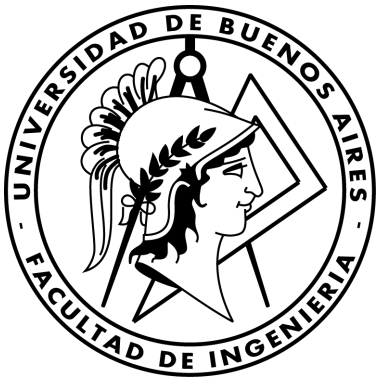

Universidad de Buenos Aires

Facultad de Ingeniería

2do Cuatrimestre de 2020

[86.37 / 66.20] Organización de Computadoras

Curso 2

Trabajo práctico 1

Conjunto de instrucciones MIPS

Padrón	Alumno	Email
103442	Lovera, Daniel	dlovera@fi.uba.ar
102914	More, Agustín	amore@fi.uba.ar
99846	Torresetti, Lisandro	ltorresetti@fi.uba.ar

Repositorio: https://github.com/DanieLovera/Orga

Índice

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de `ABI`, realizando un programa que calcula el mínimo común múltiplo (`mcm`) y el máximo común divisor (`mcd`) entre dos números, utilizando para éste último el *algoritmo de Euclides* [1].

2. Introducción

El algoritmo de Euclides es un método antiguo y eficiente para calcular el máximo común divisor (`mcd`) que se basa en lo siguiente:

Al dividir a entre b (números enteros), se obtiene un cociente q y un residuo r. Es posible demostrar que el máximo común divisor de a y b es el mismo que el de b y r. Sea c el máximo común divisor de a y b, como $a = b * q + r$ y c divide a a y a b divide también a r. Si existiera otro número mayor que c que divide a b y a r, también dividiría a a , por lo que c no sería el `mcd` de a y b (lo que contradice la hipótesis). Este es el fundamento principal del algoritmo.

Para hallar el `mcm` entre dos números se utiliza el `mcd`, dado que el `mcm` entre dos números es el producto de ambos divido entre su `mcd`.

$$mcm(a, b) = \frac{a * b}{mcd(a, b)}$$

3. Detalles de implementación

El programa principal consta de dos rutinas externas importantes escritas en lenguaje *assembly* para procesadores mips32, se encuentran programadas en los módulos `mcd_euclides.S` y `mcm_euclides.S`, las cuales se hicieron siguiendo una implementación realizada para soporte en `C` (`euclides_algorithm.c`) y siguen línea a línea este archivo de código fuente, además tienen asociados cada instrucción de alto nivel en `C` a las instrucciones de bajo nivel en *assembly*. A continuación se detallan cada una de ellas.

3.1. Módulo `mcd_euclides.S`

La subrutina contenida dentro de este módulo (`mcd_euclides(unsigned int a, insigned int b)`), recibe dos parámetros y a su vez utiliza una variable local, por lo cual siguiendo la convención de la `ABI` para mips32 se reservan 40 bytes de memoria en el stack, y en general se distribuye como se muestra a continuación.

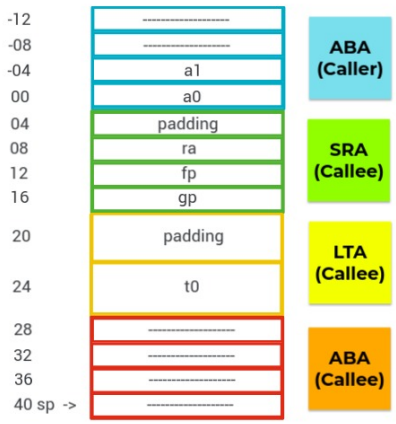

Figura 1: Stack general de subrutina `mcd_euclides(unsigned int a, insigned int b)`

En la Figura 1 se incluye la sección `ABA` (caller), la cual no pertenece a la función en cuestión, sin embargo, para mas claridad se incluyó debido a que por convención es el invocado es el que debe almacenar los parámetros recibidos en los registros dentro del stack del invocador. La primera función que llamará a `mcd_euclides.S` es `main.c` y se programo en alto nivel, por la tanto se confía en que esta función cumple con la convención necesaria.

El `SRA` (callee) corresponde a los registros que el invocado debe respaldar por convención, se incluye el registro `ra` (junto al `fp` y `gp`) en esta región ya que esta es una función no hoja, y se debe evitar que las direcciones de retorno se sobrescriban en cada llamado a una nueva función. Observar que pese a que solo se necesitan 4 bytes (excluyendo los 8 bytes del registro `fp` y `gp`) para el registro `ra`, se reservaron 8 bytes de stack extra para mantener la convención de alineación, por lo cual los 4 bytes restantes quedan rellenos con padding.

El `LTA` (callee) no es necesario crearlo por convención, sin embargo para evitar trabajar únicamente con registros, lo cual haría el código mas complicado de entender y propenso a errores, se decidió crearla para priorizar la claridad del código y facilitar el trabajo de programación. Nuevamente se reservan 4 bytes adicionales por cuestiones de alineación.

Finalmente la complejidad de esta implementación reside en el llamado recursivo que realiza. Esta es la razón por la cual estamos obligados a reservar 16 bytes extras que corresponden a la sección `ABA` (callee), como se mencionó, por convención se debe reservar espacio en memoria para que la función invocada pueda almacenar sus parámetros en esta sección, una vez se realice el nuevo llamado, la función pasara de su condición de invocada a invocadora una y otra vez hasta que finalice su condición de corte y empiece a retornar el resultado por cada uno de los nuevos marcos de pila que se crearon hasta la primera función invocadora (el `main`). Se debe señalar que por seguir la convención de la `ABI` inicialmente, los llamados recursivos resultaron ser algo trivial pues la función ya sabe que debe guardar sus parámetros en la función invocadora y cuánto espacio tiene que reservar, por lo cual no fallará cuando tenga que llamarse a si misma.

A continuación se muestra como se ve el stack de la función invocada una vez finaliza sus tareas y esta lista para retornar o invocarse nuevamente.

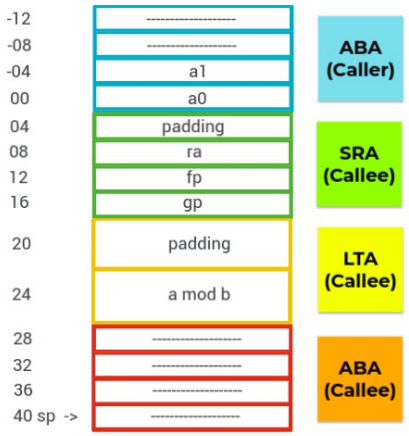

Figura 2: Stack en ejecución de la subrutina `mcd_euclides(unsigned int a, insigned int b)`

Se observa que todas las posiciones del stack se mantienen sin cambios con respecto a la anterior versión con excepción del registro `t0` esto se debe a que fue el que se utilizó para guardar el contenido de la variable local de la rutina, los demás registros están forzados a ser almacenados como respaldo por convención.

El código escrito para este módulo puede encontrarse en la sección apéndice A.2 código MIPS32 `mcd_euclides`.

3.2. Módulo `mcm_euclides.S`

La subrutina contenida dentro de este módulo (`mcm_euclides(unsigned int a, insigned int b)`), recibe dos parámetros y a su vez utiliza dos variables locales, por lo cual siguiendo la convención de la `ABI` para mips32 se reservan 40 bytes de memoria en el stack, y en general se distribuye como se muestra en la figura 3.

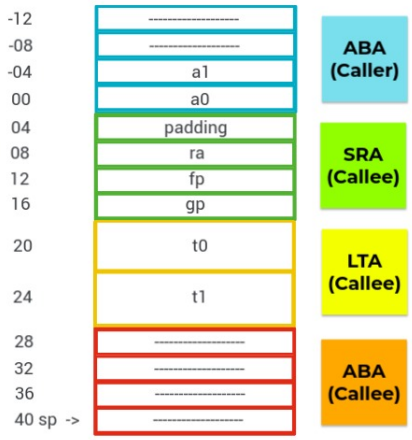

Figura 3: Stack general de la subrutina `mcm_euclides(unsigned int a, unsigned int b)`

Para esta rutina se observa que la estructura general del stack es muy parecida a la anterior, esto se debe a que son programas cortos y muy parecidos entre sí, de hecho el funcionamiento de

esta rutina depende de `mcd_euclides(unsigned int a, insigned int b)` y vista en alto nivel únicamente requiere dos operaciones aritméticas sencillas para obtener el resultado esperado.

