U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica

Organización de computadoras 66-20

TRABAJO PRÁCTICO 1

Conjunto de instrucciones MIPS

Curso: 2020 - 2do Cuatrimestre

Turno: Jueves

GRUPO N° 1	
Integrantes	**Padrón**
Gonzalo Almendro	80698
Cynthia Marlene Gamarra Silva	92702
Camila Serra	97422
Fecha de entrega:	12-11-2020
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

Índice **1**

1. Enunciado del trabajo práctico **2**

2. Objetivos **6**

3. Diseño e implementación **7**
3.1. Algoritmo de MCM . . . 7
3.2. Algoritmo de MCD . . . 7
3.3. Implementación de mcm.S . . . 8
3.4. Implementación de mcd.S . . . 9
3.5. Funciones de comunicación con el usuario . . . 10

4. Parámetros del programa **11**

5. Compilación del programa **11**

6. Pruebas realizadas **12**

7. Conclusiones **15**

Referencias **16**

A. Makefile **17**
A.1. Makefile . . . 17

B. Código fuente **18**
B.0.1. main.c . . . 18
B.0.2. mcd.c . . . 20
B.0.3. mcm.c . . . 20
B.0.4. command.c . . . 21
B.0.5. command.h . . . 24
B.0.6. constants.h . . . 25
B.0.7. file.c . . . 25
B.0.8. file.h . . . 26
B.0.9. mcm.S . . . 27
B.0.10. mcd.S . . . 30

1. 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 contenter 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`.

2. Objetivos

El presente trabajo tiene como objetivo calcular el mínimo común múltiplo(mcm) y el máximo común divisor entre dos números utilizando el Algortimo de Euclides para el mcd. Estos números serán provisto mediante algún stream (archivo o stdin). Para esto se realiza en código assembly MIPS32 dos rutinas: *mcd* y *mcm*. El resto del programa es dedicado a la interpretación de la entrada y correspondiente chequeo de errores el cual está escrito en C.

Además plantea como objetivos secundarios:

- Utilizar la ABI presentada por la cátedra para el desarrollo de los códigos en Assembly MIPS.
- Compilar dicho en un entorno que emula una arquitectura MIPS32 (QEMU).
- Verificar los casos mínimos de prueba solicitados por la cátedra.
- Realizar para verificar que el programa está funcionando de manera correcta.

3. Diseño e implementación

El programa propuesto en el enunciado del trabajo práctico consta del cálculo del mínimo común múltiplo y el máximo común divisor entre dos números ingresados como texto por entrada estándar (stdin). El resultado de la operación se muestra por salida estándar (stdout).

La implementación del sistema se realizó sobre el lenguaje de programación C, como fue determinado por el enunciado del trabajo práctico. El uso de un lenguaje de tipeo estático significó una ventaja por la oportunidad de usar chequeo de errores en tiempo de compilación, además de la conocida eficiencia propia del lenguaje C.

La aplicación resultante es un archivo ejecutable, el cual puede ser corrido desde consola con el nombre completo del fichero, además cuenta con la posibilidad de determinar por consola los parámetros de ejecución que se mencionarán en la seccion de "Parámetros del programa".

3.1. Algoritmo de MCM

Se implementó una versión de este algoritmo de forma preliminar en C, para que la transcripción a MIPS sea algo más fácil.

```
1  extern unsigned int mcd(unsigned int m, unsigned int n);
2
3  /*
4    Mínimo Comun Multiplo
5  */
6  #ifdef MIPS
7  extern unsigned int mcm(unsigned int m, unsigned int n);
8  #else
9  unsigned int mcm(unsigned int m, unsigned int n)
10 {
11     unsigned int mcm_v = 0;
12     unsigned int mcd_v = 0;
13
14     unsigned int a = m;
15     unsigned int b = n;
16     if( n > m ) {
17         a = n;
18         b = m;
19     }
20
21     mcd_v = mcd(a, b);
22     mcm_v = ( a / mcd_v ) * b;
23
24     return mcm_v;
25 }
26 #endif
```

En primer lugar el algoritmo calcula el máximo común divisor (utilizando la otra función realizada) entre los dos números (el menor divide al mayor). Luego, con este resultado, se divide el mayor por lo obtenido, y se multiplica el menor.

