            fprintf( stderr, "ERROR: no se ha podido crear el archivo
            exit(EXIT_FAILURE);
```

```
        }
    }
    FILE* file;
    if (strcmp(filename,"stdin") == 0){
        file=stdin;
    }
    else{
        file = fopen(filename,"r");
    }
    if (!file){
        fprintf( stderr, "ERROR: no se encuentra el archivo de entrada
        exit(EXIT_FAILURE);
    }
    char*line= read_line(file);
    while(line!=NULL){
        process_line(line,dfile);
        free(line);
        line= read_line(file);
    }
    if (strcmp(filename,"stdin") != 0){
        fclose(file);
    }
    if (strcmp(destination,"Unspecified") != 0){
        fclose(dfile);
    }
}




int main(int argc, char *argv[]){

    //stdin salida por pantalla
    if(!argv[1]){
        process_file("stdin","Unspecified");
    }
```

```
    //stdin salida por archivo
    else if (strcmp(argv[1],"-o") == 0){
        if (!argv[2]){
        fprintf( stderr, "ERROR: ingrese la ruta del archivo de salida
        exit(EXIT_FAILURE);
        }
        process_file("stdin",argv[2]);
    }

    // version
else if(strcmp(argv[1],"-V") == 0){
printf("versión TP1 Organización de computadoras\n");
return 0;
}

//help
else if(strcmp(argv[1],"-h") == 0){
help();
return 0;
}

//entrada por archivo y salida por pantalla o por archivo
else if(strcmp(argv[1],"-i") == 0){
         if(!argv[2]){
            fprintf( stderr, "ERROR: ingrese la ruta del archivo de en
            exit(EXIT_FAILURE);
         }

         if (!argv[3]){
            process_file(argv[2],"Unspecified");
         }
 else if(strcmp(argv[3],"-o") == 0){
            if (!argv[4]){
                fprintf( stderr, "ERROR: ingrese la ruta del archivo d
                exit(EXIT_FAILURE);
```

```
            }
            process_file(argv[2],argv[4]);
}

}

else{
        fprintf( stderr, "ERROR: uno o más parámetros son incorrectos\
        exit(EXIT_FAILURE);