Como se mencionó anteriormente la sección `ABA` (caller) no pertenece a la función en cuestión, sin embargo, para mantener coherencia con las imágenes que están siendo presentadas se incluyó de nuevo. Igualmente esta función confía en que el programa principal escrito en C respete la convención establecida al momento de invocarla.

Las sección `LRA` se repite debido a que es obligatoria tenerla y al ser una función no hoja se deben reservar 16 bytes con 4 de padding para esta sección y además los correspondientes 16 bytes extras reservados del `ABA` para la función invocada.

En esta oportunidad la función de alto nivel en C requiere dos variables locales y pese a que el código assembly era igualmente sencillo, se decidió seguir con esta idea de ser cuidadosos a la hora de programar en lugar de buscar eficiencia, por lo cual el `LTA` no requiere ningún tipo de padding. En ejecución únicamente se modificara esta sección y pasara a tomar los siguientes valores:

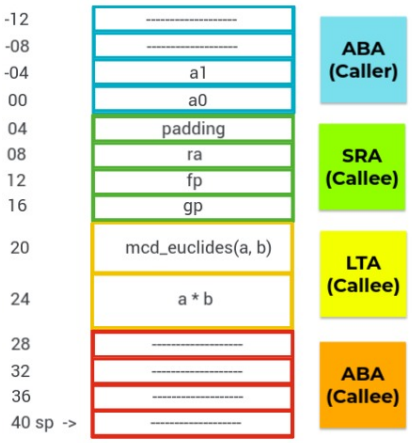

Figura 4: Stack general de la subrutina `mcm_euclides(unsigned int a, unsigned int b)`

Esta función tiene el inconveniente de que trabaja con enteros sin signos de 32 bits e involucra una multiplicación entre ambos, por lo tanto se corre el riesgo de obtener resultados inesperados si la multiplicación entre a y b es mayor a 2^{32}, ya que no podrá ser representado correctamente.

4. Compilación y ejecución

4.1. Compilación

Para compilar el programa implementado se incluyo un archivo `Makefile` en el repositorio, se debe ejecutar el comando `make` [3], esto generará los siguientes archivos:

- `mcm_euclides.o`: Código objeto del programa que contiene el programa para calcular el mínimo común múltiplo.
- `mcd_euclides.o`: Código objeto del programa que contiene el programa para calcular el máximo común múltiplo.
- `main.o`: Código objeto del programa principal.
- `common`: Código ejecutable.

En caso que se requiera limpiar automáticamente el `build` realizado se puede ejecutar el comando `make clean`, y removerá todos los archivos generados por el comando `make`.

4.2. Ejecución

Para ejecutar el archivo compilado:

```
./common [Opciones] PARÁMETRO1 PARÁMETRO2
```

4.2.1. Descripción de parámetros

`-V` --version: Muestra la versión y sale del programa.

`-h` --help: Muestra información de ayuda de cómo ejecutar el programa y sale del programa.

`-o` --output: Path para el archivo de salida (en caso de indicar el carácter '`-`' o no se utiliza el parámetro, se utilizará el `stdout`).

`-d` --divisor: Calcula únicamente el máximo común divisor.

`-m` --multiple: Calcula únicamente el mínimo común múltiplo.

4.2.2. Ejemplos de ejecución

Calcular el mínimo común múltiplo o máximo común divisor de dos números, por ejemplo 256 y 192, y mostrarlo en pantalla, se puede realizar de cualquiera de las siguientes maneras:

```
./common -d -o - 256 192
```

```
./common -m -o - 256 192
```

```
./common -m 256 192
```

```
./common -d 256 192
```

Si deseamos calcular el `mcm` y `mcd` a la vez, y mostrarlo en la pantalla, se puede realizar de las siguientes maneras:

```
./common -o - 256 192
```

```
./common 256 192
```

4.2.3. Pruebas automatizadas

Ejecutar el comando `make testing` para correr automáticamente todas las pruebas.

5. Pruebas

5.1. Prueba 1

Se prueba que al indicar un número negativo, se lo toma como un número fuera de rango (y no como un parámetro) en el caso de restringir por la opción `divisor` de la forma larga.

```
./common --divisor 10 -10

Error: N mero fuera de rango [2, 2147483647).
```

5.2. Prueba 2

Se prueba la opción de indicar el output como `stdout` pero sin indicar los números, con lo cual se espera un mensaje de error indicando el problema.

```
./common -o -

Error: Faltan los n meros.
```

5.3. Prueba 3

Se prueba indicando explícitamente ambas opciones de `divisor` y `multiple` y se obtiene el resultado esperado.

```
./common -d -m 10 20
10
20
```

5.4. Prueba 4

Se prueba indicando explícitamente la opción `divisor` de forma corta y se obtiene el resultado esperado en el `stdout`.

```
./common -o - -d 10 20
10
```

5.5. Prueba 5

Se prueba la opción `version` de la forma corta.

```
./common -V
Version 1.0
```

5.6. Prueba 6

Se prueba la opción más básica que reconoce el sistema con un número fuera de rango por exceso. Se espera que se obtenga mensaje indicando que algún número se encuentra fuera de rango.

```
./common 10 100000000000000000000

Error: N mero fuera de rango [2, 2147483647).
```

5.7. Prueba 7

Se prueba la opción más básica que reconoce el sistema con ambos números dentro del rango, pero en los extremos. Se espera obtener los valores correspondiente sin ningún error.

```
./common 2 2147483646
2
2147483646
```

5.8. Prueba 8

Se prueba valores fuera de rango por rango inferior. Se espera obtener mensaje de error 'Fuera de rango'

```
./common 1 1

Error: N mero fuera de rango [2, 2147483647).
```

5.9. Prueba 9

Se prueba el valor que se encuantra al principio del intervalo admitido, se espera obtener el resultado sin ningún mensaje de error. En particular, al tratarse de $f(2,2)$ se espera obtener $mcm(2,2) = 2$ y $mcd(2,2) = 2$

```
./common 2 2
2
2
```

5.10. Prueba 10

Se prueba ambos valores fuera de rango, con parámetros adicioneles, se espera el mensaje indicando 'Número fuera de rango'.

```
./common -o - -10 1

Error: N mero fuera de rango [2, 2147483647).
```

5.11. Prueba 11

Se prueba la opción más básica que reconoce el sistema con ambos números dentro del rango, pero en los extremos invertidos. Se espera obtener los valores correspondiente sin ningún error.

```
./common 2147483646 2
2
2147483646
```

5.12. Prueba 12

Se prueba indicando explícitamente la opción `multiple` de forma corta.

```
./common -m 10 20
20
```

5.13. Prueba 13

Se prueban ambas opciones, esta vez con el `output` al `stdout` explícito con el carácter '-'

```
./common -o - 10 20
10
20
```

5.14. Prueba 14

Se prueba la función `divisor` de forma larga con números medianos.

```
./common --divisor 192 168
24
```

5.15. Prueba 15

Se prueba con entrada dos números iguales no primos.

```
./common 10 10
10
10
```

5.16. Prueba 16

Se prueba el programa con las opciones básicas, con un sólo parámetro numérico, se espera un mensaje de error.

```
./common 10

Error: Faltan los n meros.
```

5.17. Prueba 17

Se prueba el programa con las opciones básicas, con ningún parámetro numérico, se espera un mensaje de error.

```
./common

Error: Faltan los n meros.
```

5.18. Prueba 18

Se prueba el programa con las opciones básicas, con un número negativo, se espera un mensaje de error de número fuera de rango.

```
./common -10 5

Error: Numero fuera de rango [2, 2147483647).
```

5.19. Prueba 19

Se prueban dos números iguales y primos.

```
./common 11 11
11
11
```

5.20. Prueba 20

Se prueba la opción de `help` en la versión corta.

```
./common -h
Usage:
        common -h
        common -V
        common [options] M N
Options:
        -h, --help Prints usage information.
        -V, --version Prints version information.
        -o, --output Path to output file.
        -d --divisor Just the divisor
        -m --multiple Just the multiple
```

5.21. Prueba 21

Se prueba la opción `version` de la forma corta, con parámetros adicionales, se espera que el sistema solo tome en consideración la primer acción y que ignore los demás parámetros.

```
./common -V 10 10
Version 1.0
```

6. Conclusiones

El objetivo principal del trabajo consistió en el entendimiento de la ABI de mips32, a partir de los módulos implementados en este lenguaje, esta convención es útil para evitar errores entre librerías escritas por distintos programadores y se comprobó que se cumple debido a que las partes escritas en C que fueron ensambladas por el compilador no tuvieron problemas con las escritas por nosotros, por ejemplo la función main reservó espacio automáticamente en el stack para las

funciones .S por lo cual no se corrompió la memoria de esta.
Por otra parte, al escribir las funciones de mcm y mcd de 'backup' en C, se pudo ver la diferencia y la complejidad en escribir el mismo programa a más bajo nivel que C, como es el caso de mips32. Los programas realizan lo mismo de una manera similar, pero al ser de tan bajo nivel mips32, escribir el código de las funciones lo volvió mucho más lento.

