



**FACULTAD
DE INGENIERIA**
Universidad de Buenos Aires

**Asignatura: (66.20) Organización de
Computadoras**

Curso: 1 (Martes)

Grupo: 1

TP N° 1: Programación MIPS¹

Nombre y apellido	Padrón	Correo electrónico
Bauni, Chiara	102981	cbauni@fi.uba.ar
Bilbao, Manuel	102732	mbilbao@fi.uba.ar
Stroia, Lautaro	100901	lstroia@fi.uba.ar

Primer cuatrimestre 2021

¹Link a repositorio: <https://github.com/chiabauni/OrgaDeCompus-TP1>

Índice

1. Enunciado	3
1.1. Objetivos	3
1.2. Alcance	3
1.3. Requisitos	3
1.4. Descripción	3
1.4.1. Ejemplos	4
1.5. Implementación	5
1.6. Informe	5
1.7. Regresiones	5
1.8. Entrega de TPs	6
2. Documentación relevante al diseño e implementación del programa	6
2.1. Explicación implementación $r0 = r2 \% r1$	6
3. Comando(s) para compilar el programa	6
4. Corridas de prueba	7
4.1. Comandos de ayuda	7
4.2. Comandos de versión	8
4.3. Pruebas	9
4.3.1. Prueba: leer información desde un archivo y guardar el resultado en otro	9
4.3.2. Prueba: información en un archivo, resultados por salida standard	10
4.3.3. Prueba: información por stdin, salida por archivo	11
4.3.4. Prueba: información por stdin, salida por stdout	11
5. Conclusiones	12
6. Código fuente	13
6.1. utils.h	13
6.2. main.c	14
6.3. max_divisor.h	16
6.4. max_divisor.c	18
6.5. gcd.S	22
7. Diagramas de Stack Frame	26
8. Código MIPS por el compilador	28
8.1. main.s	28
8.2. max_divisor.s	42

1. Enunciado

1.1. Objetivos

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

1.2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual y de carácter obligatorio para todos los 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 confeccionado de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

El trabajo y eventuales reentregas deberán ser presentadas exclusivamente a través del campus según las fechas preestablecidas para el mismo en la planificación del curso presentada en la primera clase del cuatrimestre. Asimismo todas las devoluciones se realizarán en horario de clase, en persona, para lo cual deberán estar presentes todos los integrantes.

1.4. Descripción

El programa a desarrollar deberá procesar un stream de texto compuesto por una cantidad arbitrariamente grande de líneas de longitud arbitraria.

La entrada del programa estará compuesta por una secuencia de pares de números enteros. Luego de leerlos, deberá calcular e imprimir el máximo común divisor (GCD) de cada par siguiendo los lineamientos indicados más abajo.

Por ejemplo, dado el siguiente flujo de entrada:

```
$ cat input.txt
60 45
90 9
17 13
```

Al ejecutar el programa la salida será:

```
$ tpl -i input.txt -o -
GCD(60, 45) = 15
GCD(90, 9) = 9
GCD(17, 13) = 1
```

1.4.1. 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: primero, veamos qué sucede cuando el archivo de entrada está vacío:

```
$ ./tp1 -o salida.txt </dev/null
$ ls -l salida.txt
-rw-r--r-- 1 leandro leandro 0 Oct 20 20:14 salida.txt
```

Aquí puede verse que el programa se comporta según lo esperado, ya que cuando la entrada está vacía, la salida lo estará también.

Veamos qué ocurre al ingresar un archivo con una única línea, la cual contiene un único par de valores:

```
$ echo 1 1 | ./tp1 -o -
1
```

Lo mismo debería ocurrir si la entrada se encuentra alojada en el sistema de archivos:

```
$ echo 1 1 >entrada.txt
$ ./tp1 -i entrada.txt -o -
1
```

1.5. Implementación

El programa a desarrollar constará de una mezcla entre código MIPS32 y C, siendo la parte escrita en assembly la encargada de calcular los GCDs utilizando el algoritmo de Euclides. El formato de dicha función será: **void euclides(struct gcd*, size_t);**

En donde el primer parámetro, de tipo struct gcd*, es un puntero a un arreglo de estructuras que describen los números de la entrada, mientras que el segundo parámetro es la cantidad de elementos presentes en el arreglo.

```
struct gcd {  
    int num_a; /* input */  
    int num_b; /* input */  
    int gcd_ab; /* output */  
};
```

De esta manera, con una única invocación a euclides(), el programa podrá calcular y retornar todos los GCDs usando el arreglo pasado por parámetro.

1.6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.
- Comando(s) para compilar el programa.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C y MIPS.
- El código MIPS32 generado por el compilador.
- Este enunciado.

1.7. Regresiones

El programa deberá pasar todas las regresiones definidas en el código fuente suministrado en este TP 1:

```
$ make  
cc -Wall -g -o regressions regressions.c gcd.c gcd.S  
:  
$ make test  
./regressions
```

Asimismo deberá usarse el modo 1 del sistema operativo para manejo de acceso no alineado a memoria.

1.8. Entrega de TPs

La entrega de este trabajo deberá realizarse usando el campus virtual de la materia dentro del plazo de tiempo preestablecido. Asimismo, en todos los casos, estas presentaciones deberán ser realizadas durante los días martes. El feedback estará disponible de un martes hacia el otro, para lo cual deberán estar presentes los integrantes de cada grupo tal como ocurre durante modalidad presencial de cursada.

Por otro lado, la última fecha de entrega y presentación para este trabajo será el martes 18/5.

2. Documentación relevante al diseño e implementación del programa

El objetivo de este primer trabajo práctico es familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, a través de la creación de un algoritmo que busca el Máximo Común Divisor (GCD) entre 2 números conocido como **Algoritmo de Euclides**.

Lo primero que decidimos hacer fue crear un programa, escrito completamente en lenguaje C, para leer un archivo o la entrada standard y de ese input parsear la información de sus líneas buscando el GCD entre los números leídos.

Luego, basándonos en el código de **gcd.c**, procedimos a crear el archivo **gcd.S** en Assembly MIPS32, respetando la ABI de MIPS. Con este archivo finalizado, procedimos a la compilación y ejecución del código en la máquina virtual **QEMU**, que corre un SO Debian con kernel Linux para asegurarnos de que todo funcionase como corresponde.

Por último, se realizaron varias pruebas: algunas provistas por la cátedra en el archivo **regressions.c** y otras pasándole un archivo con números vía entrada standard para asegurarnos que el programa funcione como corresponde.

2.1. Explicación implementación $r0 = r2 \% r1$

Para implementar esta parte del código C en MIPS, se escriben las siguientes instrucciones:

<pre>divu t1, t0 mfhi t2</pre>

Lo que hace `divu` es tomar el valor de palabra de 32 bits en el registro `t1` y se divide por el valor de 32 bits en el registro `t0`, tratando ambos operandos como valores sin signo. El cociente de 32 bits se coloca en el registro especial `LO` y el resto de 32 bits se coloca en el registro especial `HI`. Luego la instrucción `mfhi` toma el contenido del registro `HI` y se carga en `t2`. Guardando en el registro `t2` el resto de la división entre `t1` y `t0`.

3. Comando(s) para compilar el programa

En el repositorio podemos encontrar dos carpetas: Enunciado y Resolución.

En la carpeta Enunciado tenemos los archivos `gcd.c`, `gcd.h`, `regressions.c` y un `Makefile` provistos por la cátedra y el `gcd.S` completado por nosotros.

Para correr las pruebas del archivo provisto por la cátedra:

1. **make;**

2. `./regressions`

Por otro lado encontramos la carpeta Resolución en donde están: `main.c` y `max_divisor.c` creados por nosotros para el manejo de archivos y el mismo `gcd.S` de la carpeta Enunciado.

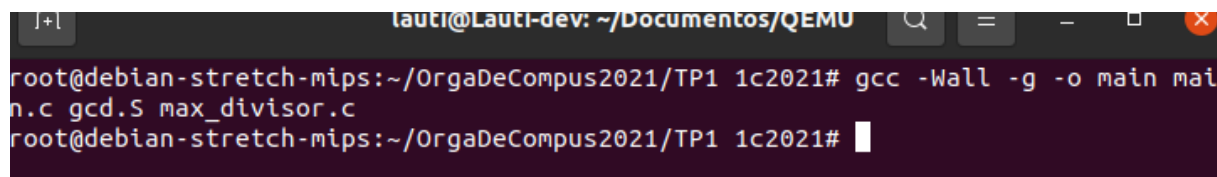
El comando para compilar el programa y obtener el ejecutable correctamente es:

1. `make`;
2. `./main -i tests/input.txt -o -`

También se puede correr el ejecutable con las distintas variaciones para ingresar el input y/o output.

4. Corridas de prueba

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 línea:



```
lauti@Lauti-dev: ~/Documentos/QEMU
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# gcc -Wall -g -o main main.c gcd.S max_divisor.c
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021#
```

4.1. Comandos de ayuda

La primera prueba es para el comando de ayuda expresado con el flag `'-h'`, el cual muestra por salida standard al usuario las distintas opciones que tiene para ejecutar el programa.



```
lauti@Lauti-dev: ~/Documentos/QEMU
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main -h
Comandos:
Uso:
ejecutable -h
ejecutable -V
ejecutable -i in_file -o out_file

Opciones:
-V, --version Imprime la version del programa y finaliza la ejecucion.
-h, --help Muestra esto por pantalla y finaliza la ejecucion.
-i, --input Especifica la ruta para el archivo de entrada, por default se usa stdin.
-o, --output Especifica la ruta para el archivo de salida, por default se usa stdout.

Ejemplos:
ejecutable < in.txt > out.txt
cat in.txt | ejecutable -i - > out.txt
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021#
```

La forma alternativa a este comando, es utilizar el flag '**-help**'.

```
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# gcc -Wall -g -o main mairoot@
debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main --help
Comandos:
Uso:
ejecutable -h
ejecutable -V
ejecutable -i in_file -o out_file

Opciones:
-V, --version Imprime la version del programa y finaliza la ejecucion.
-h, --help Muestra esto por pantalla y finaliza la ejecucion.
-i, --input Especifica la ruta para el archivo de entrada, por default se usa stdin.
-o, --output Especifica la ruta para el archivo de salida, por default se usa stdout.

Ejemplos:
ejecutable < in.txt > out.txt
cat in.txt | ejecutable -i - > out.txtroot@debian-stretch-mips:~/OrgaDeCompus2021/TP1
1c2021# █
```

4.2. Comandos de versión

Similarmente, el usuario es capaz de obtener la versión del programa mediante el flag '**-V**'.

```
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main -V
Organizacion de Computadoras TP1
1° cuatrimestre 2021
Entrega 1

Integrantes
Chiara Bauni 102981
Lautaro Stroia 100901
Manuel Bilbao 102732

Codigo disponible en : https://github.com/chiabauni/OrgaDeCompus-TP1root@debian-stret
ch-mips:~/OrgaDeCompus2021/TP1 1c2021# █
```

O también, puede utilizar el flag '**--version**'.

```
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main --version
Organizacion de Computadoras TP1
1° cuatrimestre 2021
Entrega 1

Integrantes
Chiara Bauni 102981
Lautaro Stroia 100901
Manuel Bilbao 102732

Codigo disponible en : https://github.com/chiabauni/OrgaDeCompus-TP1root@debian-stret
ch-mips:~/OrgaDeCompus2021/TP1 1c2021# █
```