}

return 0;
}
```

7. 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

merge:
.frame fp,40,ra
subu sp,sp,40
.cprestore 24
sw ra,32(sp)
sw fp,28(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
sw v0,20(fp) #guardo puntero al temporal al liberar

free_memory:
lw a0,20(fp) #cargo el temporal
jal myfree

fin:
lw ra,32(sp)
lw fp,28(fp)
addu sp,sp,40

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


merge_sub_arrays:
.frame fp,40,ra
subu sp,sp,40
sw ra,32(sp)
sw fp,28(sp)
.cprestore 24
move fp,sp

#Parametros
sw a0,40(fp) #array
sw a1,44(fp) #left index 1
sw a2,48(fp) #right index 1
sw a3,52(fp) #left index 2
lw t3,56(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 t4,60(sp) #max size: Hago lo mismo. 40+20

sw t3,56(fp) #guardo right index
sw t4,60(fp) #guardo max size


create_temporal_array:
lw a0,60(fp) #a0: max size
sll a0,a0,2 #sizex4
jal mymalloc
```

```
sw v0,20(fp) #temporal array


load_variables:
lw s0,40(fp) #array original
lw s1,20(fp) #array temporal
lw t0,44(fp) #left index 1
lw t1,48(fp) #right index 1
lw t2,52(fp) #left index 2
lw t3,56(fp) #right index  2
lw t4,60(fp) #max size

sll t0,t0,2
sll t1,t1,2 #multiplico por 4 los indices.
sll t2,t2,2
sll t3,t3,2 #incluso el size
sll t4,t4,2


#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 t9,0 #auxiliar acceso a array temporal

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 s0,40(fp)
addu s0,s0,t5 #s0: dir de array[i]
lw t8,0(s0) #t8: array[i]

lw s0,40(fp) #s0: posicion inicial
addu s0,s0,t6 #s0: dir de array[j]
```

```
lw t9,0(s0) #t9: array[j]
lw s0,40(fp) #s0: posicion inicial
bge t8,t9,else_element_is_less_than  #condicion

if_element_is_less_than: #if(a[i] < a[j])
lw s1,20(fp)
addu s1,s1,t7 #s1: dir de temp[k]
sw t8,0(s1) #temp[k] = array[i]
addiu t7,t7,4
addiu t5,t5,4
lw s1,20(fp) #s1: posicion inicial
b while_elements_in_both_arrays

else_element_is_less_than:
lw s1,20(fp)
addu s1,s1,t7 #s1:dir temp[k]
sw t9,0(s1) #temp[k] = array[j]
addiu t7,t7,4
addiu t6,t6,4
lw s1,20(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 s1,20(fp) #cargo temporal
addu s1,s1,t7 #s1: dir de temp[k]

lw s0,40(fp) #cargo array
addu s0,s0,t5 #dir de array[i]
lw t8,0(s0) #t8: array[i]
sw t8,0(s1) #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 s1,20(fp) #cargo temporal
addu s1,s1,t7 #s1: dir de temp[k]

lw s0,40(fp) #s0: array
addu s0,s0,t6 #s0: dir de array[j]
lw t8,0(s0) #t8: array[j]
sw t8,0(s1) #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,return_merge_sub_arrays
lw s0,40(fp) #cargo array original
lw s1,20(fp) #cargo temporal
addu s0,s0,t5 #dir de array[i]
addu s1,s1,t6 #dir de temp[j]
lw t9,0(s1) #temp[j]
sw t9,0(s0) #array[i] = temp[j]
addiu t5,t5,4
addiu t6,t6,4
b while_i_less_or_equal_than_R2

return_merge_sub_arrays:
lw v0,20(fp)
lw ra,32(sp)
lw fp,28(sp)
```

```
addiu sp,sp,40
j ra
.end merge_sub_arrays


.globl merge_sort
.ent merge_sort

merge_sort:  #(a0:array[size], a1:left index, a2:right index)
.frame fp,40,ra
subu  sp,sp,40
sw ra,32(sp)
sw fp,28(sp)
.cprestore 24
move fp,sp

sw a0,40(fp) #array
sw a1,44(fp) #left index
sw a2,48(fp) #right index
sw a3,52(fp) #max size

lw t0,40(fp)
lw t1,44(fp)
lw t2,48(fp)
lw t3,52(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 t4,56(fp) #medio

left_recursion:
lw a0,40(fp) #array
lw a1,44(fp) #left
lw a2,56(fp) #medio
```

```
jal merge_sort

right_recursion:
lw a0,40(fp) #array
lw a1,56(fp) #medio
addiu a1,a1,1 #medio+1
lw a2,48(fp) #right index
jal merge_sort

merging_sub_arrays:
lw a0,40(fp) #array
lw a1,44(fp) #left index
lw  a2,56(fp) #medio
addiu a3,a2,1 #medio +1

lw t0,48(fp) #j por stack
sw t0,16(sp) #Lo mando por stack
lw t1,52(fp) #t1: max_size
sw t1,20(sp) #lo mando por stack

jal  merge_sub_arrays

return_merge_sort:
lw ra,32(sp)
lw fp,28(sp)
addiu sp,sp,40
j ra

.end merge_sort
```