A. Código

A.1. Código en C

A.1.1. main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <stdbool.h>
#include <getopt.h>
#include <string.h>
#include <ctype.h>

#define MAX_FUNCTIONS_TO_RUN 2
#define STDIN_PARAM_IDENTIFIER "-"
#define INVALID_RESULT 0
#define MIN_VALUE_INPUT 2
#define MAX_VALUE_INPUT INT_MAX
#define CORRECT_INPUT 0
#define ALPHA_ERROR 1
#define NEGATIVE_ERROR 2

extern unsigned int mcd_euclides(unsigned int a, unsigned int b);
extern unsigned int mcm_euclides(unsigned int a, unsigned int b);

typedef unsigned int (*bin_operation_t) (unsigned int, unsigned int);

bool is_in_range(unsigned int value, unsigned int min, unsigned int
    ↪ max) {
  return min <= value && value < max;
}

int is_a_number(char* num){
  if (num[0] != '-' && !isdigit(num[0])) return ALPHA_ERROR;

  if (num[0] == '-' || isdigit(num[0])){
    for (size_t i = 1; i < strlen(num); i++){
      if (!isdigit(num[i])) return ALPHA_ERROR;
    }
    if (num[0] == '-') return NEGATIVE_ERROR;

    return CORRECT_INPUT;
  }

  return ALPHA_ERROR;
}

bool correct_input(char* num1, char* num2){
  int result_1 = is_a_number(num1);
  int result_2 = is_a_number(num2);
  if (result_1 + result_2 == CORRECT_INPUT) return true;

  if (result_1 == ALPHA_ERROR || result_2 == ALPHA_ERROR){
```

```
48      fprintf(stderr, "Error: deben ingresarse numeros no cadenas de
            ↪ texto\n");
49      return false;
50    }
51
52    fprintf(stderr, "Error: Los numeros ingresados deben ser positivos
          ↪ y estar en el rango [%d, %d]\n", MIN_VALUE_INPUT,
53      MAX_VALUE_INPUT);
54
55  }
56
57  unsigned int bin_operation_decorator(bin_operation_t operation,
        ↪ unsigned int a,
58                       unsigned int b) {
59    if (!is_in_range(a, MIN_VALUE_INPUT, MAX_VALUE_INPUT) ||
60      !is_in_range(b, MIN_VALUE_INPUT, MAX_VALUE_INPUT))
61      return INVALID_RESULT;
62    return operation(a, b);
63  }
64
65  void show_usage() {
66    printf("Usage:\n"
67           "  common -h\n"
68           "  common -V\n"
69           "  common [options] M N\n"
70           "Options:\n"
71           "  -h, --help Prints usage information.\n"
72           "  -V, --version Prints version information.\n"
73           "  -o, --output Path to output file.\n"
74           "  -d --divisor Just the divisor\n"
75           "  -m --multiple Just the multiple\n");
76  }
77
78  void show_version() {
79    printf("Version 1.0\n");
80  }
81
82  int parse_argv(int argc, char *argv[], FILE* output_file,
83           bin_operation_t functions[], int *functions_to_run) {
84    opterr = 0; // getopt suprime los mensajes de error
85    static struct option argument_options[] = {
86      {"help", no_argument, 0, 'h'},
87      {"version", no_argument, 0, 'V'},
88      {"output", required_argument, 0, 'o'},
89      {"divisor", no_argument, 0, 'd'},
90      {"multiple", no_argument, 0, 'm'},
91      {0, 0, 0, 0} // Lo pide getopt
92    };
93
94    int opt;
95    int option_index = 0;
96    while ((opt = getopt_long(argc, argv, "hVo:dm", argument_options,
97                              &option_index)) != -1) {
98      switch (opt) {
```

```
99      case 'h':
100       show_usage();
101       exit(EXIT_SUCCESS);
102     case 'V':
103       show_version();
104       exit(EXIT_SUCCESS);
105     case 'o':
106       if (strcmp(optarg, STDIN_PARAM_IDENTIFIER) != 0) {
107         output_file = fopen(optarg, "w");
108         if (!output_file) {
109           fprintf(stderr, "No se pudo abrir el archivo %s\n", optarg);
110           exit(EXIT_FAILURE);
111         }
112       }
113       break;
114     case 'd':
115       functions[(*functions_to_run)++] = mcd_euclides;
116       break;
117     case 'm':
118       functions[(*functions_to_run)++] = mcm_euclides;
119       break;
120     case 0:
121       printf("long option %s", argument_options[option_index].name);
122       if (optarg)
123         printf(" with arg %s", optarg);
124       printf("\n");
125       break;
126     case '?':
127       // TODO: Hacer un tratado mejor de errores
128       show_usage();
129       exit(EXIT_FAILURE);
130     default:
131       show_usage();
132       exit(EXIT_FAILURE);
133     }
134     option_index = 0;
135   }
136   return optind;
137 }
138
139
140 int main(int argc, char *argv[]) {
141   FILE* output_file = stdout;
142   bin_operation_t functions[] = {
143     mcd_euclides,
144     mcm_euclides
145   };
146   int functions_to_run = 0;
147
148   int last_index = parse_argv(argc, argv, output_file, functions,
149                               &functions_to_run);
150
151   if (!output_file) {
152     fprintf(stderr, "No se pudo acceder al archivo de salida.\n");
```

```
153       exit(EXIT_FAILURE);
154     }
155
156     if (!argv[last_index] || !argv[last_index + 1]) {
157       fprintf(stderr, "Faltan los n meros.\n");
158       show_usage();
159       exit(EXIT_FAILURE);
160     }
161
162     if(!correct_input(argv[last_index], argv[last_index + 1]))
        ↪ exit(EXIT_FAILURE);
163
164     unsigned int a = atoi(argv[last_index]);
165     unsigned int b = atoi(argv[last_index + 1]);
166
167     if (functions_to_run == 0)
168       functions_to_run = MAX_FUNCTIONS_TO_RUN;
169
170     for (int i = 0; i < functions_to_run; i++) {
171       unsigned int result = bin_operation_decorator(functions[i], a, b);
172       if (result == INVALID_RESULT) {
173         fprintf(stderr, "N mero fuera de rango [%d, %d).\n",
174                         MIN_VALUE_INPUT, MAX_VALUE_INPUT);
175         break;
176       }
177       fprintf(output_file, "%u\n", result);
178     }
179
180     if (output_file != stdout)
181       fclose(output_file);
182
183     return EXIT_SUCCESS;
184 }
```

A.2. Código MIPS32

A.2.1. mcd euclides.S

```
1  #include <sys/regdef.h>
2
3  .text
4  .align 2
5  .globl mcd_euclides
6  .ent mcd_euclides
7  /*
8   * Espacio reservado para el stack de la funcion
9   */
10 #define STACK_SIZE 40
11
12 /*
13  * Argument building area (ABA) caller
14  * 16 bytes
```

```
15  * - a1
16  * - a0
17  */
18 #define O_A1 STACK_SIZE + 4
19 #define O_A0 STACK_SIZE
20
21 /*
22  * Save register area (SRA) callee
23  * 16 bytes
24  * - padding
25  * - ra
26  * - fp
27  * - gp
28  */
29 #define O_RA STACK_SIZE - 8
30 #define O_FP STACK_SIZE - 12
31 #define O_GP STACK_SIZE - 16
32
33 /*
34  * Local and temporary area (LTA) callee
35  * 8 bytes
36  * - padding
37  * - t0
38  */
39 #define O_L0 STACK_SIZE - 24
40
41 /*
42  * Argument building area (ABA) callee
43  * 16 bytes
44  * Se reserva para el llamado recursivo
45  */
46
47 /* Firma de la funcion:
48  * unsigned int mcd_euclides(unsigned int a, unsigned int b);
49  */
50 mcd_euclides:
51   .frame fp, STACK_SIZE, ra
52   addiu sp, sp, -STACK_SIZE
53   sw ra, O_RA(sp)
54   sw fp, O_FP(sp)
55   .cprestore O_GP
56   move fp, sp
57   sw a0, O_A0(fp)
58   sw a1, O_A1(fp)
59
60   /* unsigned int r = 0 */
61   move t0, zero
62   sw t0, O_L0(fp)
63
64   /* if (b == 0) */
65   lw t0, O_A1(fp)
66   beqz t0, return
67
68   /* r = (a %b); */
```