Y al abrir el archivo `output.txt` vemos lo siguiente:

[illegible]

4.3.2. Prueba: información en un archivo, resultados por salida standard

Ahora vamos a mostrar el funcionamiento del programa recibiendo el mismo archivo de entrada, `input.txt`, y devolviendo el resultado por `stdout`.

```
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main -i input.txt -o -
GCD(1, 1) = 1
GCD(7, 1) = 1
GCD(13, 13) = 13
GCD(100, 10) = 10
GCD(100, 60) = 20
GCD(50, 75) = 25
GCD(60, 45) = 15
GCD(90, 9) = 9
GCD(17, 13) = 1
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021#
```

4.3.3. Prueba: información por stdin, salida por archivo

Mostraremos el funcionamiento del programa escribiendo pares de números por entrada std, y guardando el resultado en un archivo de salida.

```
xtot@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main -i - -o output2.txt
1 5
4 3
100 25

root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# vim output2.txt
```

[illegible]

4.3.4. Prueba: información por stdin, salida por stdout

Finalmente, mostramos por salida std el resultado del programa ingresando pares de números por entrada std.

```
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021# ./main -i - -o -
175 6
190 3
45 4

GCD(175, 6) = 1
GCD(190, 3) = 1
GCD(45, 4) = 1
root@debian-stretch-mips:~/OrgaDeCompus2021/TP1 1c2021#
```

5. Conclusiones

Durante la resolución del problema planteado nos fue de ayuda escribir el código MIPS lo más claro posible ayudándonos de la estructura proporcionada por el gcd.c. Otra herramienta muy útil fueron los diagramas del stack frame que nos permitieron visualizar mejor en qué sector del stack frame debería encontrarse cada registro. En conclusión, logramos familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI, logrando de esta manera desarrollar un programa que cumple con lo pedido por la consigna.

6. Código fuente

6.1. utils.h

Este archivo contiene la definición de constantes y funciones que van a ser utilizadas en el archivo principal.

```
1  #ifndef UTILS_H
2  #define UTILS_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdbool.h>
7  #include <string.h>
8  #include <getopt.h>
9
10 #define MAX_MENSAJE 150
11 #define EOL '\n'
12 #define ERROR 1
13 #define EXITO 0
14
15 #define COMANDO_AYUDA_CORTO 'h'
16 #define COMANDO_VERSION_CORTO 'V'
17 #define COMANDO_AYUDA_LARGO "help"
18 #define COMANDO_VERSION_LARGO "version"
19 #define COMANDO_INPUT_CORTO 'i'
20 #define COMANDO_OUTPUT_CORTO 'o'
21 #define COMANDO_INPUT_LARGO "input"
22 #define COMANDO_OUTPUT_LARGO "output"
23
24 #define RUTA_AYUDA "comandos/help.txt"
25 #define RUTA_VERSION "comandos/version.txt"
26
27 #define MENSAJE_COMANDO_INVALIDO_ERROR "\n El comando usado es invalido, use
    ↪ -h para ayuda.\n"
28 #define MENSAJE_GCD_ERROR "\n No se pudo obtener el maximo comun divisor,
    ↪ ocurrio un error.\n"
29 #define MENSAJE_MEM_DINAMICA_ERROR "\n Problema asignando memoria dinamica.\n"
30 #define MENSAJE_INPUT_ERROR "\nLa cantidad de numeros ingresada es invalida.
    ↪ Solo se pueden ingresar 2 numeros por cada calculo de GCD.\n"
31 #define MENSAJE_CONVERTIR_CHAR_ERROR "\nError a la hora de convertir la linea
    ↪ de caracteres.\n"
32 #define MENSAJE_LINEA_INVALIDA_ERROR "\nProblema leyendo el archivo.\n"
33 #define MENSAJE_IMPRIMIR_SALIDA_ERROR "\nProblema al imprimir la salida.\n"
34
35 #endif //UTILS_H
```

6.2. main.c

Junto a **max_divisor.c** son los archivos principales del programa. Permiten manejar archivos de entrada y salida, tanto los standard como de archivos ingresados por stdin, y extraer información necesaria de su interior, para calcular el GCD.

```
1  #include "utils.h"
2  #include "max_divisor.h"
3
4  /*Pre: Recibe una ruta a un archivo en formato de string.
5  Pos: Muestra por stdin dicho archivo (similar al comando unix "cat".*/
6  int mostrar_en_pantalla(char* ruta);
7
8  /*Pre: Recibe una ruta a un archivo en formato de string.
9  Pos: Notifica por stderr que se tuvo un error con un archivo de dicha
   ↪ ruta.*/
10 void notificar_problema_ruta(char *ruta);
11
12
13 int main(int argc, char** argv){
14
15     FILE* stream_entrada = NULL;
16     FILE* stream_salida = NULL;
17     int flag_divisor = -1;
18     int long_index = 0;
19     int opt = 0;
20     char estandar[] = "-";
21
22     static struct option long_options[] = {
23         {COMANDO_VERSION_LARGO,      no_argument,      NULL,
24          ↪ COMANDO_VERSION_CORTO },
25         {COMANDO_AYUDA_LARGO,        no_argument,      NULL,
26          ↪ COMANDO_AYUDA_CORTO },
27         {COMANDO_INPUT_LARGO,        required_argument, NULL,
28          ↪ COMANDO_INPUT_CORTO },
29         {COMANDO_OUTPUT_LARGO,       required_argument, NULL,
30          ↪ COMANDO_OUTPUT_CORTO },
31         {NULL,                       0,                NULL,      0 }
32     };
33
34     while ( (opt = getopt_long(argc, argv,"Vhi:o:",
35                               long_options, &long_index )) != -1) {
36     switch (opt) {
37         case COMANDO_VERSION_CORTO :
38             return mostrar_en_pantalla(RUTA_VERSION);
39         case COMANDO_AYUDA_CORTO :
40             return mostrar_en_pantalla(RUTA_AYUDA);
41     }
```

```

38     case COMANDO_INPUT_CORTO :
39         if (strcmp(optarg, estandar)) {
40             stream_entrada = fopen(optarg, "r");
41             if(stream_entrada == NULL){
42                 notificar_problema_ruta(optarg);
43                 if(stream_salida != NULL ){ //
44                     ↪ se pudo abrir el archivo de
45                     ↪ salida.
46                     fclose(stream_salida);
47                 }
48                 return ERROR;
49             }
50         }
51         break;
52     case COMANDO_OUTPUT_CORTO :
53         if (strcmp(optarg, estandar)) {
54             stream_salida = fopen(optarg, "w");
55             if(stream_salida == NULL){
56                 notificar_problema_ruta(optarg);
57                 if(stream_entrada != NULL ){ //
58                     ↪ se pudo abrir el archivo de
59                     ↪ entrada.
60                     fclose(stream_entrada);
61                 }
62                 return ERROR;
63             }
64         }
65         break;
66     default:
67         perror(MENSAJE_COMANDO_INVALIDO_ERROR);
68         return ERROR;
69 }
70
71 // Caso en que se usa la entrada/salida estandar.
72 if(stream_entrada == NULL ) stream_entrada = stdin;
73 if(stream_salida == NULL ) stream_salida = stdout;
74
75 flag_divisor = procesar_archivos(stream_entrada, stream_salida);
76
77 if(stream_salida != stdout) fclose(stream_salida);
78 if(stream_entrada != stdin ) fclose(stream_entrada);
79
80 if(flag_divisor == ERROR){
81     perror(MENSAJE_GCD_ERROR);
82     return ERROR;
83 }

```

```

81         return EXITO;
82     }
83
84
85     int mostrar_en_pantalla(char * ruta){
86         FILE* archivo = fopen(ruta,"r");
87         if (archivo == NULL){
88             perror("\n No se pudo abrir el archivo. \n");
89             return ERROR;
90         }
91
92         int character  = 1;
93
94         while(character != EOF){
95             character = fgetc(archivo);
96             if(character != EOF) putc(character, stdout);
97         }
98
99         fclose(archivo);
100        return EXITO;
101    }
102
103    void notificar_problema_ruta(char *ruta){
104        char mensaje [MAX_MENSAJE];
105        strcpy(mensaje, "\nEl archivo en la ruta: ");
106        strcat(mensaje, ruta);
107        strcat(mensaje, ". No se pudo abrir correctamente\n");
108        perror(mensaje);
109    }

```

6.3. max_divisor.h

Contiene la definición de funciones utilizadas en **max_divisor.c** para manejo de archivos.

```

1  #ifndef _ORDENADOR_H_
2  #define _ORDENADOR_H_
3
4  #include "utils.h"
5  #include "gcd.h"
6
7  /*
8  Pre: Recibe un stream correctamente abierto en modo de lectura
9       y otro en modo escritura.
10 Pos: En el caso que el stream de entrada tenga formato de lineas consecutivas
11       de pares de numeros enteros separados por espacios. Calcula el maximo
12       comun divisor
13       y dicho resultado lo imprime en el stream de salida. Devolviendo el
14       flag "EXITO".

```



```

13         En caso que no se respete el formato se devuelve "FALLO".
14     */
15     int procesar_archivos(FILE* entrada, FILE* salida);
16
17     /*
18     Pre: Recibe un file stream correctamente abierto en modo lectura.
19     Pos: Lee una linea de dicho stream, devolviendo por parametros la misma
20           en forma de array de caracteres y el largo de la linea. En forma de
21     ↪ retorno
22           devuelve un valor indicando el resultado de la lectura:
23           FIN_DE_ARCHIVO: En caso de encontrarse con un EOF.
24           ERROR_LINEA_INVALIDA: En caso que se haya tenido que detener la
25     ↪ ejecucion debido a un
26                                   caracter invalido encontrado
27     ↪ en el stream.
28           ERROR_DE_MEMORIA: En caso que se haya tenido que detener la ejecucion
29     ↪ debido a un
30                                   problema a la hora de reservar o
31     ↪ asignar memoria dinamica.
32           0: En caso de que el archivo continúe.
33     */
34     int leer_linea (FILE* stream, int *largo_linea, char** linea);
35
36     /*
37     Pre: Recibe un array de caracteres que contiene pares numeros enteros
38           separados por un espacio(Esto es previamente validado) junto con su largo.
39     Pos: Devuelve por parametro el array de enteros y su largo.
40     */
41     int pasar_a_enteros(char* linea, int largo_linea, int *enteros);
42
43     /*
44     Pre : Recibe un array de enteros, su largo y un stream de salida.
45     Pos: "Imprime" dicho array en el stream.
46     */
47     int imprimir_salida(struct gcd *gcd, size_t n, FILE* salida);
48
49     /*
50     Pre: Recibe un caracter.
51     Pos: Devuelve TRUE si el caracter indica el fin de linea.
52     */
53     bool es_fin_de_linea(char caracter);
54
55     /*
56     Pre: Recibe un caracter.
57     Pos: Devuelve TRUE si el caracter es considerado como numerico
58           (se considera los signos de + y - como numericos).
59     */

```

```

55 bool es_numerico(char caracter);
56
57 /*
58  Pre: Recibe un caracter
59  Pos: Si el caracter es numerico, EOF, EOL o espacio devuelve TRUE.
60      en otro caso devuelve FALSE. (dicho caracter no se deberia
61      ↪ encontrar en el input).
62  */
63 bool es_caracter_invalido(char caracter);
64 #endif

```

6.4. max_divisor.c

Se encarga de procesar los archivos: reservando memoria para guardar los datos que se van leyendo linea por linea. Luego de leer una linea se pasa de un string a un array de dos enteros. Finalmente se imprime la salida en el archivo de salida.

```

1  #include "max_divisor.h"
2
3  #define FIN_DE_ARCHIVO 1
4  #define ERROR_LINEA_INVALIDA -1
5  #define ERROR_DE_MEMORIA -2
6  #define ERROR_DE_INPUT -3
7
8  #define SEPARADOR ' '
9
10 int procesar_archivos(FILE* entrada, FILE* salida) {
11
12     int lectura = 0;
13     int enteros[2];
14     int largo_linea = 0;
15     char* linea = NULL;
16
17     int largo_buffer = 20;
18     size_t largo_arreglo = 0;
19     struct gcd* arreglo_structs = malloc(sizeof(struct gcd) * largo_buffer);
20     if((arreglo_structs) == NULL){
21         perror(MENSAJE_MEM_DINAMICA_ERROR);
22         free(arreglo_structs);
23         return ERROR_DE_MEMORIA;
24     }
25
26     while(lectura == 0) {
27         if((largo_arreglo) == (largo_buffer-1)) { // Tengo que agrandar mi
28             ↪ memoria.