8. Código MIPS32 generado por el compilador

```
.file 1 "merge.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
.rdata
.align 2
$LC12:
.ascii "ERROR: Uno o m\303\241s datos del archivo no son v\303\241"
.ascii "lidos, por ejemeplo el simbolo %c no se corresponde con "
.ascii "un n\303\272mero.\012\000"
.align 2
$LC13:
.ascii "ERROR: No se pudo reservar la memoria suficiente.\012\000"
.align 2
$LC14:
.ascii "\012\000"
.text
.align 2
.globl to_vector
.set nomips16
.set nomicromips
.ent to_vector
.type to_vector, @function
to_vector:
.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 $0,24($fp)
sw $0,28($fp)
b $L3
nop

$L8:
lw $2,28($fp)
lw $3,64($fp)
addu $2,$3,$2
lb $3,0($2)
li $2,32 # 0x20
bne $3,$2,$L4
nop

lw $2,24($fp)
addiu $2,$2,1
sw $2,24($fp)
b $L5
nop

$L4:
lw $2,%call16(__ctype_b_loc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,__ctype_b_loc
1: jalr $25
nop

lw $28,16($fp)
lw $3,0($2)
lw $2,28($fp)
lw $4,64($fp)
addu $2,$4,$2
lb $2,0($2)
sll $2,$2,1
addu $2,$3,$2
```

```
lhu $2,0($2)
andi $2,$2,0x8
bne $2,$0,$L6
nop

lw $2,28($fp)
lw $3,64($fp)
addu $2,$3,$2
lb $3,0($2)
li $2,45 # 0x2d
bne $3,$2,$L7
nop

$L6:
lw $2,28($fp)
lw $3,64($fp)
addu $2,$3,$2
lb $3,0($2)
li $2,45 # 0x2d
bne $3,$2,$L5
nop

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

lw $28,16($fp)
lw $3,0($2)
lw $2,28($fp)
addiu $2,$2,1
lw $4,64($fp)
addu $2,$4,$2
lb $2,0($2)
sll $2,$2,1
```

```
addu $2,$3,$2
lhu $2,0($2)
andi $2,$2,0x8
bne $2,$0,$L5
nop

$L7:
lw $2,%got(stderr)($28)
lw $4,0($2)
lw $2,28($fp)
lw $3,64($fp)
addu $2,$3,$2
lb $2,0($2)
move $6,$2
lw $2,%got($LC12)($28)
addiu $5,$2,%lo($LC12)
lw $2,%call16(fprintf)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fprintf
1: jalr $25
nop

lw $28,16($fp)
li $4,1 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L5:
lw $2,28($fp)
addiu $2,$2,1
sw $2,28($fp)
$L3:
lw $2,28($fp)
```

```
lw $3,64($fp)
addu $2,$3,$2
lb $3,0($2)
li $2,10 # 0xa
bne $3,$2,$L8
nop

lw $2,24($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 $2,40($fp)
lw $2,40($fp)
bne $2,$0,$L9
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,50 # 0x32
li $5,1 # 0x1
lw $2,%got($LC13)($28)
addiu $4,$2,%lo($LC13)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L9:
li $2,8192 # 0x2000
sh $2,44($fp)
addiu $2,$fp,44
move $5,$2
lw $4,64($fp)
lw $2,%call16(strtok)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strtok
1: jalr $25
nop

lw $28,16($fp)
sw $2,32($fp)
sw $0,36($fp)
b $L10
nop

$L12:
lw $2,36($fp)
sll $2,$2,2
lw $3,40($fp)
addu $16,$3,$2
lw $4,32($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 $2,36($fp)
addiu $2,$2,1
sw $2,36($fp)
addiu $2,$fp,44
move $5,$2
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 $2,32($fp)
lw $2,32($fp)
beq $2,$0,$L14
nop

$L10:
lw $2,%got($LC14)($28)
addiu $5,$2,%lo($LC14)
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)
bne $2,$0,$L12
nop

b $L11
```

```
nop

$L14:
nop
$L11:
lw $2,68($fp)
lw $3,36($fp)
sw $3,0($2)
lw $2,40($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 to_vector
.size to_vector, .-to_vector
.rdata
.align 2
$LC15:
.ascii "%i \000"
.text
.align 2
.globl print_vector
.set nomips16
.set nomicromips
.ent print_vector
.type print_vector, @function
print_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 $0,24($fp)
b $L16
nop

$L17:
lw $2,24($fp)
sll $2,$2,2
lw $3,40($fp)
addu $2,$3,$2
lw $2,0($2)
move $5,$2
lw $2,%got($LC15)($28)
addiu $4,$2,%lo($LC15)
lw $2,%call16(printf)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,printf
1: jalr $25
nop

lw $28,16($fp)
lw $2,24($fp)
addiu $2,$2,1
sw $2,24($fp)
$L16:
lw $2,44($fp)
lw $3,0($2)
```

```
lw $2,24($fp)
slt $2,$2,$3
bne $2,$0,$L17
nop

li $4,10 # 0xa
lw $2,%call16(putchar)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,putchar
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 print_vector
.size print_vector, .-print_vector
.rdata
.align 2
$LC16:
.ascii "%i\000"
.text
.align 2
.globl write_file
.set nomips16
.set nomicromips
.ent write_file
.type write_file, @function
```

```
write_file:
.frame $fp,48,$31 # vars= 16, regs= 2/0, args= 16, gp= 8
.mask 0xc0000000,-4
.fmask 0x00000000,0
.set noreorder
.cpload $25
.set nomacro
addiu $sp,$sp,-48
sw $31,44($sp)
sw $fp,40($sp)
move $fp,$sp
.cprestore 16
sw $4,48($fp)
sw $5,52($fp)
sw $6,56($fp)
sw $0,24($fp)
b $L19
nop

$L20:
lw $2,24($fp)
sll $2,$2,2
lw $3,48($fp)
addu $2,$3,$2
lw $2,0($2)
addiu $3,$fp,28
move $6,$2
lw $2,%got($LC16)($28)
addiu $5,$2,%lo($LC16)
move $4,$3
lw $2,%call16(sprintf)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,sprintf
1: jalr $25
nop
```

```
lw $28,16($fp)
addiu $2,$fp,28
lw $5,52($fp)
move $4,$2
lw $2,%call16(fputs)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fputs
1: jalr $25
nop

lw $28,16($fp)
lw $5,52($fp)
li $4,32 # 0x20
lw $2,%call16(fputc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fputc
1: jalr $25
nop

lw $28,16($fp)
lw $2,24($fp)
addiu $2,$2,1
sw $2,24($fp)
$L19:
lw $2,56($fp)
lw $3,0($2)
lw $2,24($fp)
slt $2,$2,$3
bne $2,$0,$L20
nop

lw $5,52($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 $31,44($sp)
lw $fp,40($sp)
addiu $sp,$sp,48
jr $31
nop

.set macro
.set reorder
.end write_file
.size write_file, .-write_file
.rdata
.align 2
$LC17:
.ascii "ERROR: no se pudo reservar memoria para almacenar la lon"
.ascii "gitud del arreglo\000"
.align 2
$LC18:
.ascii "ERROR: se insert\303\263 una l\303\255nea vac\303\255a\012"
.ascii "\000"
.text
.align 2
.globl process_line
.set nomips16
.set nomicromips
.ent process_line
.type process_line, @function
process_line:
.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)
li $4,4 # 0x4
lw $2,%call16(malloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,malloc
1: jalr $25
nop

lw $28,16($fp)
sw $2,24($fp)
lw $2,24($fp)
bne $2,$0,$L22
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,73 # 0x49
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)
li $4,1 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L22:
lw $5,24($fp)
lw $4,40($fp)
lw $2,%got(to_vector)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,to_vector
1: jalr $25
nop

lw $28,16($fp)
sw $2,28($fp)
lw $2,24($fp)
lw $3,0($2)
li $2,1 # 0x1
bne $3,$2,$L23
nop

lw $2,28($fp)
lw $2,0($2)
bne $2,$0,$L23
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,37 # 0x25
li $5,1 # 0x1
lw $2,%got($LC18)($28)
```

```
addiu $4,$2,%lo($LC18)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L23:
lw $2,24($fp)
lw $2,0($2)
blez $2,$L24
nop

lw $2,24($fp)
lw $2,0($2)
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)
$L24:
lw $2,44($fp)
bne $2,$0,$L25
nop
```

```
lw $5,24($fp)
lw $4,28($fp)
lw $2,%got(print_vector)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,print_vector
1: jalr $25
nop

lw $28,16($fp)
b $L26
nop

$L25:
lw $6,24($fp)
lw $5,44($fp)
lw $4,28($fp)
lw $2,%got(write_file)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,write_file
1: jalr $25
nop

lw $28,16($fp)
$L26:
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)
lw $4,24($fp)
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 $31,36($sp)
lw $fp,32($sp)
addiu $sp,$sp,40
jr $31
nop

.set macro
.set reorder
.end process_line
.size process_line, .-process_line
.rdata
.align 2
$LC19:
.ascii "ERROR: No se pudo reservar memoria suficente para la lec"
.ascii "tura de la linea\000"
.align 2
$LC20:
.ascii "ERROR: No se pudo almacenar la linea correctamente\000"
.text
.align 2
.globl read_line
.set nomips16
.set nomicromips
.ent read_line
.type read_line, @function
read_line:
.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)
li $2,100 # 0x64
sw $2,24($fp)
lw $2,24($fp)
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 $2,28($fp)
lw $2,28($fp)
bne $2,$0,$L28
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,72 # 0x48
li $5,1 # 0x1
lw $2,%got($LC19)($28)
addiu $4,$2,%lo($LC19)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L28:
sw $0,36($fp)
li $2,1 # 0x1
sw $2,40($fp)
lw $4,56($fp)
lw $2,%call16(fgetc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fgetc
1: jalr $25
nop

lw $28,16($fp)
sb $2,32($fp)
lb $3,32($fp)
li $2,-1 # 0xffffffffffffffff
bne $3,$2,$L31
nop

move $2,$0
b $L30
nop

$L34:
lw $3,40($fp)
li $2,100 # 0x64
bne $3,$2,$L32
```

```
nop

lw $2,24($fp)
addiu $2,$2,100
sw $2,24($fp)
lw $2,24($fp)
move $5,$2
lw $4,28($fp)
lw $2,%call16(realloc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,realloc
1: jalr $25
nop

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

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,50 # 0x32
li $5,1 # 0x1
lw $2,%got($LC20)($28)
addiu $4,$2,%lo($LC20)
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 # 0x1
lw $2,%call16(exit)($28)
```