```
69    lw t0, O_A0(fp)
70    lw t1, O_A1(fp)
71    divu t0, t0, t1
72    mfhi t0
73    sw t0, O_L0(fp)
74
75  return_recursivo:
76    /* return mcd_euclides(b, r); */
77    lw a0, O_A1(fp)
78    lw a1, O_L0(fp)
79    jal mcd_euclides
80
81    lw ra, O_RA(sp)
82    lw fp, O_FP(sp)
83    lw gp, O_GP(sp)
84    addiu sp, sp, STACK_SIZE
85    jr ra
86
87  return:
88    /* return a; */
89    lw v0, O_A0(sp)
90
91    lw ra, O_RA(sp)
92    lw fp, O_FP(sp)
93    lw gp, O_GP(sp)
94    addiu sp, sp, STACK_SIZE
95    jr ra
96  .end mcd_euclides
```

A.2.2. mcm euclides.S

```
1  #include <sys/regdef.h>
2
3  .text
4  .align 2
5  .globl mcm_euclides
6  .extern mcd_euclides
7  .ent mcm_euclides
8  /*
9   * Espacio reservado para el stack de la funcion
10  */
11 #define STACK_SIZE 40
12
13 /*
14  * Argument building area (ABA) caller
15  * 16 bytes
16  * - a1
17  * - a0
18  */
19 #define O_A1 STACK_SIZE + 4
20 #define O_A0 STACK_SIZE
21
```

```
22 /*
23  * Save register area (SRA) callee
24  * 16 bytes
25  * - padding
26  * - ra
27  * - fp
28  * - gp
29  */
30 #define O_RA STACK_SIZE - 8
31 #define O_FP STACK_SIZE - 12
32 #define O_GP STACK_SIZE - 16
33
34 /*
35  * Local and temporary area (LTA) callee
36  * 8 bytes
37  * - t0
38  * - t1
39  */
40 #define O_L0 STACK_SIZE - 20
41 #define O_L1 STACK_SIZE - 24
42
43 /*
44  * Argument building area (ABA) callee
45  * 16 bytes
46  * Se reserva para el llamado a mcd_euclides
47  */
48
49 /* Firma de la funcion:
50  * unsigned int mcm_euclides(unsigned int a, unsigned int b);
51  */
52 mcm_euclides:
53   .frame fp, STACK_SIZE, ra
54   addiu sp, sp, -STACK_SIZE
55   sw ra, O_RA(sp)
56   sw fp, O_FP(sp)
57   .cprestore O_GP
58   move fp, sp
59   sw a0, O_A0(fp)
60   sw a1, O_A1(fp)
61
62   /* unsigned int mcd = mcd_euclides(a, b); */
63   lw a0, O_A0(fp)
64   lw a1, O_A1(fp)
65   jal mcd_euclides
66   sw v0, O_L0(sp)
67
68   /* unsigned int a_x_b = (a * b); */
69   lw t0, O_A0(fp)
70   lw t1, O_A1(fp)
71   multu t0, t1
72   mflo t0
73   sw t0, O_L1(fp)
74
75 return:
```