3.2. Algoritmo de MCD

Para este algoritmo, también se implementó una versión de forma preliminar en C, para que la transcripción a MIPS sea algo más fácil.

```
1  /*
2    Máximo Comun Divisor
3  */
4  #ifdef MIPS
5  extern unsigned int mcd(unsigned int m, unsigned int n);
```

```
#else
unsigned int mcd(unsigned int m, unsigned int n)
{
    if (m == 0) return m;
    if (n == 0) return n;
    if (m == n) return m;

    unsigned int a = m;
    unsigned int b = n;
    unsigned int mcd_v = 0;
    if( n > m )
    {
        a = n;
        b = m;
    }

    do
    {
        mcd_v = b;
        b = a % b;
        a = mcd_v;
    } while( b != 0 );

    return mcd_v;
}
#endif
```

Para el cálculo del máximo común divisor, se realizó el algoritmo iterativo. Lo que hace es dividir el mayor número (a) por el menor (b), y luego repetir con el mayor número igual al menor (b es ahora a), y el menor igual al módulo de la división (b igual a a %b). Esto se repite mientras el módulo de la división sea distinto de cero.

3.3. Implementación de mcm.S

El código implementado en C nos sirve de guía para implementar la versión del mínimo común múltiplo en assembly MIPS. Cabe destacar que no es una transcripción directa, sino código C adaptado a MIPS.

El código fuente se puede ver en el anexo: mcm.S

El stack frame que crea la función se muestra en el cuadro .

	Argument-building area Caller (ABA caller)
52	n
48	m
	Saved-registers area (SRA)
44	–PADDING–
40	RA
36	GP
32	FP
	Local and Temporary Area (LTA)
28	b
24	a
20	mcd_v
16	mcm_v
	Argument-building area (ABA)
12	a3
08	a2
04	a1
00	a0

Cuadro 1: Diseño del stack frame de `mcm`. La primer columna se corresponde con el offset en bytes respecto del frame pointer.

El tamaño del stack queda definido por la directiva:

```
1 #define SIZE_SF PADDING + 4
```

con *PADDING* igual a 44

3.4. Implementación de mcd.S

Lo mismo realizado para el mcm fue realizado para implementar el máximo común divisor.

El código fuente se puede ver en el anexo: mcm.S

El stack frame que crea la función se muestra en el cuadro .

	Argument-building area Caller (ABA caller)
28	n
24	m
	Saved-registers area (SRA)
20	GP
16	FP
	Local and Temporary Area (LTA)
12	–PADDING_0–
08	mcd_v
04	b
00	a

Cuadro 2: Diseño del stack frame de `mcd`. La primer columna se corresponde con el offset en bytes respecto del frame pointer.

El tamaño del stack queda definido por la directiva:

```
1 #define SIZE_SF 24
```

3.5. Funciones de comunicación con el usuario

- `show_help()`: Imprime por salida estándar los distintos comandos posibles. Desglosa el menú que dispondrá el usuario para usar el programa, así como la sintaxis correcta.
- `set_error()`: Setea al TDA command_options_st en error, para que el programa no siga.
- `has_error() / show_error()`: En caso de error, la primere función indicará que hubo error y la segunda imprimirá el mismo y nuevamente el menú de ayuda, con ejemplos de ayuda para la sintáxis.
- `show_version()`: Imprime por salida estándar la versión del programa. La versión del programa es un número que se le asigna para mencionar su nivel de desarrollo y su actualización. El primer número es el mayor, que determina cambios grandes o saltos cualitativos en el desarrollo, mientras que el segundo es el menor y varía con las alteraciones o correcciones más pequeñas.

4. Parámetros del programa

Los parámetros del programa serán varios:

- *-h:* Con el cuál se mostrará el menú de ayuda detallado, de que otros parámetros hay y como invocarlos apropiadamente.
- *-V:* Permite verificar que versión del programa es actualmente, para determinar si se han hecho cambios grandes o pequeños.
- *-o:* Especifica la salida del stream/archivo. Con "-"será stdout.
- *-d:* Sólo calcular el mcd.
- *-m:* Sólo calcular el mcm.

5. Compilación del programa

Debido al requerimiento de utilizar el programa en una computadora con arquitectura MIPS32, se utiliza el emulador `QEMU`, previamente configurado como se explicó en clase.