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

$L33:
li $2,1 # 0x1
sw $2,40($fp)
$L32:
lw $2,36($fp)
lw $3,28($fp)
addu $2,$3,$2
lbu $3,32($fp)
sb $3,0($2)
lw $2,36($fp)
addiu $2,$2,1
sw $2,36($fp)
lw $2,40($fp)
addiu $2,$2,1
sw $2,40($fp)
lw $4,56($fp)
lw $2,%call16(fgetc)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,fgetc
1: jalr $25
nop

lw $28,16($fp)
sb $2,32($fp)
$L31:
lb $3,32($fp)
li $2,10 # 0xa
bne $3,$2,$L34
nop

lw $2,36($fp)
```

```
lw $3,28($fp)
addu $2,$3,$2
lbu $3,32($fp)
sb $3,0($2)
lw $2,28($fp)
$L30:
move $sp,$fp
lw $31,52($sp)
lw $fp,48($sp)
addiu $sp,$sp,56
jr $31
nop

.set macro
.set reorder
.end read_line
.size read_line, .-read_line
.rdata
.align 2
$LC21:
.ascii "Unspecified\000"
.align 2
$LC22:
.ascii "w+\000"
.align 2
$LC23:
.ascii "ERROR: no se ha podido crear el archivo en la ruta espec"
.ascii "ificada\012\000"
.align 2
$LC24:
.ascii "stdin\000"
.align 2
$LC25:
.ascii "r\000"
.align 2
$LC26:
```