```
76    /* return (a_x_b / mcd); */
77    lw t0, O_L0(fp)
78    lw t1, O_L1(fp)
79    divu v0, t1, t0
80
81    lw ra, O_RA(sp)
82    lw fp, O_FP(sp)
83    lw gp, O_GP(sp)
84    addiu sp, sp, STACK_SIZE
85    jr ra
86  .end mcm_euclides
```

A.2.3. 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     .text
9     .align 2
10    .globl is_in_range
11    .set nomips16
12    .set nomicromips
13    .ent is_in_range
14    .type is_in_range, @function
15  is_in_range:
16    .frame $fp,8,$31    # vars= 0, regs= 1/0, args= 0, gp= 0
17    .mask 0x40000000,-4
18    .fmask 0x00000000,0
19    .set noreorder
20    .set nomacro
21    addiu $sp,$sp,-8
22    sw $fp,4($sp)
23    move $fp,$sp
24    sw $4,8($fp)
25    sw $5,12($fp)
26    sw $6,16($fp)
27    lw $3,12($fp)
28    lw $2,8($fp)
29    sltu $2,$2,$3
30    bne $2,$0,$L2
31    nop
32
33    lw $3,8($fp)
34    lw $2,16($fp)
35    sltu $2,$3,$2
36    beq $2,$0,$L2
37    nop
38
```

```
39    li   $2,1        # 0x1
40    b $L3
41    nop
42
43  $L2:
44    move   $2,$0
45  $L3:
46    andi   $2,$2,0x1
47    andi   $2,$2,0x00ff
48    move   $sp,$fp
49    lw   $fp,4($sp)
50    addiu $sp,$sp,8
51    jr   $31
52    nop
53
54    .set   macro
55    .set   reorder
56    .end   is_in_range
57    .size is_in_range, .-is_in_range
58    .align   2
59    .globl   is_a_number
60    .set   nomips16
61    .set   nomicromips
62    .ent   is_a_number
63    .type is_a_number, @function
64  is_a_number:
65    .frame   $fp,40,$31     # vars= 8, regs= 2/0, args= 16, gp= 8
66    .mask 0xc0000000,-4
67    .fmask   0x00000000,0
68    .set   noreorder
69    .cpload $25
70    .set   nomacro
71    addiu $sp,$sp,-40
72    sw   $31,36($sp)
73    sw   $fp,32($sp)
74    move   $fp,$sp
75    .cprestore   16
76    sw   $4,40($fp)
77    lw   $2,40($fp)
78    lb   $3,0($2)
79    li   $2,45       # 0x2d
80    beq $3,$2,$L6
81    nop
82
83    lw   $2,%call16(__ctype_b_loc)($28)
84    move   $25,$2
85    .reloc   1f,R_MIPS_JALR,__ctype_b_loc
86  1:  jalr   $25
87    nop
88
89    lw   $28,16($fp)
90    lw   $3,0($2)
91    lw   $2,40($fp)
92    lb   $2,0($2)
```

```
93    sll $2,$2,1
94    addu  $2,$3,$2
95    lhu $2,0($2)
96    andi  $2,$2,0x8
97    bne $2,$0,$L6
98    nop
99
100   li  $2,1        # 0x1
101   b $L7
102   nop
103
104 $L6:
105   lw  $2,40($fp)
106   lb  $3,0($2)
107   li  $2,45       # 0x2d
108   beq $3,$2,$L8
109   nop
110
111   lw  $2,%call16(__ctype_b_loc)($28)
112   move  $25,$2
113   .reloc  1f,R_MIPS_JALR,__ctype_b_loc
114 1:  jalr  $25
115   nop
116
117   lw  $28,16($fp)
118   lw  $3,0($2)
119   lw  $2,40($fp)
120   lb  $2,0($2)
121   sll $2,$2,1
122   addu  $2,$3,$2
123   lhu $2,0($2)
124   andi  $2,$2,0x8
125   beq $2,$0,$L9
126   nop
127
128 $L8:
129   li  $2,1        # 0x1
130   sw  $2,24($fp)
131   b $L10
132   nop
133
134 $L12:
135   lw  $2,%call16(__ctype_b_loc)($28)
136   move  $25,$2
137   .reloc  1f,R_MIPS_JALR,__ctype_b_loc
138 1:  jalr  $25
139   nop
140
141   lw  $28,16($fp)
142   lw  $3,0($2)
143   lw  $4,40($fp)
144   lw  $2,24($fp)
145   addu  $2,$4,$2
146   lb  $2,0($2)
```

```
147   sll $2,$2,1
148   addu  $2,$3,$2
149   lhu $2,0($2)
150   andi  $2,$2,0x8
151   bne $2,$0,$L11
152   nop
153
154   li  $2,1        # 0x1
155   b $L7
156   nop
157
158 $L11:
159   lw  $2,24($fp)
160   addiu $2,$2,1
161   sw  $2,24($fp)
162 $L10:
163   lw  $4,40($fp)
164   lw  $2,%call16(strlen)($28)
165   move  $25,$2
166   .reloc  1f,R_MIPS_JALR,strlen
167 1:  jalr  $25
168   nop
169
170   lw  $28,16($fp)
171   move  $3,$2
172   lw  $2,24($fp)
173   sltu  $2,$2,$3
174   bne $2,$0,$L12
175   nop
176
177   lw  $2,40($fp)
178   lb  $3,0($2)
179   li  $2,45       # 0x2d
180   bne $3,$2,$L13
181   nop
182
183   li  $2,2        # 0x2
184   b $L7
185   nop
186
187 $L13:
188   move  $2,$0
189   b $L7
190   nop
191
192 $L9:
193   li  $2,1        # 0x1
194 $L7:
195   move  $sp,$fp
196   lw  $31,36($sp)
197   lw  $fp,32($sp)
198   addiu $sp,$sp,40
199   jr  $31
200   nop
```

```
201
202     .set    macro
203     .set    reorder
204     .end    is_a_number
205     .size   is_a_number, .-is_a_number
206     .rdata
207     .align  2
208 $LC0:
209     .ascii  "Error: deben ingresarse numeros no cadenas de
            ↪ texto\012\000"
210     .align  2
211 $LC1:
212     .ascii  "Error: Los numeros ingresados deben ser positivos y esta"
213     .ascii  "r en el rango [%d, %d]\012\000"
214     .text
215     .align  2
216     .globl  correct_input
217     .set    nomips16
218     .set    nomicromips
219     .ent    correct_input
220     .type   correct_input, @function
221 correct_input:
222     .frame  $fp,40,$31      # vars= 8, regs= 2/0, args= 16, gp= 8
223     .mask   0xc0000000,-4
224     .fmask  0x00000000,0
225     .set    noreorder
226     .cpload $25
227     .set    nomacro
228     addiu   $sp,$sp,-40
229     sw      $31,36($sp)
230     sw      $fp,32($sp)
231     move    $fp,$sp
232     .cprestore  16
233     sw      $4,40($fp)
234     sw      $5,44($fp)
235     lw      $4,40($fp)
236     lw      $2,%got(is_a_number)($28)
237     move    $25,$2
238     .reloc  1f,R_MIPS_JALR,is_a_number
239 1:  jalr    $25
240     nop
241
242     lw      $28,16($fp)
243     sw      $2,24($fp)
244     lw      $4,44($fp)
245     lw      $2,%got(is_a_number)($28)
246     move    $25,$2
247     .reloc  1f,R_MIPS_JALR,is_a_number
248 1:  jalr    $25
249     nop
250
251     lw      $28,16($fp)
252     sw      $2,28($fp)
253     lw      $3,24($fp)
```

```
254   lw   $2,28($fp)
255   addu   $2,$3,$2
256   bne $2,$0,$L15
257   nop
258
259   li   $2,1        # 0x1
260   b $L16
261   nop
262
263 $L15:
264   lw   $3,24($fp)
265   li   $2,1        # 0x1
266   beq $3,$2,$L17
267   nop
268
269   lw   $3,28($fp)
270   li   $2,1        # 0x1
271   bne $3,$2,$L18
272   nop
273
274 $L17:
275   lw   $2,%got(stderr)($28)
276   lw   $2,0($2)
277   move   $7,$2
278   li   $6,52       # 0x34
279   li   $5,1        # 0x1
280   lw   $2,%got($LC0)($28)
281   addiu $4,$2,%lo($LC0)
282   lw   $2,%call16(fwrite)($28)
283   move   $25,$2
284   .reloc   1f,R_MIPS_JALR,fwrite
285 1:   jalr   $25
286   nop
287
288   lw   $28,16($fp)
289   move   $2,$0
290   b $L16
291   nop
292
293 $L18:
294   lw   $2,%got(stderr)($28)
295   lw   $3,0($2)
296   li   $2,2147418112       # 0x7fff0000
297   ori $7,$2,0xffff
298   li   $6,2        # 0x2
299   lw   $2,%got($LC1)($28)
300   addiu $5,$2,%lo($LC1)
301   move   $4,$3
302   lw   $2,%call16(fprintf)($28)
303   move   $25,$2
304   .reloc   1f,R_MIPS_JALR,fprintf
305 1:   jalr   $25
306   nop
307
```

```
308   lw   $28,16($fp)
309   b $L14
310   nop
311
312 $L16:
313 $L14:
314   move   $sp,$fp
315   lw   $31,36($sp)
316   lw   $fp,32($sp)
317   addiu $sp,$sp,40
318   jr   $31
319   nop
320
321   .set   macro
322   .set   reorder
323   .end   correct_input
324   .size correct_input, .-correct_input
325   .align   2
326   .globl   bin_operation_decorator
327   .set   nomips16
328   .set   nomicromips
329   .ent   bin_operation_decorator
330   .type bin_operation_decorator, @function
331 bin_operation_decorator:
332   .frame   $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
333   .mask 0xc0000000,-4
334   .fmask   0x00000000,0
335   .set   noreorder
336   .cpload $25
337   .set   nomacro
338   addiu $sp,$sp,-32
339   sw   $31,28($sp)
340   sw   $fp,24($sp)
341   move   $fp,$sp
342   .cprestore   16
343   sw   $4,32($fp)
344   sw   $5,36($fp)
345   sw   $6,40($fp)
346   li   $2,2147418112      # 0x7fff0000
347   ori $6,$2,0xffff
348   li   $5,2       # 0x2
349   lw   $4,36($fp)
350   lw   $2,%got(is_in_range)($28)
351   move   $25,$2
352   .reloc   1f,R_MIPS_JALR,is_in_range
353 1:   jalr   $25
354   nop
355
356   lw   $28,16($fp)
357   xori   $2,$2,0x1
358   andi   $2,$2,0x00ff
359   bne $2,$0,$L20
360   nop
361
```

```
362   li   $2,2147418112      # 0x7fff0000
363   ori  $6,$2,0xffff
364   li   $5,2        # 0x2
365   lw   $4,40($fp)
366   lw   $2,%got(is_in_range)($28)
367   move  $25,$2
368   .reloc  1f,R_MIPS_JALR,is_in_range
369 1:  jalr  $25
370   nop
371
372   lw   $28,16($fp)
373   xori  $2,$2,0x1
374   andi  $2,$2,0x00ff
375   beq $2,$0,$L21
376   nop
377
378 $L20:
379   move  $2,$0
380   b $L22
381   nop
382
383 $L21:
384   lw   $2,32($fp)
385   lw   $5,40($fp)
386   lw   $4,36($fp)
387   move  $25,$2
388   jalr  $25
389   nop
390
391   lw   $28,16($fp)
392 $L22:
393   move  $sp,$fp
394   lw   $31,28($sp)
395   lw   $fp,24($sp)
396   addiu $sp,$sp,32
397   jr   $31
398   nop
399
400   .set  macro
401   .set  reorder
402   .end  bin_operation_decorator
403   .size bin_operation_decorator, .-bin_operation_decorator
404   .rdata
405   .align  2
406 $LC2:
407   .ascii  "Usage:\012\011common -h\012\011common -V\012\011common ["