La compilación del programa se realizó de la siguiente manera:

- 1°: Se inició el emulador de `QEMU`, ejecutando un script proporcionado por la cátedra `./start_quemu.sh`.
- 2°: Una vez iniciado el emulador, debemos configurar los `"Túneles"` con SSH.
- 3°: Una vez configurado los `"Túneles"`, con otra terminal desde el `$Host` copiamos los archivos a compilar en el entorno de QEMU con el comando `scp -P 5555 -r ruta/de/origen root@localhost:/ruta/de/destino`.
- 4°: Para compilar en el entorno de QEMU, ejecutamos dicha compilación usando `make`. Se dispone de un archivo `Makefile` con la definición de este target. Alternativamente, es posible compilar el trabajo práctico usando directamente `gcc`:

```
gcc -o0 -g -Wall -Werror -pedantic -std=c99 command.h command.c file.h file.c mcm.S mcd.S
main.c -o tp1
```

De esta forma se creará el ejecutable tp1 en el directorio donde se compiló.

6. Pruebas realizadas

Realizamos un set de pruebas automatizadas con los siguientes casos, los cuales corren todos satisfactoriamente. Para correr los tests automáticos se debe estar parado en la carpeta test del proyecto teniendo el archivo ejecutable tp1 en el mismo. Para ejecutarlo:

```
$ ./tests-automatic.sh
```

El código del mismo es el siguiente:

```
#!/bin/bash

echo "############################################################"
echo "#################### Tests Automaticos #####################"
echo "############################################################"
echo ""

echo "#-----------------------------------------------------------"
echo "#----------# COMIENZA Test 01 - Help #-----------"

./common -h > ./test-out/test01-stdout.txt

if diff -b ./test-out/test01-stdout.txt ./expected/test01-stdout.txt ; then
        echo "[OK]";
else
        echo "[ERROR]";
fi

echo "#----------# TERMINA  Test 01 - Help #-----------"
echo "#-----------------------------------------------------------"
echo ""

echo "#----------# COMIENZA Test 02 - Version   #-----------"

./common -V > ./test-out/test02-stdout.txt

if diff -b ./test-out/test02-stdout.txt ./expected/test02-stdout.txt ; then
        echo "[OK]";
else
        echo "[ERROR]";
fi

echo "#----------# TERMINA  Test 02 - Version  #-----------"
echo "#-----------------------------------------------------------"
echo ""

echo "#----------# COMIENZA Test 03 - mcm stdout #-----------"

./common -m 256 192 > ./test-out/test03-stdout.txt

if diff -b ./test-out/test03-stdout.txt ./expected/test03-stdout.txt ; then
        echo "[OK]";
else
        echo "[ERROR]";
fi

echo "#----------# TERMINA  Test 03 - mcm stdout #-----------"
echo "#-----------------------------------------------------------"
echo ""

echo "#----------# COMIENZA Test 04 - mcm -o #-----------"

./common -m 256 192 -o ./test-out/test04-stdout.txt

if diff -b ./test-out/test04-stdout.txt ./expected/test04-stdout.txt ; then
        echo "[OK]";
else
```

```
58          echo "[ERROR]";
59  fi
60
61  echo "#----------# TERMINA  Test 04 - mcm -o #-----------"
62  echo "#-------------------------------------------------------------"
63  echo ""
64
65
66  echo "#----------# COMIENZA Test 05 - mcd stdout #-----------"
67
68  ./common -d 256 192 > ./test-out/test05-stdout.txt
69
70  if diff -b ./test-out/test05-stdout.txt ./expected/test05-stdout.txt ; then
71          echo "[OK]";
72  else
73          echo "[ERROR]";
74  fi
75
76  echo "#----------# TERMINA  Test 05 - mcd stdout #-----------"
77  echo "#-------------------------------------------------------------"
78  echo ""
79
80  echo "#----------# COMIENZA Test 06 - mcd -o #-----------"
81
82  ./common -d 256 192 -o ./test-out/test06-stdout.txt
83
84  if diff -b ./test-out/test06-stdout.txt ./expected/test06-stdout.txt ; then
85          echo "[OK]";
86  else
87          echo "[ERROR]";
88  fi
89
90  echo "#----------# TERMINA  Test 06 - mcd -o #-----------"
91  echo "#-------------------------------------------------------------"
92  echo ""
93
94
95
96  echo "#----------# COMIENZA Test 07 - both stdout #-----------"
97
98  ./common 256 192 > ./test-out/test07-stdout.txt
99
100 if diff -b ./test-out/test07-stdout.txt ./expected/test07-stdout.txt ; then
101         echo "[OK]";
102 else
103         echo "[ERROR]";
104 fi
105
106 echo "#----------# TERMINA  Test 07 - both stdout #-----------"
107 echo "#-------------------------------------------------------------"
108 echo ""
109
110 echo "#----------# COMIENZA Test 08 - both -o #-----------"
111
112 ./common 256 192 -o ./test-out/test08-stdout.txt
113
114 if diff -b ./test-out/test08-stdout.txt ./expected/test08-stdout.txt ; then
115         echo "[OK]";
116 else
117         echo "[ERROR]";
118 fi
119
120 echo "#----------# TERMINA  Test 08 - both -o #-----------"
121 echo "#-------------------------------------------------------------"
122 echo ""
```