```
.ascii "ERROR: no se encuentra el archivo de entrada\012\000"
.text
.align 2
.globl process_file
.set nomips16
.set nomicromips
.ent process_file
.type process_file, @function
process_file:
.frame $fp,48,$31 # vars= 16, regs= 2/0, args= 16, gp= 8
.mask 0xc0000000,-4
.fmask 0x00000000,0
.set noreorder
.cpload $25
.set nomacro
addiu $sp,$sp,-48
sw $31,44($sp)
sw $fp,40($sp)
move $fp,$sp
.cprestore 16
sw $4,48($fp)
sw $5,52($fp)
sw $0,24($fp)
lw $2,%got($LC21)($28)
addiu $5,$2,%lo($LC21)
lw $4,52($fp)
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L36
nop
```

```
lw $2,%got($LC22)($28)
addiu $5,$2,%lo($LC22)
lw $4,52($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,$0,$L36
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,64 # 0x40
li $5,1 # 0x1
lw $2,%got($LC23)($28)
addiu $4,$2,%lo($LC23)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop
```

```
$L36:
lw $2,%got($LC24)($28)
addiu $5,$2,%lo($LC24)
lw $4,48($fp)
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
bne $2,$0,$L37
nop

lw $2,%got(stdin)($28)
lw $2,0($2)
sw $2,28($fp)
b $L38
nop

$L37:
lw $2,%got($LC25)($28)
addiu $5,$2,%lo($LC25)
lw $4,48($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)
$L38:
lw $2,28($fp)
bne $2,$0,$L39
nop
```

```
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,45 # 0x2d
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 $4,1 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L39:
lw $4,28($fp)
lw $2,%got(read_line)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,read_line
1: jalr $25
nop

lw $28,16($fp)
sw $2,32($fp)
b $L40
nop

$L41:
```

```
lw $5,24($fp)
lw $4,32($fp)
lw $2,%got(process_line)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,process_line
1: jalr $25
nop

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

lw $28,16($fp)
sw $2,32($fp)
$L40:
lw $2,32($fp)
bne $2,$0,$L41
nop

lw $2,%got($LC24)($28)
addiu $5,$2,%lo($LC24)
lw $4,48($fp)
lw $2,%call16(strcmp)($28)
move $25,$2
```

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

lw $28,16($fp)
beq $2,$0,$L42
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)
$L42:
lw $2,%got($LC21)($28)
addiu $5,$2,%lo($LC21)
lw $4,52($fp)
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
beq $2,$0,$L44
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)
$L44:
nop
move $sp,$fp
lw $31,44($sp)
lw $fp,40($sp)
addiu $sp,$sp,48
jr $31
nop

.set macro
.set reorder
.end process_file
.size process_file, .-process_file
.rdata
.align 2
$LC27:
.ascii "-o\000"
.align 2
$LC28:
.ascii "ERROR: ingrese la ruta del archivo de salida\012\000"
.align 2
$LC29:
.ascii "-V\000"
.align 2
$LC30:
.ascii "versi\303\263n TP1 Organizaci\303\263n de computadoras\000"
.align 2
$LC31:
.ascii "-h\000"
.align 2
$LC32:
.ascii "-i\000"
.align 2
$LC33:
```

```
.ascii "ERROR: ingrese la ruta del archivo de entrada\012\000"
.align 2
$LC34:
.ascii "ERROR: uno o m\303\241s par\303\241metros son incorrecto"
.ascii "s\012\000"
.text
.align 2
.globl main
.set nomips16
.set nomicromips
.ent main
.type main, @function
main:
.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 $2,36($fp)
addiu $2,$2,4
lw $2,0($2)
bne $2,$0,$L46
nop

lw $2,%got($LC21)($28)
addiu $5,$2,%lo($LC21)
lw $2,%got($LC24)($28)
addiu $4,$2,%lo($LC24)
```