408   .ascii  "options] M N\012Options:\012\011-h, --help Prints usage "
409   .ascii  "information.\012\011-V, --version Prints version informa"
410   .ascii  "tion.\012\011-o, --output Path to output file.\012\011-d"
411   .ascii  " --divisor Just the divisor\012\011-m --multiple Just th"
412   .ascii  "e multiple\000"
413   .text
414   .align  2
415   .globl  show_usage
```

```
416     .set    nomips16
417     .set    nomicromips
418     .ent    show_usage
419     .type   show_usage, @function
420 show_usage:
421     .frame  $fp,32,$31      # vars= 0, regs= 2/0, args= 16, gp= 8
422     .mask   0xc0000000,-4
423     .fmask  0x00000000,0
424     .set    noreorder
425     .cpload $25
426     .set    nomacro
427     addiu   $sp,$sp,-32
428     sw      $31,28($sp)
429     sw      $fp,24($sp)
430     move    $fp,$sp
431     .cprestore  16
432     lw      $2,%got($LC2)($28)
433     addiu   $4,$2,%lo($LC2)
434     lw      $2,%call16(puts)($28)
435     move    $25,$2
436     .reloc  1f,R_MIPS_JALR,puts
437 1:  jalr    $25
438     nop
439
440     lw      $28,16($fp)
441     nop
442     move    $sp,$fp
443     lw      $31,28($sp)
444     lw      $fp,24($sp)
445     addiu   $sp,$sp,32
446     jr      $31
447     nop
448
449     .set    macro
450     .set    reorder
451     .end    show_usage
452     .size   show_usage, .-show_usage
453     .rdata
454     .align  2
455 $LC3:
456     .ascii  "Version 1.0\000"
457     .text
458     .align  2
459     .globl  show_version
460     .set    nomips16
461     .set    nomicromips
462     .ent    show_version
463     .type   show_version, @function
464 show_version:
465     .frame  $fp,32,$31      # vars= 0, regs= 2/0, args= 16, gp= 8
466     .mask   0xc0000000,-4
467     .fmask  0x00000000,0
468     .set    noreorder
469     .cpload $25
```

```
470     .set    nomacro
471     addiu   $sp,$sp,-32
472     sw      $31,28($sp)
473     sw      $fp,24($sp)
474     move    $fp,$sp
475     .cprestore  16
476     lw      $2,%got($LC3)($28)
477     addiu   $4,$2,%lo($LC3)
478     lw      $2,%call16(puts)($28)
479     move    $25,$2
480     .reloc  1f,R_MIPS_JALR,puts
481 1:  jalr    $25
482     nop
483
484     lw      $28,16($fp)
485     nop
486     move    $sp,$fp
487     lw      $31,28($sp)
488     lw      $fp,24($sp)
489     addiu   $sp,$sp,32
490     jr      $31
491     nop
492
493     .set    macro
494     .set    reorder
495     .end    show_version
496     .size   show_version, .-show_version
497     .rdata
498     .align  2
499 $LC4:
500     .ascii  "-\000"
501     .align  2
502 $LC5:
503     .ascii  "w\000"
504     .align  2
505 $LC6:
506     .ascii  "No se pudo abrir el archivo %s\012\000"
507     .align  2
508 $LC7:
509     .ascii  "long option %s\000"
510     .align  2
511 $LC8:
512     .ascii  " with arg %s\000"
513     .align  2
514 $LC9:
515     .ascii  "hVo:dm\000"
516     .text
517     .align  2
518     .globl  parse_argv
519     .set    nomips16
520     .set    nomicromips
521     .ent    parse_argv
522     .type   parse_argv, @function
523 parse_argv:
```

```
524   .frame  $fp,48,$31     # vars= 8, regs= 2/0, args= 24, gp= 8
525   .mask 0xc0000000,-4
526   .fmask  0x00000000,0
527   .set  noreorder
528   .cpload $25
529   .set  nomacro
530   addiu $sp,$sp,-48
531   sw  $31,44($sp)
532   sw  $fp,40($sp)
533   move  $fp,$sp
534   .cprestore  24
535   sw  $4,48($fp)
536   sw  $5,52($fp)
537   sw  $6,56($fp)
538   sw  $7,60($fp)
539   lw  $2,%got(opterr)($28)
540   sw  $0,0($2)
541   sw  $0,36($fp)
542   b $L26
543   nop
544
545 $L39:
546   lw  $2,32($fp)
547   li  $3,100      # 0x64
548   beq $2,$3,$L28
549   nop
550
551   slt $3,$2,101
552   beq $3,$0,$L29
553   nop
554
555   li  $3,63     # 0x3f
556   beq $2,$3,$L30
557   nop
558
559   li  $3,86     # 0x56
560   beq $2,$3,$L31
561   nop
562
563   beq $2,$0,$L32
564   nop
565
566   b $L27
567   nop
568
569 $L29:
570   li  $3,109      # 0x6d
571   beq $2,$3,$L33
572   nop
573
574   li  $3,111      # 0x6f
575   beq $2,$3,$L34
576   nop
577
```

```
578   li   $3,104        # 0x68
579   bne  $2,$3,$L27
580   nop
581
582   lw   $2,%got(show_usage)($28)
583   move  $25,$2
584   .reloc  1f,R_MIPS_JALR,show_usage
585 1:  jalr  $25
586   nop
587
588   lw   $28,24($fp)
589   move  $4,$0
590   lw   $2,%call16(exit)($28)
591   move  $25,$2
592   .reloc  1f,R_MIPS_JALR,exit
593 1:  jalr  $25
594   nop
595
596 $L31:
597   lw   $2,%got(show_version)($28)
598   move  $25,$2
599   .reloc  1f,R_MIPS_JALR,show_version
600 1:  jalr  $25
601   nop
602
603   lw   $28,24($fp)
604   move  $4,$0
605   lw   $2,%call16(exit)($28)
606   move  $25,$2
607   .reloc  1f,R_MIPS_JALR,exit
608 1:  jalr  $25
609   nop
610
611 $L34:
612   lw   $2,%got(optarg)($28)
613   lw   $3,0($2)
614   lw   $2,%got($LC4)($28)
615   addiu  $5,$2,%lo($LC4)
616   move  $4,$3
617   lw   $2,%call16(strcmp)($28)
618   move  $25,$2
619   .reloc  1f,R_MIPS_JALR,strcmp
620 1:  jalr  $25
621   nop
622
623   lw   $28,24($fp)
624   beq  $2,$0,$L41
625   nop
626
627   lw   $2,%got(optarg)($28)
628   lw   $3,0($2)
629   lw   $2,%got($LC5)($28)
630   addiu  $5,$2,%lo($LC5)
631   move  $4,$3
```

```
632   lw   $2,%call16(fopen)($28)
633   move   $25,$2
634   .reloc   1f,R_MIPS_JALR,fopen
635 1:   jalr   $25
636   nop
637
638   lw   $28,24($fp)
639   sw   $2,56($fp)
640   lw   $2,56($fp)
641   bne $2,$0,$L41
642   nop
643
644   lw   $2,%got(stderr)($28)
645   lw   $3,0($2)
646   lw   $2,%got(optarg)($28)
647   lw   $2,0($2)
648   move   $6,$2
649   lw   $2,%got($LC6)($28)
650   addiu $5,$2,%lo($LC6)
651   move   $4,$3
652   lw   $2,%call16(fprintf)($28)
653   move   $25,$2
654   .reloc   1f,R_MIPS_JALR,fprintf
655 1:   jalr   $25
656   nop
657
658   lw   $28,24($fp)
659   li   $4,1        # 0x1
660   lw   $2,%call16(exit)($28)
661   move   $25,$2
662   .reloc   1f,R_MIPS_JALR,exit
663 1:   jalr   $25
664   nop
665
666 $L28:
667   lw   $2,64($fp)
668   lw   $2,0($2)
669   addiu $4,$2,1
670   lw   $3,64($fp)
671   sw   $4,0($3)
672   sll $2,$2,2
673   lw   $3,60($fp)
674   addu   $2,$3,$2
675   lw   $3,%got(mcd_euclides)($28)
676   sw   $3,0($2)
677   b $L37
678   nop
679
680 $L33:
681   lw   $2,64($fp)
682   lw   $2,0($2)
683   addiu $4,$2,1
684   lw   $3,64($fp)
685   sw   $4,0($3)
```

```
686   sll $2,$2,2
687   lw  $3,60($fp)
688   addu  $2,$3,$2
689   lw  $3,%got(mcm_euclides)($28)
690   sw  $3,0($2)
691   b $L37
692   nop
693
694 $L32:
695   lw  $3,36($fp)
696   lw  $2,%got(argument_options.2692)($28)
697   sll $3,$3,4
698   addiu $2,$2,%lo(argument_options.2692)
699   addu  $2,$3,$2
700   lw  $2,0($2)
701   move  $5,$2
702   lw  $2,%got($LC7)($28)
703   addiu $4,$2,%lo($LC7)
704   lw  $2,%call16(printf)($28)
705   move  $25,$2
706   .reloc  1f,R_MIPS_JALR,printf
707 1:  jalr  $25
708   nop
709
710   lw  $28,24($fp)
711   lw  $2,%got(optarg)($28)
712   lw  $2,0($2)
713   beq $2,$0,$L38
714   nop
715
716   lw  $2,%got(optarg)($28)
717   lw  $2,0($2)
718   move  $5,$2
719   lw  $2,%got($LC8)($28)
720   addiu $4,$2,%lo($LC8)
721   lw  $2,%call16(printf)($28)
722   move  $25,$2
723   .reloc  1f,R_MIPS_JALR,printf
724 1:  jalr  $25
725   nop
726
727   lw  $28,24($fp)
728 $L38:
729   li  $4,10       # 0xa
730   lw  $2,%call16(putchar)($28)
731   move  $25,$2
732   .reloc  1f,R_MIPS_JALR,putchar
733 1:  jalr  $25
734   nop
735
736   lw  $28,24($fp)
737   b $L37
738   nop
739
```

```
740 $L30:
741   lw   $2,%got(show_usage)($28)
742   move   $25,$2
743   .reloc   1f,R_MIPS_JALR,show_usage
744 1:   jalr   $25
745   nop
746
747   lw   $28,24($fp)
748   li   $4,1        # 0x1
749   lw   $2,%call16(exit)($28)
750   move   $25,$2
751   .reloc   1f,R_MIPS_JALR,exit
752 1:   jalr   $25
753   nop
754
755 $L27:
756   lw   $2,%got(show_usage)($28)
757   move   $25,$2
758   .reloc   1f,R_MIPS_JALR,show_usage
759 1:   jalr   $25
760   nop
761
762   lw   $28,24($fp)
763   li   $4,1        # 0x1
764   lw   $2,%call16(exit)($28)
765   move   $25,$2
766   .reloc   1f,R_MIPS_JALR,exit
767 1:   jalr   $25
768   nop
769
770 $L41:
771   nop
772 $L37:
773   sw   $0,36($fp)
774 $L26:
775   addiu  $2,$fp,36
776   sw   $2,16($sp)
777   lw   $2,%got(argument_options.2692)($28)
778   addiu  $7,$2,%lo(argument_options.2692)
779   lw   $2,%got($LC9)($28)
780   addiu  $6,$2,%lo($LC9)
781   lw   $5,52($fp)
782   lw   $4,48($fp)
783   lw   $2,%call16(getopt_long)($28)
784   move   $25,$2
785   .reloc   1f,R_MIPS_JALR,getopt_long
786 1:   jalr   $25
787   nop
788
789   lw   $28,24($fp)
790   sw   $2,32($fp)
791   lw   $3,32($fp)
792   li   $2,-1       # 0xffffffffffffffff
793   bne  $3,$2,$L39
```

```
794   nop
795
796   lw   $2,%got(optind)($28)
797   lw   $2,0($2)
798   move   $sp,$fp
799   lw   $31,44($sp)
800   lw   $fp,40($sp)
801   addiu  $sp,$sp,48