```

```

29     largo_buffer += 10; //Voy agregando de a 10 lugares.
30     arreglo_structs = realloc(arreglo_structs, sizeof(struct
    ↪ gcd) * largo_buffer); // Re ubico en la memoria.
31
32     if((arreglo_structs) == NULL){
33         perror(MENSAJE_MEM_DINAMICA_ERROR);
34         free(arreglo_structs);
35         return ERROR_DE_MEMORIA;
36     }
37 }
38
39     lectura = leer_linea(entrada, &largo_linea, &linea);
40
41     if(lectura == ERROR_DE_INPUT) {
42         perror(MENSAJE_INPUT_ERROR);
43         free(arreglo_structs);
44         free(linea);
45         return ERROR;
46     }
47
48     if(lectura != ERROR_LINEA_INVALIDA) {
49
50         if(pasar_a_enteros(linea, largo_linea, enteros)) {
51             free(arreglo_structs);
52             free(linea);
53             return ERROR;
54         }
55
56         arreglo_structs[largo_arreglo].num_a = enteros[0];
57         arreglo_structs[largo_arreglo].num_b = enteros[1];
58         arreglo_structs[largo_arreglo].gcd_ab = 0;
59         largo_arreglo += 1;
60     }
61     free(linea);
62 }
63
64     if(lectura == ERROR_LINEA_INVALIDA){
65         free(arreglo_structs);
66         return ERROR;
67     }
68
69     euclides(arreglo_structs, largo_arreglo);
70     if(imprimir_salida(arreglo_structs, largo_arreglo, salida)){
71         perror(MENSAJE_IMPRIMIR_SALIDA_ERROR);
72         return ERROR;
73     }
74     free(arreglo_structs);

```

```

75     return EXITO;
76 }
77
78 int leer_linea(FILE* stream, int *largo_linea, char** linea){
79     int largo_buffer = 20;
80     *linea = (char*) malloc(sizeof(char) * largo_buffer); // Asigno un lugar
81     ↪ en memoria para el linea.
82
83     if( (*linea) == NULL){
84         return ERROR_DE_MEMORIA;
85     }
86
87     (*largo_linea) = 0;
88     int caracter = 1; // Un valor trivial
89     bool primer_numero_leido = false;
90     bool separador_leido = false;
91     bool segundo_numero_leido = false;
92
93     while ( (caracter != EOL ) && (caracter != EOF) ) {
94
95         if( (*largo_linea) == (largo_buffer-1) ){ // Tengo que agrandar mi
96             ↪ memoria.
97             largo_buffer +=10; // Voy agregando de a 10 lugares.
98             (*linea) = (char*) realloc((*linea), sizeof(char) *
99             ↪ largo_buffer); // Re ubico en la memoria.
100             if( (*linea) == NULL){
101                 return ERROR_DE_MEMORIA;
102             }
103         }
104
105         caracter = getc(stream); // Leo un caracter del stream.
106         if( ferror(stream) ){
107             perror(MENSAJE_LINEA_INVALIDA_ERROR);
108             return ERROR_LINEA_INVALIDA;
109         }
110
111         primer_numero_leido |= (es_numerico(caracter));
112         separador_leido |= (primer_numero_leido && caracter ==
113         ↪ SEPARADOR);
114         segundo_numero_leido |= (separador_leido &&
115         ↪ es_numerico(caracter));
116
117         if( es_caracter_invalido(caracter) ){
118             return ERROR_LINEA_INVALIDA; // Si lee un caracter que no
119             ↪ corresponde, devuelve linea invalida.
120         }
121     }

```

```

115         (*linea) [ (*largo_linea) ] = (char) caracter; //Lo guardo en el
        ↪ linea.
116         (*largo_linea)+=1; // Incremento mi tope.
117     }
118
119     if (!(primer_numero_leido && separador_leido && segundo_numero_leido) &&
        ↪ (*largo_linea) > 1) {
120         return ERROR_LINEA_INVALIDA;
121     }
122
123     if(caracter == EOF || (*largo_linea) <=1){ // Siempre va a leer por lo
        ↪ menos un caracter, sea eof o fin de linea
124         return FIN_DE_ARCHIVO;
125     }
126
127     return 0;
128
129 }
130
131 int pasar_a_enteros(char* linea, int largo_linea, int *enteros) {
132     char temporal [largo_linea];
133     char caracter = 'A';
134     int largo_enteros = 0;
135
136     int i = 0;
137     int j = 0;
138
139     while(i < largo_linea) {
140         caracter = linea[i]; i++;
141         if(largo_enteros >= 2) {
142             perror(MENSAJE_INPUT_ERROR);
143             return ERROR_DE_INPUT;
144         } else if(es_numerico(caracter)) {
145             temporal[j] = caracter; j++;
146         } else if((caracter ==SEPARADOR || es_fin_de_linea(caracter)) &&
            ↪ j!=0) {
147             temporal[j] = '\0';
148             enteros[largo_enteros] = atoi(temporal);
149             largo_enteros += 1;
150             j = 0;
151         }
152     }
153     return 0;
154 }
155
156 int imprimir_salida(struct gcd *gcd, size_t largo, FILE* salida) {
157     if(largo == 0) return 0;

```

```

158         for (int i = 0; i < (largo-1); i++) {
159             fprintf(salida, "GCD(%i, %i) = %i \n", gcd[i].num_a,
160                 ↪ gcd[i].num_b, gcd[i].gcd_ab);
161             if(ferror(salida) ){
162                 return 1;
163             }
164         }
165         return 0;
166     }
167
168     bool es_fin_de_linea(char caracter) {
169         return(caracter == EOL || caracter == (char) EOF);
170     }
171
172     bool es_numerico(char caracter) {
173         return((caracter >= '0' && caracter <= '9') ||
174             ↪ caracter=='-' || caracter=='+');
175     }
176
177     bool es_caracter_invalido(char caracter) {
178         return !(es_numerico(caracter) || es_fin_de_linea(caracter) || caracter
179             ↪ ==SEPARADOR);
180     }

```

6.5. gcd.S

El **gcd.S** es el código MIPS encargado de buscar el GCD entre dos numeros que recibe por parámetro.

```

1  #include <sys/regdef.h>
2
3  #define TAM_SF_EUC 40
4  #define OFF_RA_EUC 36
5  #define OFF_GP_EUC 32
6  #define OFF_FP_EUC 28
7  #define OFF_S0_EUC 24
8  #define OFF_N 44
9  #define OFF_GCD 40
10 #define OFF_K 16
11
12 #define TAM_SF_EUC_ 24
13 #define OFF_GP_EUC_ 20
14 #define OFF_FP_EUC_ 16
15 #define OFF_B 28
16 #define OFF_A 24
17 #define OFF_R0 8

```

```

18 #define OFF_R1      4
19 #define OFF_R2      0
20
21 .abicalls
22 .text
23 .align 2
24 .globl euclides
25 .ent euclides
26
27 euclides:
28     .frame fp, TAM_SF_EUC, ra      # Se crea el Stack Frame
29     subu    sp, sp, TAM_SF_EUC
30
31     .cprestore OFF_GP_EUC          # Guarda gp en sp+OFF_GP_EUC
32     sw      ra, OFF_RA_EUC(sp)     # Se guardan los registros
33     sw      fp, OFF_FP_EUC(sp)     # segun ABI
34     sw      s0, OFF_S0_EUC(sp)
35     move    fp, sp
36
37     sw      a0, OFF_GCD(fp)         # struct gcd* gcd
38     sw      a1, OFF_N(fp)          # size_t n
39     sw      zero, OFF_K(fp)        # size_t k = 0
40
41     b       condicion_for
42
43 # Obtiene gcd[k].num_a y le aplica ABS
44 abs_num_a:
45     lw      t1, OFF_K(fp)          # t1 = k
46     sll     t0, t1, 3              # t0 = t1 << 3 = k * 8
47     sll     t1, t1, 2              # t1 = t1 << 2 = k * 4
48     addu    t1, t1, t0             # t1 = t1 + t0 = 4*k + 8*k = 12*k
49
50     lw      t0, OFF_GCD(fp)        # t0 = gcd
51     addu    s0, t0, t1              # s0 = t0 + t1 = gcd + 12*k = &gcd[k]
52
53     lw      t0, 0(s0)              # t0 = mem(s0) = *(&gcd[k]) = gcd[k].num_a
54
55     # ABS(gcd[k].num_a)
56     bgez    t0, abs_num_b          # if gcd[k].num_a >= 0 GOTO abs_num_b
57
58     subu    t0, zero, t0           # t0 = -t0 = -gcd[k].num_a
59
60 # Obtiene gcd[k].num_b y le aplica ABS
61 abs_num_b:
62     move    a0, t0                 # Prepara el primer argumento de euclides_
63
64     lw      t0, 4(s0)              # t0 = mem(s0+4) = *(&gcd[k] + 4) = gcd[k].num_b

```

```

65
66     # ABS(gcd[k].num_b)
67     bgez    t0, llamada_euclides_    # if gcd[k].num_b >= 0 GOTO llamada_euclides_
68
69     subu    t0, zero, t0              # t0 = -t0 = -gcd[k].num_b
70
71     llamada_euclides_: # Llamada a euclides_ y ++k
72     move    a1, t0                    # Prepara el segundo argumento de euclides_
73
74     jal     euclides_
75
76     # Guarda el resultado en memoria
77     sw      v0, 8(s0)                 # gcd[k].gcd_ab = v0
78
79     # Incrementa k
80     lw      t0, OFF_K(fp)              # t0 = k
81     addiu   t0, t0, 1                  # k++
82     sw      t0, OFF_K(fp)              # Guarda k
83
84     condicion_for: # (k < n)
85     lw      t1, OFF_K(fp)              # t1 = k
86     lw      t0, OFF_N(fp)              # t0 = n
87     sltu    t0, t1, t0                 # t0 = (k < n) ? 1 : 0
88     bne     t0, zero, abs_num_a        # if (t0 != 0) GOTO abs_num_a
89                                           # If (k < n) GOTO abs_num_a
90
91     # Termina euclides
92     move    sp, fp                    # Recupera los registros que se guardaron
93     lw      ra, OFF_RA_EUC(sp)
94     lw      fp, OFF_FP_EUC(sp)
95     lw      s0, OFF_S0_EUC(sp)
96
97     addiu   sp, sp, TAM_SF_EUC
98     jr      ra                        # Vuelve al caller
99
100     .end euclides
101
102
103     .align 2
104     .ent euclides_
105     euclides_:
106     .frame  fp, TAM_SF_EUC_, ra        # Se crea el Stack Frame
107     subu    sp, sp, TAM_SF_EUC_
108
109     .cpstore OFF_GP_EUC_               # Se guardan los registros segun ABI
110     sw      fp, OFF_FP_EUC_(sp)
111     move    fp, sp