El resultado del mismo en la consola del QEMU es el siguiente:

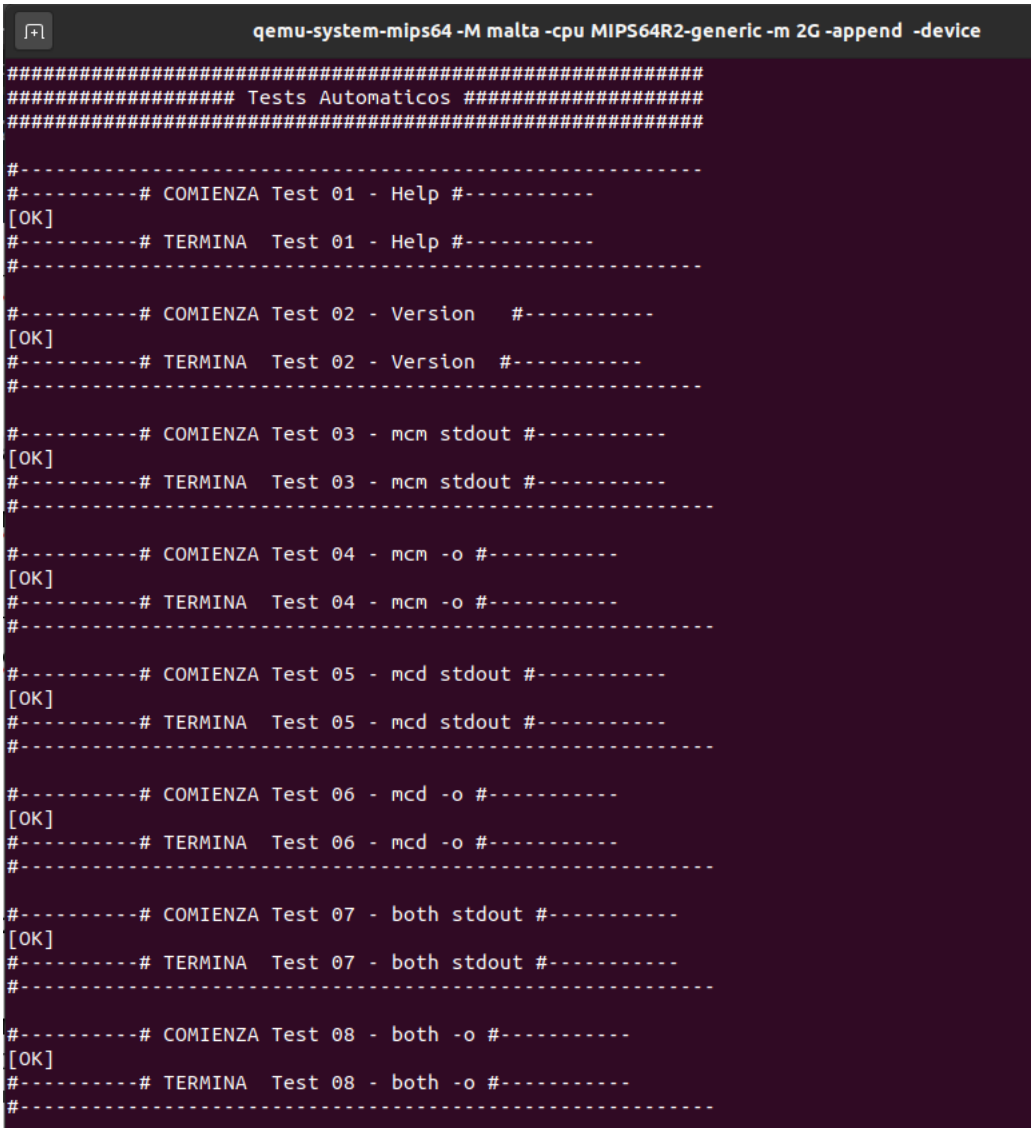

Figura 1: Captura de pantalla de ejecuciones

7. Conclusiones

Se implementaron en lenguaje assembly MIPS32 las funciones que permitieron calcular el mcm y el mcd de dos números.