802   jr   $31
803   nop
804
805   .set   macro
806   .set   reorder
807   .end   parse_argv
808   .size  parse_argv, .-parse_argv
809   .rdata
810   .align   2
811 $LC10:
812   .ascii   "No se pudo acceder al archivo de salida.\012\000"
813   .align   2
814 $LC11:
815   .ascii   "Faltan los n\303\272meros.\012\000"
816   .align   2
817 $LC12:
818   .ascii   "N\303\272mero fuera de rango [%d, %d).\012\000"
819   .align   2
820 $LC13:
821   .ascii   "%u\012\000"
822   .text
823   .align   2
824   .globl   main
825   .set   nomips16
826   .set   nomicromips
827   .ent   main
828   .type  main, @function
829 main:
830   .frame   $fp,80,$31    # vars= 40, regs= 2/0, args= 24, gp= 8
831   .mask  0xc0000000,-4
832   .fmask   0x00000000,0
833   .set   noreorder
834   .cpload  $25
835   .set   nomacro
836   addiu  $sp,$sp,-80
837   sw   $31,76($sp)
838   sw   $fp,72($sp)
839   move   $fp,$sp
840   .cprestore   24
841   sw   $4,80($fp)
842   sw   $5,84($fp)
843   lw   $2,%got(stdout)($28)
844   lw   $2,0($2)
845   sw   $2,36($fp)
846   lw   $2,%got(mcd_euclides)($28)
847   sw   $2,56($fp)
```

```
848   lw   $2,%got(mcm_euclides)($28)
849   sw   $2,60($fp)
850   sw   $0,64($fp)
851   addiu $3,$fp,56
852   addiu $2,$fp,64
853   sw   $2,16($sp)
854   move  $7,$3
855   lw   $6,36($fp)
856   lw   $5,84($fp)
857   lw   $4,80($fp)
858   lw   $2,%got(parse_argv)($28)
859   move  $25,$2
860   .reloc  1f,R_MIPS_JALR,parse_argv
861 1:  jalr  $25
862   nop
863
864   lw   $28,24($fp)
865   sw   $2,40($fp)
866   lw   $2,36($fp)
867   bne  $2,$0,$L43
868   nop
869
870   lw   $2,%got(stderr)($28)
871   lw   $2,0($2)
872   move  $7,$2
873   li   $6,41      # 0x29
874   li   $5,1       # 0x1
875   lw   $2,%got($LC10)($28)
876   addiu $4,$2,%lo($LC10)
877   lw   $2,%call16(fwrite)($28)
878   move  $25,$2
879   .reloc  1f,R_MIPS_JALR,fwrite
880 1:  jalr  $25
881   nop
882
883   lw   $28,24($fp)
884   li   $4,1       # 0x1
885   lw   $2,%call16(exit)($28)
886   move  $25,$2
887   .reloc  1f,R_MIPS_JALR,exit
888 1:  jalr  $25
889   nop
890
891 $L43:
892   lw   $2,40($fp)
893   sll  $2,$2,2
894   lw   $3,84($fp)
895   addu  $2,$3,$2
896   lw   $2,0($2)
897   beq  $2,$0,$L44
898   nop
899
900   lw   $2,40($fp)
901   addiu $2,$2,1
```

```
902   sll $2,$2,2
903   lw  $3,84($fp)
904   addu  $2,$3,$2
905   lw  $2,0($2)
906   bne $2,$0,$L45
907   nop
908
909 $L44:
910   lw  $2,%got(stderr)($28)
911   lw  $2,0($2)
912   move  $7,$2
913   li  $6,21       # 0x15
914   li  $5,1        # 0x1
915   lw  $2,%got($LC11)($28)
916   addiu $4,$2,%lo($LC11)
917   lw  $2,%call16(fwrite)($28)
918   move  $25,$2
919   .reloc  1f,R_MIPS_JALR,fwrite
920 1:  jalr  $25
921   nop
922
923   lw  $28,24($fp)
924   lw  $2,%got(show_usage)($28)
925   move  $25,$2
926   .reloc  1f,R_MIPS_JALR,show_usage
927 1:  jalr  $25
928   nop
929
930   lw  $28,24($fp)
931   li  $4,1        # 0x1
932   lw  $2,%call16(exit)($28)
933   move  $25,$2
934   .reloc  1f,R_MIPS_JALR,exit
935 1:  jalr  $25
936   nop
937
938 $L45:
939   lw  $2,40($fp)
940   sll $2,$2,2
941   lw  $3,84($fp)
942   addu  $2,$3,$2
943   lw  $4,0($2)
944   lw  $2,40($fp)
945   addiu $2,$2,1
946   sll $2,$2,2
947   lw  $3,84($fp)
948   addu  $2,$3,$2
949   lw  $2,0($2)
950   move  $5,$2
951   lw  $2,%got(correct_input)($28)
952   move  $25,$2
953   .reloc  1f,R_MIPS_JALR,correct_input
954 1:  jalr  $25
955   nop
```

```
956
957     lw   $28,24($fp)
958     xori   $2,$2,0x1
959     andi   $2,$2,0x00ff
960     beq $2,$0,$L46
961     nop
962
963     li   $4,1        # 0x1
964     lw   $2,%call16(exit)($28)
965     move   $25,$2
966     .reloc   1f,R_MIPS_JALR,exit
967 1:  jalr   $25
968     nop
969
970 $L46:
971     lw   $2,40($fp)
972     sll $2,$2,2
973     lw   $3,84($fp)
974     addu   $2,$3,$2
975     lw   $2,0($2)
976     move   $4,$2
977     lw   $2,%call16(atoi)($28)
978     move   $25,$2
979     .reloc   1f,R_MIPS_JALR,atoi
980 1:  jalr   $25
981     nop
982
983     lw   $28,24($fp)
984     sw   $2,44($fp)
985     lw   $2,40($fp)
986     addiu $2,$2,1
987     sll $2,$2,2
988     lw   $3,84($fp)
989     addu   $2,$3,$2
990     lw   $2,0($2)
991     move   $4,$2
992     lw   $2,%call16(atoi)($28)
993     move   $25,$2
994     .reloc   1f,R_MIPS_JALR,atoi
995 1:  jalr   $25
996     nop
997
998     lw   $28,24($fp)
999     sw   $2,48($fp)
1000    lw   $2,64($fp)
1001    bne $2,$0,$L47
1002    nop
1003
1004    li   $2,2        # 0x2
1005    sw   $2,64($fp)
1006 $L47:
1007    sw   $0,32($fp)
1008    b $L48
1009    nop
```

```
1010
1011 $L51:
1012   lw   $2,32($fp)
1013   sll  $2,$2,2
1014   addiu $3,$fp,32
1015   addu  $2,$3,$2
1016   lw   $2,24($2)
1017   lw   $6,48($fp)
1018   lw   $5,44($fp)
1019   move  $4,$2
1020   lw   $2,%got(bin_operation_decorator)($28)
1021   move  $25,$2
1022   .reloc  1f,R_MIPS_JALR,bin_operation_decorator
1023 1:  jalr  $25
1024   nop
1025
1026   lw   $28,24($fp)
1027   sw   $2,52($fp)
1028   lw   $2,52($fp)
1029   bne  $2,$0,$L49
1030   nop
1031
1032   lw   $2,%got(stderr)($28)
1033   lw   $3,0($2)
1034   li   $2,2147418112      # 0x7fff0000
1035   ori  $7,$2,0xffff
1036   li   $6,2       # 0x2
1037   lw   $2,%got($LC12)($28)
1038   addiu $5,$2,%lo($LC12)
1039   move  $4,$3
1040   lw   $2,%call16(fprintf)($28)
1041   move  $25,$2
1042   .reloc  1f,R_MIPS_JALR,fprintf
1043 1:  jalr  $25
1044   nop
1045
1046   lw   $28,24($fp)
1047   b $L50
1048   nop
1049
1050 $L49:
1051   lw   $6,52($fp)
1052   lw   $2,%got($LC13)($28)
1053   addiu $5,$2,%lo($LC13)
1054   lw   $4,36($fp)
1055   lw   $2,%call16(fprintf)($28)
1056   move  $25,$2
1057   .reloc  1f,R_MIPS_JALR,fprintf
1058 1:  jalr  $25
1059   nop
1060
1061   lw   $28,24($fp)
1062   lw   $2,32($fp)
1063   addiu $2,$2,1
```

```
1064    sw   $2,32($fp)
1065  $L48:
1066    lw   $2,64($fp)
1067    lw   $3,32($fp)
1068    slt  $2,$3,$2
1069    bne  $2,$0,$L51
1070    nop
1071
1072  $L50:
1073    lw   $2,%got(stdout)($28)
1074    lw   $2,0($2)
1075    lw   $3,36($fp)
1076    beq  $3,$2,$L52
1077    nop
1078
1079    lw   $4,36($fp)
1080    lw   $2,%call16(fclose)($28)
1081    move  $25,$2
1082    .reloc  1f,R_MIPS_JALR,fclose
1083  1:  jalr  $25
1084    nop
1085
1086    lw   $28,24($fp)
1087  $L52:
1088    move  $2,$0
1089    move  $sp,$fp
1090    lw   $31,76($sp)
1091    lw   $fp,72($sp)
1092    addiu $sp,$sp,80
1093    jr   $31
1094    nop
1095
1096    .set  macro
1097    .set  reorder
1098    .end  main
1099    .size main, .-main
1100    .rdata
1101    .align  2
1102  $LC14:
1103    .ascii  "help\000"
1104    .align  2
1105  $LC15:
1106    .ascii  "version\000"
1107    .align  2
1108  $LC16:
1109    .ascii  "output\000"
1110    .align  2
1111  $LC17:
1112    .ascii  "divisor\000"
1113    .align  2
1114  $LC18:
1115    .ascii  "multiple\000"
1116    .section  .data.rel.local,"aw",@progbits
1117    .align  2
```

```
1118    .type argument_options.2692, @object
1119    .size argument_options.2692, 96
1120  argument_options.2692:
1121    .word $LC14
1122    .word 0
1123    .word 0
1124    .word 104
1125    .word $LC15
1126    .word 0
1127    .word 0
1128    .word 86
1129    .word $LC16
1130    .word 1
1131    .word 0
1132    .word 111
1133    .word $LC17
1134    .word 0
1135    .word 0
1136    .word 100
1137    .word $LC18
1138    .word 0
1139    .word 0
1140    .word 109
1141    .word 0
1142    .word 0
1143    .word 0
1144    .word 0
1145    .ident  "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```

B. Enunciado del trabajo práctico

66:20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI[1], escribiendo un programa portable que resuelva el problema descripto en la sección 5.

2. Alcance

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

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, usando la herramienta TEX/ LATEX.

4. Recursos

Usaremos el programa QEMU [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo Debian [2].

[1] Application Binary Interface

5. Programa

Se trata de una versión en lenguaje C de un programa que calcula el mínimo común múltiplo (mcm) y el máximo común divisor (mcd) entre dos números, utilizando el Algoritmo de Euclides [4] para el mcd. El programa recibirá por como argumentos dos números naturales M y N, y dará el resultado por `stdout` (o lo escribirá en un archivo). De haber errores, los mensajes de error deberán salir exclusivamente por `stderr`.

5.1. Comportamiento deseado

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ common -h
Usage:
  common -h
  common -V
  common [options] M N
Options:
  -h, --help     Prints usage information.
  -V, --version  Prints version information.
  -o, --output   Path to output file.
  -d  --divisor  Just the divisor
  -m  --multiple Just the multiple
Examples:
  common -o - 256 192
```