```



```

112
113
114     sw      a0, OFF_A(fp)          # Guarda los parametros (a, b) en memoria
115     sw      a1, OFF_B(fp)
116
117     # r2 = MAX(a, b)
118     move     t0, a1                # t0 = b
119     move     t1, a0                # t1 = a
120     slt      t2, t1, t0            # t2 = (t1 < t0) ? 1 : 0
121                                     # t2 = (a < b) ? 1 : 0
122
123     movz     t0, t1, t2            # if (t2 = 0) => t0 = t1
124                                     # If (a >= b) => t0 = a
125     sw      t0, OFF_R2(fp)         # mem(fp+OFF_R2) = t0 (Guarda r2)
126
127
128     # r1 = MIN(a, b)
129     move     t0, a1                # t0 = b
130     move     t1, a0                # t1 = a
131     slt      t2, t0, t1            # t2 = (t0 < t1) ? 1 : 0
132                                     # t2 = (b < a) ? 1 : 0
133
134     movz     t0, t1, t2            # if (t2 = 0) => t0 = t1
135                                     # si b >= a => t0 = a
136     sw      t0, OFF_R1(fp)         # mem(fp+OFF_R1) = t0 (Guarda r1)
137
138 while:
139     lw      t1, OFF_R2(fp)         # t1 = r2
140     lw      t0, OFF_R1(fp)         # t0 = r1
141
142     # If ((r0 = r2 % r1) == 0)
143     # Para obtener r2 % r1 realizo la division y obtengo el resto del
144     # registro high del accumulator
145     divu     t1, t0                # r2 / r1
146     mfhi     t2                    # t2 = r0 = r2 % r1
147     sw      t2, OFF_R0(fp)         # mem(fp+OFF_R0) = t2 (Guarda r0)
148     beq      t2, zero, retorno_euclides_ # if (r0 == 0) GOTO retorno_euclides_ (break)
149
150     sw      t0, OFF_R2(fp)         # r2 = r1
151     sw      t2, OFF_R1(fp)         # r1 = r0
152     b        while                # while(1)
153
154 retorno_euclides_:
155     lw      v0, OFF_R1(fp)         # v0 = r1 (return r1)
156
157     move     sp, fp                # Recupera los registros guardados
158     lw      fp, OFF_FP_EUC_(sp)

```

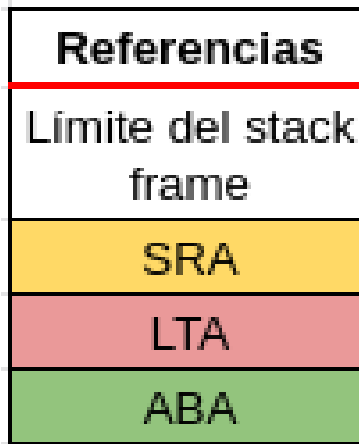
```

159 |      addiu    sp, sp, TAM_SF_EUC_
160 |      jr      ra
161 |
162 |      .end euclides_

```

7. Diagramas de Stack Frame

A continuacion se muestran como quedan los stack frames, bajo determinadas circunstancias, de las funciones codificadas en Assembly en gcd.S



A la hora de llamar a la función euclides el stack frame se encuentra de la siguiente forma:

	...
	padding
SP + 44	n (size_t)
SP + 40	gcd (struct gcd*)
SP + 36	ra
SP + 32	gp
SP + 28	fp
SP + 24	s0
SP + 20	padding
SP + 16	k (size_t)
SP + 12	
SP + 08	
SP + 04	
SP, FP	

En el siguiente diagrama se encuentra el stack frame de la función de euclides_:

	...
	padding
SP + 28	b (int)
SP + 24	a (int)
SP + 20	gp
SP + 16	fp
SP + 12	padding
SP + 08	r0 (int)
SP + 04	r1 (int)
SP, FP	r2 (int)

8. Código MIPS por el compilador

Para finalizar, vamos a mostrar el código MIPS generado por el compilador GCC para los archivos main.c y max_divisor.c

8.1. main.s

```

1      .file      1 "main.c"
2      .section   .mdebug.abi32
3      .previous
4      .nan       legacy
5      .module    fp=xx
6      .module    nooddspreg
7      .abicalls
8      .rdata
9      .align     2
10     $LC0:
11         .ascii   "comandos/version.txt\000"
12         .align   2
13     $LC1:
14         .ascii   "comandos/help.txt\000"
15         .align   2
16     $LC2:
17         .ascii   "r\000"

```

```

18     .align      2
19 $LC3:
20     .ascii      "w\000"
21     .align      2
22 $LC4:
23     .ascii      "\012 El comando usado es invalido, use -h para ayuda.\012"
24     .ascii      "\000"
25     .align      2
26 $LC5:
27     .ascii      "Vhi:o:\000"
28     .align      2
29 $LC6:
30     .ascii      "\012 No se pudo obtener el maximo comun divisor, ocurrio"
31     .ascii      " un error.\012\000"
32     .text
33     .align      2
34     .globl      main
35     .set        nomips16
36     .set        nomicromips
37     .ent        main
38     .type       main, @function
39 main:
40     .frame      $fp,64,$31          # vars= 24, regs= 2/0, args= 24, gp= 8
41     .mask       0xc0000000,-4
42     .fmask      0x00000000,0
43     .set        noreorder
44     .cpload     $25
45     .set        nomacro
46     addiu       $sp,$sp,-64
47     sw          $31,60($sp)
48     sw          $fp,56($sp)
49     move        $fp,$sp
50     .cprestore  24
51     sw          $4,64($fp)
52     sw          $5,68($fp)
53     sw          $0,32($fp)
54     sw          $0,36($fp)
55     li          $2,-1               # 0xffffffffffffffff
56     sw          $2,40($fp)
57     sw          $0,48($fp)
58     sw          $0,44($fp)
59     li          $2,11520            # 0x2d00
60     sh          $2,52($fp)
61     b           $L2
62     nop
63
64 $L14:

```

```

65     lw     $2,44($fp)
66     li     $3,104          # 0x68
67     beq    $2,$3,$L4
68     nop
69
70     slt    $3,$2,105
71     beq    $3,$0,$L5
72     nop
73
74     li     $3,86           # 0x56
75     beq    $2,$3,$L6
76     nop
77
78     b      $L3
79     nop
80
81 $L5:
82     li     $3,105          # 0x69
83     beq    $2,$3,$L7
84     nop
85
86     li     $3,111          # 0x6f
87     beq    $2,$3,$L8
88     nop
89
90     b      $L3
91     nop
92
93 $L6:
94     lw     $2,%got($LC0)($28)
95     addiu   $4,$2,%lo($LC0)
96     lw     $2,%got(mostrar_en_pantalla)($28)
97     move    $25,$2
98     .reloc  1f,R_MIPS_JALR,mostrar_en_pantalla
99 1:      jalr    $25
100     nop
101
102     lw     $28,24($fp)
103     b      $L20
104     nop
105
106 $L4:
107     lw     $2,%got($LC1)($28)
108     addiu   $4,$2,%lo($LC1)
109     lw     $2,%got(mostrar_en_pantalla)($28)
110     move    $25,$2

```

```

111 | .reloc      1f,R_MIPS_JALR,mostrar_en_pantalla
112 | 1: jalr     $25
113 | nop
114 |
115 | lw         $28,24($fp)
116 | b         $L20
117 | nop
118 |
119 | $L7:
120 | lw         $2,%got(optarg)($28)
121 | lw         $2,0($2)
122 | addiu      $3,$fp,52
123 | move       $5,$3
124 | move       $4,$2
125 | lw         $2,%call16(strcmp)($28)
126 | move       $25,$2
127 | .reloc      1f,R_MIPS_JALR,strcmp
128 | 1: jalr     $25
129 | nop
130 |
131 | lw         $28,24($fp)
132 | beq        $2,$0,$L2
133 | nop
134 |
135 | lw         $2,%got(optarg)($28)
136 | lw         $3,0($2)
137 | lw         $2,%got($LC2)($28)
138 | addiu      $5,$2,%lo($LC2)
139 | move       $4,$3
140 | lw         $2,%call16(fopen)($28)
141 | move       $25,$2
142 | .reloc      1f,R_MIPS_JALR,fopen
143 | 1: jalr     $25
144 | nop
145 |
146 | lw         $28,24($fp)
147 | sw         $2,32($fp)
148 | lw         $2,32($fp)
149 | bne        $2,$0,$L2
150 | nop
151 |
152 | lw         $2,%got(optarg)($28)
153 | lw         $2,0($2)
154 | move       $4,$2
155 | lw         $2,%got(notificar_problema_ruta)($28)
156 | move       $25,$2

```

```

157     .reloc      1f,R_MIPS_JALR,notificar_problema_ruta
158 1:    jalr      $25
159     nop
160
161     lw          $28,24($fp)
162     lw          $2,36($fp)
163     beq         $2,$0,$L11
164     nop
165
166     lw          $4,36($fp)
167     lw          $2,%call16(fclose)($28)
168     move        $25,$2
169     .reloc      1f,R_MIPS_JALR,fclose
170 1:    jalr      $25
171     nop
172
173     lw          $28,24($fp)
174 $L11:
175     li          $2,1           # 0x1
176     b           $L20
177     nop
178
179 $L8:
180     lw          $2,%got(optarg)($28)
181     lw          $2,0($2)
182     addiu       $3,$fp,52
183     move        $5,$3
184     move        $4,$2
185     lw          $2,%call16(strcmp)($28)
186     move        $25,$2
187     .reloc      1f,R_MIPS_JALR,strcmp
188 1:    jalr      $25
189     nop
190
191     lw          $28,24($fp)
192     beq         $2,$0,$L2
193     nop
194
195     lw          $2,%got(optarg)($28)
196     lw          $3,0($2)
197     lw          $2,%got($LC3)($28)
198     addiu       $5,$2,%lo($LC3)
199     move        $4,$3
200     lw          $2,%call16(fopen)($28)
201     move        $25,$2
202     .reloc      1f,R_MIPS_JALR,fopen