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

lw $28,16($fp)
b $L47
nop

$L46:
lw $2,36($fp)
addiu $2,$2,4
lw $3,0($2)
lw $2,%got($LC27)($28)
addiu $5,$2,%lo($LC27)
move $4,$3
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
bne $2,$0,$L48
nop

lw $2,36($fp)
addiu $2,$2,8
lw $2,0($2)
bne $2,$0,$L49
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
```

```
li $6,45 # 0x2d
li $5,1 # 0x1
lw $2,%got($LC28)($28)
addiu $4,$2,%lo($LC28)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L49:
lw $2,36($fp)
addiu $2,$2,8
lw $2,0($2)
move $5,$2
lw $2,%got($LC24)($28)
addiu $4,$2,%lo($LC24)
lw $2,%got(process_file)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,process_file
1: jalr $25
nop

lw $28,16($fp)
b $L47
nop

$L48:
```

```
lw $2,36($fp)
addiu $2,$2,4
lw $3,0($2)
lw $2,%got($LC29)($28)
addiu $5,$2,%lo($LC29)
move $4,$3
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
bne $2,$0,$L50
nop

lw $2,%got($LC30)($28)
addiu $4,$2,%lo($LC30)
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 $L51
nop

$L50:
lw $2,36($fp)
addiu $2,$2,4
lw $3,0($2)
lw $2,%got($LC31)($28)
addiu $5,$2,%lo($LC31)
move $4,$3
```

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

lw $28,16($fp)
bne $2,$0,$L52
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 $L51
nop

$L52:
lw $2,36($fp)
addiu $2,$2,4
lw $3,0($2)
lw $2,%got($LC32)($28)
addiu $5,$2,%lo($LC32)
move $4,$3
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
bne $2,$0,$L53
```

```
nop

lw $2,36($fp)
addiu $2,$2,8
lw $2,0($2)
bne $2,$0,$L54
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,46 # 0x2e
li $5,1 # 0x1
lw $2,%got($LC33)($28)
addiu $4,$2,%lo($LC33)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L54:
lw $2,36($fp)
addiu $2,$2,12
lw $2,0($2)
bne $2,$0,$L55
nop
```

```
lw $2,36($fp)
addiu $2,$2,8
lw $3,0($2)
lw $2,%got($LC21)($28)
addiu $5,$2,%lo($LC21)
move $4,$3
lw $2,%got(process_file)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,process_file
1: jalr $25
nop

lw $28,16($fp)
b $L47
nop

$L55:
lw $2,36($fp)
addiu $2,$2,12
lw $3,0($2)
lw $2,%got($LC27)($28)
addiu $5,$2,%lo($LC27)
move $4,$3
lw $2,%call16(strcmp)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,strcmp
1: jalr $25
nop

lw $28,16($fp)
bne $2,$0,$L47
nop

lw $2,36($fp)
addiu $2,$2,16
lw $2,0($2)
```

```
bne $2,$0,$L57
nop

lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,45 # 0x2d
li $5,1 # 0x1
lw $2,%got($LC28)($28)
addiu $4,$2,%lo($LC28)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L57:
lw $2,36($fp)
addiu $2,$2,8
lw $3,0($2)
lw $2,36($fp)
addiu $2,$2,16
lw $2,0($2)
move $5,$2
move $4,$3
lw $2,%got(process_file)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,process_file
```

```
1: jalr $25
nop

lw $28,16($fp)
b $L47
nop

$L53:
lw $2,%got(stderr)($28)
lw $2,0($2)
move $7,$2
li $6,46 # 0x2e
li $5,1 # 0x1
lw $2,%got($LC34)($28)
addiu $4,$2,%lo($LC34)
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 # 0x1
lw $2,%call16(exit)($28)
move $25,$2
.reloc 1f,R_MIPS_JALR,exit
1: jalr $25
nop

$L47:
move $2,$0
$L51:
move $sp,$fp
lw $31,28($sp)
lw $fp,24($sp)
addiu $sp,$sp,32
```

```
jr $31
nop

.set macro
.set reorder
.end main
.size main, .-main
.ident "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```