Ahora usaremos el programa para obtener el máximo común divisor y el mínimo común múltiplo entre 256 y 192. Usamos "-" como argumento de `-o` para indicarle al programa que imprima la salida por `stdout`:

```
$ common -d -o - 256 192
64
$ common -m -o - 256 192
768
$ common 256 192
64
768
```

El programa deberá retornar un error si sus argumentos están fuera del rango $[2, \text{MAXINT}]$.

6. Implementación

El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación.

6.1. API

Gran parte del programa estará implementada en lenguaje C. Sin embargo, las funciones `mcd(m,n)` y `mcm(m,n)` estarán implementadas en assembler MIPS32, para proveer soporte específico en nuestra plataforma principal de desarrollo, Debian/MIPS.

El propósito de `mcd(m,n)` es calcular el máximo común divisor de dos números naturales dados utilizando el algoritmo de Euclides [4].

```
unsigned int mcd(unsigned int m, unsigned int n);
```

El propósito de `mcm(m,n)` es calcular el mínimo común múltiplo de dos números naturales dados. Como $\text{mcm}(m,n) = \frac{m.n}{\text{mcd}(m,n)}$, la función deberá calcular este valor llamando a `mcd(m,n)` para calcular el mínimo común denominador entre m y n.

```
unsigned int mcm(unsigned int m, unsigned int n);
```

El programa en C deberá procesar los argumentos de entrada, llamar a una o a las dos funciones según las opciones, y escribir en `stdout` o un archivo el resultado. La función `mcd(m,n)` se puede implementar tanto de manera iterativa como de manera recursiva.

6.2. Portabilidad

Pese a contener fragmentos en assembler MIPS32, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad.

Para satisfacer esto, el programa deberá proveer dos versiones de `mcd` y `mcm`, incluyendo la versión MIPS32, pero también una versión C, pensada para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

6.3. ABI

El pasaje de parámetros entre el código C (`main()`, etc) y las rutinas `mcd(m,n)` y `mcm(m,n)`, en assembler, deberá hacerse usando la ABI explicada en clase: los argumentos correspondientes a los registros `$a0`-`$a3` serán almacenados por el *callee*, siempre, en los 16 bytes dedicados de la sección "function call argument area" [3].

6.4. Algoritmo

El algoritmo a implementar es el algoritmo de Euclides [4], explicado en clase.

7. Proceso de Compilación

En este trabajo, el desarrollo se hará parte en C y parte en lenguaje Assembler. Los programas escritos serán compilados o ensamblados según el caso, y posteriormente enlazados, utilizando las herramientas de GNU disponibles en el sistema Debian utilizado. Como resultado del enlace, se genera la aplicación ejecutable.

8. Informe

El informe deberá incluir:

- Este enunciado;
- Documentación relevante al diseño e implementación del programa, incluyendo un diagrama del stack;
- Corridas de prueba para los valores (5, 10), (256, 192), (1111, 1294), con los comentarios pertinentes;
- Diagramas del stack de las funciones, por ejemplo para los argumentos (256, 192);
- El código fuente completo, de los programas y del informe.

9. Fecha de entrega

La última fecha de entrega es el jueves 12 de Noviembre de 2020.

Referencias

[1] QEMU, `https://www.qemu.org/`

[2] Debian, the Universal Operating System, `https://www.debian.org/`.

[3] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

[4] Algoritmo de Euclides, `http://http://es.wikipedia.org/wiki/Algoritmo_de_Euclides`.

C. Código auxiliar

C.0.1. euclides algorithm.c

```c

#include "euclides_algorithm.h"

unsigned int mcd_euclides(unsigned int a, unsigned int b) {
  unsigned int r = 0;

  if (b == 0) {
    return a;
  }
  r = (a %b);
  return mcd_euclides(b, r);
}

unsigned int mcm_euclides(unsigned int a, unsigned int b) {
  unsigned int mcd = mcd_euclides(a, b);
  unsigned int a_x_b = (a * b);
  return (a_x_b / mcd);
}
```

Referencias

[1] *Artículo de Wikipedia [en] sobre el Algoritmo de Euclides* . https://en.wikipedia.org/wiki/Euclidean_*algorithm*

[2] *Parámetros de* `gcc`. https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html

[3] *Herramienta* `make`. https://www.gnu.org/software/make/manual/make.html