```



```

203 1: jalr    $25
204    nop
205
206    lw      $28,24($fp)
207    sw      $2,36($fp)
208    lw      $2,36($fp)
209    bne     $2,$0,$L2
210    nop
211
212    lw      $2,%got(optarg)($28)
213    lw      $2,0($2)
214    move     $4,$2
215    lw      $2,%got(notificar_problema_ruta)($28)
216    move     $25,$2
217    .reloc   1f,R_MIPS_JALR,notificar_problema_ruta
218 1: jalr    $25
219    nop
220
221    lw      $28,24($fp)
222    lw      $2,32($fp)
223    beq     $2,$0,$L13
224    nop
225
226    lw      $4,32($fp)
227    lw      $2,%call16(fclose)($28)
228    move     $25,$2
229    .reloc   1f,R_MIPS_JALR,fclose
230 1: jalr    $25
231    nop
232
233    lw      $28,24($fp)
234 $L13:
235    li      $2,1          # 0x1
236    b       $L20
237    nop
238
239 $L3:
240    lw      $2,%got($LC4)($28)
241    addiu    $4,$2,%lo($LC4)
242    lw      $2,%call16(perror)($28)
243    move     $25,$2
244    .reloc   1f,R_MIPS_JALR, perror
245 1: jalr    $25
246    nop
247
248    lw      $28,24($fp)

```

```

249     li    $2,1                # 0x1
250     b     $L20
251     nop
252
253 $L2:
254     addiu  $2,$fp,48
255     sw     $2,16($sp)
256     lw     $2,%got(long_options.2593)($28)
257     addiu  $7,$2,%lo(long_options.2593)
258     lw     $2,%got($LC5)($28)
259     addiu  $6,$2,%lo($LC5)
260     lw     $5,68($fp)
261     lw     $4,64($fp)
262     lw     $2,%call16(getopt_long)($28)
263     move   $25,$2
264     .reloc  1f,R_MIPS_JALR,getopt_long
265 1:     jalr  $25
266     nop
267
268     lw     $28,24($fp)
269     sw     $2,44($fp)
270     lw     $3,44($fp)
271     li     $2,-1              # 0xffffffffffffffff
272     bne    $3,$2,$L14
273     nop
274
275     lw     $2,32($fp)
276     bne    $2,$0,$L15
277     nop
278
279     lw     $2,%got(stdin)($28)
280     lw     $2,0($2)
281     sw     $2,32($fp)
282 $L15:
283     lw     $2,36($fp)
284     bne    $2,$0,$L16
285     nop
286
287     lw     $2,%got(stdout)($28)
288     lw     $2,0($2)
289     sw     $2,36($fp)
290 $L16:
291     lw     $5,36($fp)
292     lw     $4,32($fp)
293     lw     $2,%call16(procesar_archivos)($28)
294     move   $25,$2

```

```

295     .reloc      1f,R_MIPS_JALR,procesar_archivos
296 1:    jalr      $25
297     nop
298
299     lw          $28,24($fp)
300     sw          $2,40($fp)
301     lw          $2,%got(stdout)($28)
302     lw          $2,0($2)
303     lw          $3,36($fp)
304     beq         $3,$2,$L17
305     nop
306
307     lw          $4,36($fp)
308     lw          $2,%call16(fclose)($28)
309     move        $25,$2
310     .reloc      1f,R_MIPS_JALR,fclose
311 1:    jalr      $25
312     nop
313
314     lw          $28,24($fp)
315 $L17:
316     lw          $2,%got(stdin)($28)
317     lw          $2,0($2)
318     lw          $3,32($fp)
319     beq         $3,$2,$L18
320     nop
321
322     lw          $4,32($fp)
323     lw          $2,%call16(fclose)($28)
324     move        $25,$2
325     .reloc      1f,R_MIPS_JALR,fclose
326 1:    jalr      $25
327     nop
328
329     lw          $28,24($fp)
330 $L18:
331     lw          $3,40($fp)
332     li          $2,1           # 0x1
333     bne         $3,$2,$L19
334     nop
335
336     lw          $2,%got($LC6)($28)
337     addiu       $4,$2,%lo($LC6)
338     lw          $2,%call16(perror)($28)
339     move        $25,$2
340     .reloc      1f,R_MIPS_JALR, perror

```

```

341 1:    jalr    $25
342    nop
343
344    lw      $28,24($fp)
345    li      $2,1           # 0x1
346    b       $L20
347    nop
348
349 $L19:
350    move     $2,$0
351 $L20:
352    move     $sp,$fp
353    lw      $31,60($sp)
354    lw      $fp,56($sp)
355    addiu    $sp,$sp,64
356    jr      $31
357    nop
358
359    .set      macro
360    .set      reorder
361    .end      main
362    .size     main, .-main
363    .rdata
364    .align    2
365 $LC7:
366    .ascii    "\012 No se pudo abrir el archivo. \012\000"
367    .text
368    .align    2
369    .globl    mostrar_en_pantalla
370    .set      nomips16
371    .set      nomicromips
372    .ent      mostrar_en_pantalla
373    .type     mostrar_en_pantalla, @function
374 mostrar_en_pantalla:
375    .frame     $fp,40,$31           # vars= 8, regs= 2/0, args= 16, gp= 8
376    .mask     0xc0000000,-4
377    .fmask     0x00000000,0
378    .set      noreorder
379    .cpload    $25
380    .set      nomacro
381    addiu     $sp,$sp,-40
382    sw       $31,36($sp)
383    sw       $fp,32($sp)
384    move     $fp,$sp
385    .cprestore 16
386    sw       $4,40($fp)
387    lw       $2,%got($LC2)($28)

```

```

388      addiu    $5,$2,%lo($LC2)
389      lw      $4,40($fp)
390      lw      $2,%call16(fopen)($28)
391      move     $25,$2
392      .reloc    1f,R_MIPS_JALR,fopen
393  1:      jalr   $25
394      nop
395
396      lw      $28,16($fp)
397      sw      $2,28($fp)
398      lw      $2,28($fp)
399      bne     $2,$0,$L22
400      nop
401
402      lw      $2,%got($LC7)($28)
403      addiu    $4,$2,%lo($LC7)
404      lw      $2,%call16(perror)($28)
405      move     $25,$2
406      .reloc    1f,R_MIPS_JALR,pererror
407  1:      jalr   $25
408      nop
409
410      lw      $28,16($fp)
411      li      $2,1                # 0x1
412      b       $L23
413      nop
414
415  $L22:
416      li      $2,1                # 0x1
417      sw      $2,24($fp)
418      b       $L24
419      nop
420
421  $L25:
422      lw      $4,28($fp)
423      lw      $2,%call16(fgetc)($28)
424      move     $25,$2
425      .reloc    1f,R_MIPS_JALR,fgetc
426  1:      jalr   $25
427      nop
428
429      lw      $28,16($fp)
430      sw      $2,24($fp)
431      lw      $3,24($fp)
432      li      $2,-1              # 0xffffffffffffffff
433      beq     $3,$2,$L24

```

```

434     nop
435
436     lw     $2,%got(stdout)($28)
437     lw     $2,0($2)
438     move   $5,$2
439     lw     $4,24($fp)
440     lw     $2,%call16(_IO_putc)($28)
441     move   $25,$2
442     .reloc 1f,R_MIPS_JALR,_IO_putc
443 1:     jalr $25
444     nop
445
446     lw     $28,16($fp)
447 $L24:
448     lw     $3,24($fp)
449     li     $2,-1           # 0xffffffffffffffff
450     bne    $3,$2,$L25
451     nop
452
453     lw     $4,28($fp)
454     lw     $2,%call16(fclose)($28)
455     move   $25,$2
456     .reloc 1f,R_MIPS_JALR,fclose
457 1:     jalr $25
458     nop
459
460     lw     $28,16($fp)
461     move   $2,$0
462 $L23:
463     move   $sp,$fp
464     lw     $31,36($sp)
465     lw     $fp,32($sp)
466     addiu  $sp,$sp,40
467     jr     $31
468     nop
469
470     .set   macro
471     .set   reorder
472     .end   mostrar_en_pantalla
473     .size  mostrar_en_pantalla, .-mostrar_en_pantalla
474     .rdata
475     .align 2
476 $LC8:
477     .ascii "\012El archivo en la ruta: \000"
478     .align 2
479 $LC9:

```

```

480     .ascii      ". No se pudo abrir correctamente\000\000"
481     .text
482     .align      2
483     .globl      notificar_problema_ruta
484     .set        nomips16
485     .set        nomicromips
486     .ent        notificar_problema_ruta
487     .type       notificar_problema_ruta, @function
488 notificar_problema_ruta:
489     .frame      $fp,184,$31          # vars= 152, regs= 2/0, args= 16, gp= 8
490     .mask       0xc0000000,-4
491     .fmask      0x00000000,0
492     .set        noreorder
493     .cpload     $25
494     .set        nomacro
495     addiu       $sp,$sp,-184
496     sw          $31,180($sp)
497     sw          $fp,176($sp)
498     move        $fp,$sp
499     .cpstore    16
500     sw          $4,184($fp)
501     lw          $2,%got($LC8)($28)
502     lw          $8,%lo($LC8)($2)
503     addiu       $3,$2,%lo($LC8)
504     lw          $7,4($3)
505     addiu       $3,$2,%lo($LC8)
506     lw          $6,8($3)
507     addiu       $3,$2,%lo($LC8)
508     lw          $5,12($3)
509     addiu       $3,$2,%lo($LC8)
510     lw          $4,16($3)
511     addiu       $3,$2,%lo($LC8)
512     lw          $3,20($3)
513     sw          $8,24($fp)
514     sw          $7,28($fp)
515     sw          $6,32($fp)
516     sw          $5,36($fp)
517     sw          $4,40($fp)
518     sw          $3,44($fp)
519     addiu       $2,$2,%lo($LC8)
520     lbu         $2,24($2)
521     sb          $2,48($fp)
522     lw          $5,184($fp)
523     addiu       $2,$fp,24
524     move        $4,$2
525     lw          $2,%call16(strcat)($28)
526     move        $25,$2

```

```

527     .reloc      1f,R_MIPS_JALR, strcat
528 1:    jalr      $25
529     nop
530
531     lw         $28,16($fp)
532     addiu      $2,$fp,24
533     move       $4,$2
534     lw         $2,%call16(strlen)($28)
535     move       $25,$2
536     .reloc      1f,R_MIPS_JALR, strlen
537 1:    jalr      $25
538     nop
539
540     lw         $28,16($fp)
541     move       $3,$2
542     addiu      $2,$fp,24
543     addu       $3,$2,$3
544     lw         $2,%got($LC9)($28)
545     move       $4,$3
546     addiu      $2,$2,%lo($LC9)
547     li         $3,33             # 0x21
548     move       $6,$3
549     move       $5,$2
550     lw         $2,%call16(memcpy)($28)
551     move       $25,$2
552     .reloc      1f,R_MIPS_JALR, memcpy
553 1:    jalr      $25
554     nop
555
556     lw         $28,16($fp)
557     addiu      $2,$fp,24
558     move       $4,$2
559     lw         $2,%call16(perror)($28)
560     move       $25,$2
561     .reloc      1f,R_MIPS_JALR, perror
562 1:    jalr      $25
563     nop
564
565     lw         $28,16($fp)
566     nop
567     move       $sp,$fp
568     lw         $31,180($sp)
569     lw         $fp,176($sp)
570     addiu      $sp,$sp,184
571     jr         $31
572     nop

```



```

573
574     .set      macro
575     .set      reorder
576     .end      notificar_problema_ruta
577     .size     notificar_problema_ruta, .-notificar_problema_ruta
578     .rdata
579     .align    2
580 $LC10:
581     .ascii    "version\000"
582     .align    2
583 $LC11:
584     .ascii    "help\000"
585     .align    2
586 $LC12:
587     .ascii    "input\000"
588     .align    2
589 $LC13:
590     .ascii    "output\000"
591     .section   .data.rel.local,"aw",@progbits
592     .align    2
593     .type     long_options.2593, @object
594     .size     long_options.2593, 80
595 long_options.2593:
596     .word     $LC10
597     .word     0
598     .word     0
599     .word     86
600     .word     $LC11
601     .word     0
602     .word     0
603     .word     104
604     .word     $LC12
605     .word     1
606     .word     0
607     .word     105
608     .word     $LC13
609     .word     1
610     .word     0
611     .word     111
612     .word     0
613     .word     0
614     .word     0
615     .word     0
616     .ident    "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"

```

8.2. max_divisor.s

```
1  .file      1 "max_divisor.c"
2  .section  .mdebug.abi32
3  .previous
4  .nan      legacy
5  .module   fp=xx
6  .module   nooddspreg
7  .abicalls
8  .rdata
9  .align    2
10 $LC0:
11  .ascii    "\012 Problema asignando memoria dinamica.\012\000"
12  .align    2
13 $LC1:
14  .ascii    "\012La cantidad de numeros ingresada es invalida. Solo s"
15  .ascii    "e pueden ingresar 2 numeros por cada calculo de GCD.\012"
16  .ascii    "\000"
17  .align    2
18 $LC2:
19  .ascii    "\012Problema al imprimir la salida.\012\000"
20  .text
21  .align    2
22  .globl    procesar_archivos
23  .set      nomips16
24  .set      nomicromips
25  .ent      procesar_archivos
26  .type     procesar_archivos, @function
27 procesar_archivos:
28  .frame    $fp,64,$31          # vars= 32, regs= 2/0, args= 16, gp= 8
29  .mask     0xc0000000,-4
30  .fmask     0x00000000,0
31  .set      noreorder
32  .cpload   $25
33  .set      nomacro
34  addiu     $sp,$sp,-64
35  sw        $31,60($sp)
36  sw        $fp,56($sp)
37  move      $fp,$sp
38  .cprestore 16
39  sw        $4,64($fp)
40  sw        $5,68($fp)
41  sw        $0,24($fp)
42  sw        $0,48($fp)
43  sw        $0,52($fp)
44  li        $2,20               # 0x14
45  sw        $2,28($fp)
```

```

46     sw     $0,32($fp)
47     lw     $3,28($fp)
48     move   $2,$3
49     sll    $2,$2,1
50     addu   $2,$2,$3
51     sll    $2,$2,2
52     move   $4,$2
53     lw     $2,%call16(malloc)($28)
54     move   $25,$2
55     .reloc 1f,R_MIPS_JALR,malloc
56 1:    jalr  $25
57     nop
58
59     lw     $28,16($fp)
60     sw     $2,36($fp)
61     lw     $2,36($fp)
62     bne    $2,$0,$L4
63     nop
64
65     lw     $2,%got($LC0)($28)
66     addiu   $4,$2,%lo($LC0)
67     lw     $2,%call16(perror)($28)
68     move   $25,$2
69     .reloc 1f,R_MIPS_JALR,perror
70 1:    jalr  $25
71     nop
72
73     lw     $28,16($fp)
74     lw     $4,36($fp)
75     lw     $2,%call16(free)($28)
76     move   $25,$2
77     .reloc 1f,R_MIPS_JALR,free
78 1:    jalr  $25
79     nop
80
81     lw     $28,16($fp)
82     li     $2,-2                # 0xfffffffffffffffe
83     b      $L12
84     nop
85
86 $L9:
87     lw     $2,28($fp)
88     addiu   $2,$2,-1
89     move   $3,$2
90     lw     $2,32($fp)
91     bne    $3,$2,$L5

```

```

92     nop
93
94     lw     $2,28($fp)
95     addiu   $2,$2,10
96     sw     $2,28($fp)
97     lw     $3,28($fp)
98     move    $2,$3
99     sll     $2,$2,1
100    addu    $2,$2,$3
101    sll     $2,$2,2
102    move    $5,$2
103    lw     $4,36($fp)
104    lw     $2,%call16(realloc)($28)
105    move    $25,$2
106    .reloc   1f,R_MIPS_JALR,realloc
107 1:     jalr   $25
108     nop
109
110    lw     $28,16($fp)
111    sw     $2,36($fp)
112    lw     $2,36($fp)
113    bne    $2,$0,$L5
114    nop
115
116    lw     $2,%got($LC0)($28)
117    addiu   $4,$2,%lo($LC0)
118    lw     $2,%call16(perror)($28)
119    move    $25,$2
120    .reloc   1f,R_MIPS_JALR,perror
121 1:     jalr   $25
122     nop
123
124    lw     $28,16($fp)
125    lw     $4,36($fp)
126    lw     $2,%call16(free)($28)
127    move    $25,$2
128    .reloc   1f,R_MIPS_JALR,free
129 1:     jalr   $25
130     nop
131
132    lw     $28,16($fp)
133    li     $2,-2                # 0xfffffffffffffffe
134    b      $L12
135    nop
136
137 $L5:

```

```

138      addiu    $3,$fp,52
139      addiu    $2,$fp,48
140      move     $6,$3
141      move     $5,$2
142      lw       $4,64($fp)
143      lw       $2,%got(leer_linea)($28)
144      move     $25,$2
145      .reloc    1f,R_MIPS_JALR,leer_linea
146  1:      jalr    $25
147      nop
148
149      lw       $28,16($fp)
150      sw       $2,24($fp)
151      lw       $3,24($fp)
152      li       $2,-3           # 0xfffffffffffffffffd
153      bne      $3,$2,$L6
154      nop
155
156      lw       $2,%got($LC1)($28)
157      addiu    $4,$2,%lo($LC1)
158      lw       $2,%call16(perror)($28)
159      move     $25,$2
160      .reloc    1f,R_MIPS_JALR,pererror
161  1:      jalr    $25
162      nop
163
164      lw       $28,16($fp)
165      lw       $4,36($fp)
166      lw       $2,%call16(free)($28)
167      move     $25,$2
168      .reloc    1f,R_MIPS_JALR,free
169  1:      jalr    $25
170      nop
171
172      lw       $28,16($fp)
173      lw       $2,52($fp)
174      move     $4,$2
175      lw       $2,%call16(free)($28)
176      move     $25,$2
177      .reloc    1f,R_MIPS_JALR,free
178  1:      jalr    $25
179      nop
180
181      lw       $28,16($fp)
182      li       $2,1           # 0x1
183      b        $L12

```

```

184     nop
185
186 $L6:
187     lw     $3,24($fp)
188     li     $2,-1           # 0xffffffffffffffff
189     beq     $3,$2,$L7
190     nop
191
192     lw     $2,52($fp)
193     lw     $3,48($fp)
194     addiu   $4,$fp,40
195     move    $6,$4
196     move    $5,$3
197     move    $4,$2
198     lw     $2,%got(pasar_a_enteros)($28)
199     move    $25,$2
200     .reloc   1f,R_MIPS_JALR,pasar_a_enteros
201 1:     jalr   $25
202     nop
203
204     lw     $28,16($fp)
205     beq     $2,$0,$L8
206     nop
207
208     lw     $4,36($fp)
209     lw     $2,%call16(free)($28)
210     move    $25,$2
211     .reloc   1f,R_MIPS_JALR,free
212 1:     jalr   $25
213     nop
214
215     lw     $28,16($fp)
216     lw     $2,52($fp)
217     move    $4,$2
218     lw     $2,%call16(free)($28)
219     move    $25,$2
220     .reloc   1f,R_MIPS_JALR,free
221 1:     jalr   $25
222     nop
223
224     lw     $28,16($fp)
225     li     $2,1           # 0x1
226     b       $L12
227     nop
228
229 $L8:

```

```

230     lw     $3,32($fp)
231     move   $2,$3
232     sll    $2,$2,1
233     addu   $2,$2,$3
234     sll    $2,$2,2
235     move   $3,$2
236     lw     $2,36($fp)
237     addu   $2,$2,$3
238     lw     $3,40($fp)
239     sw     $3,0($2)
240     lw     $3,32($fp)
241     move   $2,$3
242     sll    $2,$2,1
243     addu   $2,$2,$3
244     sll    $2,$2,2
245     move   $3,$2
246     lw     $2,36($fp)
247     addu   $2,$2,$3
248     lw     $3,44($fp)
249     sw     $3,4($2)
250     lw     $3,32($fp)
251     move   $2,$3
252     sll    $2,$2,1
253     addu   $2,$2,$3
254     sll    $2,$2,2
255     move   $3,$2
256     lw     $2,36($fp)
257     addu   $2,$2,$3
258     sw     $0,8($2)
259     lw     $2,32($fp)
260     addiu  $2,$2,1
261     sw     $2,32($fp)
262 $L7:
263     lw     $2,52($fp)
264     move   $4,$2
265     lw     $2,%call16(free)($28)
266     move   $25,$2
267     .reloc 1f,R_MIPS_JALR,free
268 1:     jalr  $25
269     nop
270
271     lw     $28,16($fp)
272 $L4:
273     lw     $2,24($fp)
274     beq    $2,$0,$L9
275     nop

```

```

276
277     lw     $3,24($fp)
278     li     $2,-1           # 0xffffffffffffffff
279     bne    $3,$2,$L10
280     nop
281
282     lw     $4,36($fp)
283     lw     $2,%call16(free)($28)
284     move    $25,$2
285     .reloc  1f,R_MIPS_JALR,free
286 1:     jalr    $25
287     nop
288
289     lw     $28,16($fp)
290     li     $2,1           # 0x1
291     b      $L12
292     nop
293
294 $L10:
295     lw     $5,32($fp)
296     lw     $4,36($fp)
297     lw     $2,%call16(euclides)($28)
298     move    $25,$2
299     .reloc  1f,R_MIPS_JALR,euclides
300 1:     jalr    $25
301     nop
302
303     lw     $28,16($fp)
304     lw     $6,68($fp)
305     lw     $5,32($fp)
306     lw     $4,36($fp)
307     lw     $2,%got(imprimir_salida)($28)
308     move    $25,$2
309     .reloc  1f,R_MIPS_JALR,imprimir_salida
310 1:     jalr    $25
311     nop
312
313     lw     $28,16($fp)
314     beq     $2,$0,$L11
315     nop
316
317     lw     $2,%got($LC2)($28)
318     addiu   $4,$2,%lo($LC2)
319     lw     $2,%call16(perror)($28)
320     move    $25,$2
321     .reloc  1f,R_MIPS_JALR,perror

```



```

322 1:    jalr    $25
323    nop
324
325    lw      $28,16($fp)
326    li      $2,1          # 0x1
327    b       $L12
328    nop
329
330 $L11:
331    lw      $4,36($fp)
332    lw      $2,%call16(free)($28)
333    move     $25,$2
334    .reloc   1f,R_MIPS_JALR,free
335 1:    jalr    $25
336    nop
337
338    lw      $28,16($fp)
339    move     $2,$0
340 $L12:
341    move     $sp,$fp
342    lw      $31,60($sp)
343    lw      $fp,56($sp)
344    addiu    $sp,$sp,64
345    jr      $31
346    nop
347
348    .set     macro
349    .set     reorder
350    .end     procesar_archivos
351    .size    procesar_archivos, .-procesar_archivos
352    .rdata
353    .align   2
354 $LC3:
355    .ascii   "\012Problema leyendo el archivo.\012\000"
356    .text
357    .align   2
358    .globl   leer_linea
359    .set     nomips16
360    .set     nomicromips
361    .ent     leer_linea
362    .type    leer_linea, @function
363 leer_linea:
364    .frame    $fp,48,$31          # vars= 16, regs= 2/0, args= 16, gp= 8
365    .mask     0xc0000000,-4
366    .fmask    0x00000000,0
367    .set     noreorder

```

```

368     .cpload    $25
369     .set      nomacro
370     addiu     $sp,$sp,-48
371     sw       $31,44($sp)
372     sw       $fp,40($sp)
373     move     $fp,$sp
374     .cpstore   16
375     sw       $4,48($fp)
376     sw       $5,52($fp)
377     sw       $6,56($fp)
378     li       $2,20           # 0x14
379     sw       $2,24($fp)
380     lw       $2,24($fp)
381     move     $4,$2
382     lw       $2,%call16(malloc)($28)
383     move     $25,$2
384     .reloc    1f,R_MIPS_JALR,malloc
385 1:     jalr    $25
386     nop
387
388     lw       $28,16($fp)
389     move     $3,$2
390     lw       $2,56($fp)
391     sw       $3,0($2)
392     lw       $2,56($fp)
393     lw       $2,0($2)
394     bne     $2,$0,$L14
395     nop
396
397     li       $2,-2           # 0xfffffffffffffffe
398     b       $L15
399     nop
400
401 $L14:
402     lw       $2,52($fp)
403     sw       $0,0($2)
404     li       $2,1           # 0x1
405     sw       $2,28($fp)
406     sb       $0,32($fp)
407     sb       $0,33($fp)
408     sb       $0,34($fp)
409     b       $L16
410     nop
411
412 $L25:
413     lw       $2,52($fp)

```

```

414     lw     $3,0($2)
415     lw     $2,24($fp)
416     addiu   $2,$2,-1
417     bne     $3,$2,$L17
418     nop
419
420     lw     $2,24($fp)
421     addiu   $2,$2,10
422     sw     $2,24($fp)
423     lw     $2,56($fp)
424     lw     $2,0($2)
425     lw     $3,24($fp)
426     move    $5,$3
427     move    $4,$2
428     lw     $2,%call16(realloc)($28)
429     move    $25,$2
430     .reloc   1f,R_MIPS_JALR,realloc
431 1:     jalr   $25
432     nop
433
434     lw     $28,16($fp)
435     move    $3,$2
436     lw     $2,56($fp)
437     sw     $3,0($2)
438     lw     $2,56($fp)
439     lw     $2,0($2)
440     bne     $2,$0,$L17
441     nop
442
443     li     $2,-2                # 0xfffffffffffffffe
444     b      $L15
445     nop
446
447 $L17:
448     lw     $4,48($fp)
449     lw     $2,%call16(_IO_getc)($28)
450     move    $25,$2
451     .reloc   1f,R_MIPS_JALR,_IO_getc
452 1:     jalr   $25
453     nop
454
455     lw     $28,16($fp)
456     sw     $2,28($fp)
457     lw     $4,48($fp)
458     lw     $2,%call16(ferror)($28)
459     move    $25,$2

```

```

460     .reloc      1f,R_MIPS_JALR,error
461 1:     jalr      $25
462     nop
463
464     lw          $28,16($fp)
465     beq         $2,$0,$L18
466     nop
467
468     lw          $2,%got($LC3)($28)
469     addiu       $4,$2,%lo($LC3)
470     lw          $2,%call16(perror)($28)
471     move        $25,$2
472     .reloc      1f,R_MIPS_JALR,pererror
473 1:     jalr      $25
474     nop
475
476     lw          $28,16($fp)
477     li          $2,-1           # 0xffffffffffffffff
478     b           $L15
479     nop
480
481 $L18:
482     lw          $2,28($fp)
483     seb         $2,$2
484     move        $4,$2
485     lw          $2,%got(es_numerico)($28)
486     move        $25,$2
487     .reloc      1f,R_MIPS_JALR,es_numerico
488 1:     jalr      $25
489     nop
490
491     lw          $28,16($fp)
492     move        $3,$2
493     lbu         $2,32($fp)
494     or          $2,$3,$2
495     andi        $2,$2,0x00ff
496     sltu        $2,$0,$2
497     sb          $2,32($fp)
498     lbu         $3,33($fp)
499     lbu         $2,32($fp)
500     beq         $2,$0,$L19
501     nop
502
503     lw          $4,28($fp)
504     li          $2,32           # 0x20
505     bne         $4,$2,$L19

```

```

506     nop
507
508     li    $2,1           # 0x1
509     b     $L20
510     nop
511
512 $L19:
513     move   $2,$0
514 $L20:
515     or     $2,$3,$2
516     sltu   $2,$0,$2
517     sb     $2,33($fp)
518     lbu    $2,33($fp)
519     beq    $2,$0,$L21
520     nop
521
522     lw     $2,28($fp)
523     seb    $2,$2
524     move   $4,$2
525     lw     $2,%got(es_numerico)($28)
526     move   $25,$2
527     .reloc 1f,R_MIPS_JALR,es_numerico
528 1:     jalr $25
529     nop
530
531     lw     $28,16($fp)
532     beq    $2,$0,$L21
533     nop
534
535     li    $2,1           # 0x1
536     b     $L22
537     nop
538
539 $L21:
540     move   $2,$0
541 $L22:
542     lbu    $3,34($fp)
543     or     $2,$3,$2
544     sltu   $2,$0,$2
545     sb     $2,34($fp)
546     lw     $2,28($fp)
547     seb    $2,$2
548     move   $4,$2
549     lw     $2,%got(es_caracter_invalido)($28)
550     move   $25,$2
551     .reloc 1f,R_MIPS_JALR,es_caracter_invalido

```

```

552 1: jalr    $25
553    nop
554
555    lw      $28,16($fp)
556    beq     $2,$0,$L23
557    nop
558
559    li      $2,-1          # 0xffffffffffffffff
560    b       $L15
561    nop
562
563 $L23:
564    lw      $2,56($fp)
565    lw      $2,0($2)
566    lw      $3,52($fp)
567    lw      $3,0($3)
568    addu    $2,$2,$3
569    lw      $3,28($fp)
570    seb     $3,$3
571    sb      $3,0($2)
572    lw      $2,52($fp)
573    lw      $2,0($2)
574    addiu   $3,$2,1
575    lw      $2,52($fp)
576    sw      $3,0($2)
577 $L16:
578    lw      $3,28($fp)
579    li      $2,10          # 0xa
580    beq     $3,$2,$L24
581    nop
582
583    lw      $3,28($fp)
584    li      $2,-1          # 0xffffffffffffffff
585    bne     $3,$2,$L25
586    nop
587
588 $L24:
589    lbu     $2,32($fp)
590    xori    $2,$2,0x1
591    andi    $2,$2,0x00ff
592    bne     $2,$0,$L26
593    nop
594
595    lbu     $2,33($fp)
596    xori    $2,$2,0x1
597    andi    $2,$2,0x00ff
598    bne     $2,$0,$L26

```

```

599     nop
600
601     lbu     $2,34($fp)
602     xori    $2,$2,0x1
603     andi    $2,$2,0x00ff
604     beq     $2,$0,$L27
605     nop
606
607 $L26:
608     lw      $2,52($fp)
609     lw      $2,0($2)
610     slt     $2,$2,2
611     bne     $2,$0,$L27
612     nop
613
614     li      $2,-1           # 0xffffffffffffffff
615     b       $L15
616     nop
617
618 $L27:
619     lw      $3,28($fp)
620     li      $2,-1           # 0xffffffffffffffff
621     beq     $3,$2,$L28
622     nop
623
624     lw      $2,52($fp)
625     lw      $2,0($2)
626     slt     $2,$2,2
627     beq     $2,$0,$L29
628     nop
629
630 $L28:
631     li      $2,1           # 0x1
632     b       $L15
633     nop
634
635 $L29:
636     move    $2,$0
637 $L15:
638     move    $sp,$fp
639     lw      $31,44($sp)
640     lw      $fp,40($sp)
641     addiu   $sp,$sp,48
642     jr      $31
643     nop
644
645     .set    macro

```

```

646 .set      reorder
647 .end      leer_linea
648 .size    leer_linea, .-leer_linea
649 .align    2
650 .globl    pasar_a_enteros
651 .set      nomips16
652 .set      nomicromips
653 .ent      pasar_a_enteros
654 .type     pasar_a_enteros, @function
655 pasar_a_enteros:
656 .frame     $fp,64,$31          # vars= 24, regs= 4/0, args= 16, gp= 8
657 .mask      0xc0030000,-4
658 .fmask      0x00000000,0
659 .set      noreorder
660 .cpload     $25
661 .set      nomacro
662 addiu      $sp,$sp,-64
663 sw         $31,60($sp)
664 sw         $fp,56($sp)
665 sw         $17,52($sp)
666 sw         $16,48($sp)
667 move       $fp,$sp
668 .cprestore  16
669 sw         $4,64($fp)
670 sw         $5,68($fp)
671 sw         $6,72($fp)
672 move       $4,$sp
673 move       $17,$4
674 lw         $4,68($fp)
675 addiu      $5,$4,-1
676 sw         $5,36($fp)
677 move       $5,$4
678 move       $13,$5
679 move       $12,$0
680 srl        $5,$13,29
681 sll        $8,$12,3
682 or         $8,$5,$8
683 sll        $9,$13,3
684 move       $5,$4
685 move       $11,$5
686 move       $10,$0
687 srl        $5,$11,29
688 sll        $2,$10,3
689 or         $2,$5,$2
690 sll        $3,$11,3
691 move       $2,$4
692 addiu      $2,$2,7

```



```

693     srl     $2,$2,3
694     sll     $2,$2,3
695     subu    $sp,$sp,$2
696     addiu   $2,$sp,16
697     addiu   $2,$2,0
698     sw      $2,40($fp)
699     li      $2,65           # 0x41
700     sb      $2,44($fp)
701     sw      $0,24($fp)
702     sw      $0,28($fp)
703     sw      $0,32($fp)
704     b       $L31
705     nop
706
707 $L36:
708     lw      $2,28($fp)
709     lw      $3,64($fp)
710     addu    $2,$3,$2
711     lbu     $2,0($2)
712     sb      $2,44($fp)
713     lw      $2,28($fp)
714     addiu   $2,$2,1
715     sw      $2,28($fp)
716     lw      $2,24($fp)
717     slt     $2,$2,2
718     bne     $2,$0,$L32
719     nop
720
721     lw      $2,%got($LC1)($28)
722     addiu   $4,$2,%lo($LC1)
723     lw      $2,%call16(perror)($28)
724     move    $25,$2
725     .reloc   1f,R_MIPS_JALR,perror
726 1:     jalr   $25
727     nop
728
729     lw      $28,16($fp)
730     li      $2,-3           # 0xffffffffffffd
731     b       $L33
732     nop
733
734 $L32:
735     lb      $2,44($fp)
736     move    $4,$2
737     lw      $2,%got(es_numerico)($28)
738     move    $25,$2

```

```

739      .reloc      1f,R_MIPS_JALR,es_numerico
740  1:      jalr      $25
741      nop
742
743      lw          $28,16($fp)
744      beq         $2,$0,$L34
745      nop
746
747      lw          $3,40($fp)
748      lw          $2,32($fp)
749      addu        $2,$3,$2
750      lbu         $3,44($fp)
751      sb          $3,0($2)
752      lw          $2,32($fp)
753      addiu       $2,$2,1
754      sw          $2,32($fp)
755      b           $L31
756      nop
757
758  $L34:
759      lb          $3,44($fp)
760      li          $2,32          # 0x20
761      beq         $3,$2,$L35
762      nop
763
764      lb          $2,44($fp)
765      move        $4,$2
766      lw          $2,%got(es_fin_de_linea)($28)
767      move        $25,$2
768      .reloc      1f,R_MIPS_JALR,es_fin_de_linea
769  1:      jalr      $25
770      nop
771
772      lw          $28,16($fp)
773      beq         $2,$0,$L31
774      nop
775
776  $L35:
777      lw          $2,32($fp)
778      beq         $2,$0,$L31
779      nop
780
781      lw          $3,40($fp)
782      lw          $2,32($fp)
783      addu        $2,$3,$2
784      sb          $0,0($2)

```

```

785     lw     $2,24($fp)
786     sll    $2,$2,2
787     lw     $3,72($fp)
788     addu   $16,$3,$2
789     lw     $2,40($fp)
790     move   $4,$2
791     lw     $2,%call16(atoi)($28)
792     move   $25,$2
793     .reloc 1f,R_MIPS_JALR,atoi
794 1:     jalr  $25
795     nop
796
797     lw     $28,16($fp)
798     sw     $2,0($16)
799     lw     $2,24($fp)
800     addiu  $2,$2,1
801     sw     $2,24($fp)
802     sw     $0,32($fp)
803 $L31:
804     lw     $3,28($fp)
805     lw     $2,68($fp)
806     slt    $2,$3,$2
807     bne    $2,$0,$L36
808     nop
809
810     move   $2,$0
811 $L33:
812     move   $sp,$17
813     move   $sp,$fp
814     lw     $31,60($sp)
815     lw     $fp,56($sp)
816     lw     $17,52($sp)
817     lw     $16,48($sp)
818     addiu  $sp,$sp,64
819     jr     $31
820     nop
821
822     .set    macro
823     .set    reorder
824     .end    pasar_a_enteros
825     .size   pasar_a_enteros, .-pasar_a_enteros
826     .rdata
827     .align  2
828 $LC4:
829     .ascii  "GCD(%i, %i) = %i \012\000"
830     .text

```

```

831     .align      2
832     .globl      imprimir_salida
833     .set        nomips16
834     .set        nomicromips
835     .ent        imprimir_salida
836     .type       imprimir_salida, @function
837 imprimir_salida:
838     .frame      $fp,48,$31          # vars= 8, regs= 2/0, args= 24, gp= 8
839     .mask       0xc0000000,-4
840     .fmask      0x00000000,0
841     .set        noreorder
842     .cpload     $25
843     .set        nomacro
844     addiu       $sp,$sp,-48
845     sw          $31,44($sp)
846     sw          $fp,40($sp)
847     move        $fp,$sp
848     .cpstore    24
849     sw          $4,48($fp)
850     sw          $5,52($fp)
851     sw          $6,56($fp)
852     lw          $2,52($fp)
853     bne         $2,$0,$L39
854     nop
855
856     move        $2,$0
857     b           $L40
858     nop
859
860 $L39:
861     sw          $0,32($fp)
862     b           $L41
863     nop
864
865 $L43:
866     lw          $3,32($fp)
867     move        $2,$3
868     sll         $2,$2,1
869     addu        $2,$2,$3
870     sll         $2,$2,2
871     move        $3,$2
872     lw          $2,48($fp)
873     addu        $2,$2,$3
874     lw          $4,0($2)
875     lw          $3,32($fp)
876     move        $2,$3
877     sll         $2,$2,1

```

```

878     addu    $2,$2,$3
879     sll     $2,$2,2
880     move    $3,$2
881     lw      $2,48($fp)
882     addu    $2,$2,$3
883     lw      $5,4($2)
884     lw      $3,32($fp)
885     move    $2,$3
886     sll     $2,$2,1
887     addu    $2,$2,$3
888     sll     $2,$2,2
889     move    $3,$2
890     lw      $2,48($fp)
891     addu    $2,$2,$3
892     lw      $2,8($2)
893     sw      $2,16($sp)
894     move    $7,$5
895     move    $6,$4
896     lw      $2,%got($LC4)($28)
897     addiu   $5,$2,%lo($LC4)
898     lw      $4,56($fp)
899     lw      $2,%call16(fprintf)($28)
900     move    $25,$2
901     .reloc   1f,R_MIPS_JALR,fprintf
902 1:     jalr   $25
903     nop
904
905     lw      $28,24($fp)
906     lw      $4,56($fp)
907     lw      $2,%call16(ferror)($28)
908     move    $25,$2
909     .reloc   1f,R_MIPS_JALR,ferror
910 1:     jalr   $25
911     nop
912
913     lw      $28,24($fp)
914     beq     $2,$0,$L42
915     nop
916
917     li      $2,1             # 0x1
918     b       $L40
919     nop
920
921 $L42:
922     lw      $2,32($fp)
923     addiu   $2,$2,1

```

```

924     sw      $2,32($fp)
925 $L41:
926     lw      $2,52($fp)
927     addiu   $3,$2,-1
928     lw      $2,32($fp)
929     sltu    $2,$2,$3
930     bne     $2,$0,$L43
931     nop
932
933     move     $2,$0
934 $L40:
935     move     $sp,$fp
936     lw      $31,44($sp)
937     lw      $fp,40($sp)
938     addiu   $sp,$sp,48
939     jr      $31
940     nop
941
942     .set     macro
943     .set     reorder
944     .end     imprimir_salida
945     .size    imprimir_salida, .-imprimir_salida
946     .align   2
947     .globl   es_fin_de_linea
948     .set     nomips16
949     .set     nomicromips
950     .ent     es_fin_de_linea
951     .type    es_fin_de_linea, @function
952 es_fin_de_linea:
953     .frame   $fp,8,$31          # vars= 0, regs= 1/0, args= 0, gp= 0
954     .mask    0x40000000,-4
955     .fmask    0x00000000,0
956     .set     noreorder
957     .set     nomacro
958     addiu   $sp,$sp,-8
959     sw      $fp,4($sp)
960     move     $fp,$sp
961     move     $2,$4
962     sb      $2,8($fp)
963     lb      $3,8($fp)
964     li      $2,10               # 0xa
965     beq     $3,$2,$L45
966     nop
967
968     lb      $3,8($fp)
969     li      $2,-1               # 0xffffffffffffffff
970     bne     $3,$2,$L46

```

```

971     nop
972
973 $L45:
974     li     $2,1           # 0x1
975     b      $L47
976     nop
977
978 $L46:
979     move   $2,$0
980 $L47:
981     andi   $2,$2,0x1
982     andi   $2,$2,0x00ff
983     move   $sp,$fp
984     lw     $fp,4($sp)
985     addiu  $sp,$sp,8
986     jr     $31
987     nop
988
989     .set    macro
990     .set    reorder
991     .end    es_fin_de_linea
992     .size   es_fin_de_linea, .-es_fin_de_linea
993     .align  2
994     .globl  es_numerico
995     .set    nomips16
996     .set    nomicromips
997     .ent    es_numerico
998     .type   es_numerico, @function
999 es_numerico:
1000     .frame  $fp,8,$31     # vars= 0, regs= 1/0, args= 0, gp= 0
1001     .mask   0x40000000,-4
1002     .fmask  0x00000000,0
1003     .set    noreorder
1004     .set    nomacro
1005     addiu   $sp,$sp,-8
1006     sw      $fp,4($sp)
1007     move    $fp,$sp
1008     move    $2,$4
1009     sb      $2,8($fp)
1010     lb      $2,8($fp)
1011     slt     $2,$2,48
1012     bne     $2,$0,$L50
1013     nop
1014
1015     lb      $2,8($fp)
1016     slt     $2,$2,58
1017     bne     $2,$0,$L51

```

```

1018     nop
1019
1020 $L50:
1021     lb     $3,8($fp)
1022     li     $2,45          # 0x2d
1023     beq    $3,$2,$L51
1024     nop
1025
1026     lb     $3,8($fp)
1027     li     $2,43          # 0x2b
1028     bne    $3,$2,$L52
1029     nop
1030
1031 $L51:
1032     li     $2,1           # 0x1
1033     b      $L53
1034     nop
1035
1036 $L52:
1037     move   $2,$0
1038 $L53:
1039     andi   $2,$2,0x1
1040     andi   $2,$2,0x00ff
1041     move   $sp,$fp
1042     lw     $fp,4($sp)
1043     addiu  $sp,$sp,8
1044     jr     $31
1045     nop
1046
1047     .set    macro
1048     .set    reorder
1049     .end     es_numerico
1050     .size    es_numerico, .-es_numerico
1051     .align   2
1052     .globl   es_caracter_invalido
1053     .set     nomips16
1054     .set     nomicromips
1055     .ent     es_caracter_invalido
1056     .type    es_caracter_invalido, @function
1057 es_caracter_invalido:
1058     .frame   $fp,32,$31      # vars= 0, regs= 2/0, args= 16, gp= 8
1059     .mask    0xc0000000,-4
1060     .fmask   0x00000000,0
1061     .set     noreorder
1062     .cpload  $25
1063     .set     nomacro
1064     addiu    $sp,$sp,-32

```



```

1065     sw     $31,28($sp)
1066     sw     $fp,24($sp)
1067     move   $fp,$sp
1068     .cprestore    16
1069     move   $2,$4
1070     sb     $2,32($fp)
1071     lb     $2,32($fp)
1072     move   $4,$2
1073     lw     $2,%got(es_numerico)($28)
1074     move   $25,$2
1075     .reloc    1f,R_MIPS_JALR,es_numerico
1076 1:     jalr    $25
1077     nop
1078
1079     lw     $28,16($fp)
1080     xori   $2,$2,0x1
1081     andi   $2,$2,0x00ff
1082     beq    $2,$0,$L56
1083     nop
1084
1085     lb     $2,32($fp)
1086     move   $4,$2
1087     lw     $2,%got(es_fin_de_linea)($28)
1088     move   $25,$2
1089     .reloc    1f,R_MIPS_JALR,es_fin_de_linea
1090 1:     jalr    $25
1091     nop
1092
1093     lw     $28,16($fp)
1094     xori   $2,$2,0x1
1095     andi   $2,$2,0x00ff
1096     beq    $2,$0,$L56
1097     nop
1098
1099     lb     $3,32($fp)
1100     li     $2,32                # 0x20
1101     beq    $3,$2,$L56
1102     nop
1103
1104     li     $2,1                # 0x1
1105     b      $L57
1106     nop
1107
1108 $L56:
1109     move   $2,$0
1110 $L57:

```

```

1111      andi    $2,$2,0x1
1112      andi    $2,$2,0x00ff
1113      move    $sp,$fp
1114      lw      $31,28($sp)
1115      lw      $fp,24($sp)
1116      addiu   $sp,$sp,32
1117      jr      $31
1118      nop
1119
1120      .set     macro
1121      .set     reorder
1122      .end     es_caracter_invalido
1123      .size    es_caracter_invalido, .-es_caracter_invalido
1124      .ident   "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"

```