Pudimos adentrarnos más en la arquitectura MIPS32, en el consecuente análisis y desarrollo de código assembler MIPS utilizando el emulador "QEMU", entendiendo un poco mejor porqué son necesarias las recetas para poder debuggear con GDB.

Tuvimos un par de dificultades al tratar de llamar de el mcm.S al mcd.S, pero investigando y comparando con los ejercicios resueltos provistos, pudimos sortearlo y hacer que funcione correctamente el programa.

Se describió el flujo del programa y las funciones y estructuras más importantes del mismo, junto con el manejo de errores. Además, se describió como transferir archivos hacia el entorno de QEMU, plataforma que se utilizó para compilar el programa y el modo de compilación del mismo utilizando las herramientas gcc y gdb.

Referencias

[1] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.

[2] *GNU Make* - https://www.gnu.org/software/make

[3] *GNU Gcc* - https://gcc.gnu.org

[4] *GNU gdb* - https://www.gnu.org/software/gdb

[5] *Valgrind* - http://valgrind.org

[6] *LATEX* - latex-project.org

[7] *Sublime Text* - https://www.sublimetext.com

[8] *Kernighan, B. W. - Ritchie, D. M. - C Programming Language - 2 nd edition - Prentice*

[9] *Patton, R. - Software Testing - 2 nd edition - Sams Indianapolis, IN, USA 2005*

[10] *Apuntes del curso 66.20 Organización de Computadoras - Cátedra Hamkalo - Facultad de Ingeniería de la Universidad de Buenos Aires*

[11] *System V Application Binary Interface - MIPS/RISC Processor Supplement - 3 nd edition*

[12] *Algoritmo de Euclides* - http://http://es.wikipedia.org/wiki/ Algoritmo_de_Euclides

[13] *QEMU* - https://www.qemu.org/

[14] *Controlling the kernel unalignment handling via debugfs* - https://www.linux-mips.org/wiki/Alignment

A. Makefile

A.1. Makefile

```
CC = gcc
CFLAGS = -o0 -g -Wall -Werror -pedantic -std=c99

EXEC = common

# TP sin mips
tp: SOURCES = *.c
tp: $(EXEC)

# Código mips
tp.mips: CFLAGS += -DMIPS
tp.mips: SOURCES = mcd.S mcm.S *.c
tp.mips: $(EXEC)

# TP con mips
all: CFLAGS += -DMIPS
all: SOURCES = *.c
all: SOURCES += mcd.S mcm.S
all: $(EXEC)

command.o: command.c command.h
	$(CC) $(CFLAGS) -c command.c -o command.o

file.o: file.c file.h
	$(CC) $(CFLAGS) -c file.c -o file.o

$(EXEC): $(SOURCES)
	$(CC) $(CFLAGS) $(SOURCES) -o $(EXEC) -lm


run: $(EXEC)
	./$(EXEC)


clean:
	rm -f *.o
	rm -f $(EXEC)
```

B. Código fuente

B.0.1. main.c

```c
#include <stddef.h>
#include <getopt.h>
#include "constants.h"
#include "command.h"

int main(int argc, char **argv) {
    struct option arg_long[] = {
            {"output",   required_argument, NULL, 'o'},
            {"divisor",  no_argument, NULL, 'd'},
            {"multiple", no_argument, NULL, 'm'},
            {"help",     no_argument, NULL, 'h'},
            {"version",  no_argument, NULL, 'V'},
    };

    char arg_opt_str[] = "dmo:hV";
    int arg_opt;
    int arg_opt_idx = 0;

    command_options_st cmd_options;
    command_create(&cmd_options);

    char should_process = TRUE;
    while ((arg_opt = getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1
           &&
           should_process) {
        switch (arg_opt) {
            case 'h': {
                show_help();
                should_process = FALSE;
                }
                break;
            case 'V': {
                show_version();
                should_process = FALSE;
                }
                break;
            case 'o': {
                set_output_file(&cmd_options, optarg);
                }
                break;
            case 'd': {
                // Setea solo calculo de MCD
                set_divisor_only(&cmd_options);
                }
                break;
            case 'm': {
                // Setea solo calculo de MCM
                set_multiple_only(&cmd_options);
                }
                break;
            default:
            {
                set_error(&cmd_options, INVALID_ARGUMENT);
            }
            break;
        }
    }

    set_numbers(&cmd_options, argc, argv);

    if (!should_process) {
        return 0;
    }

```

```
65      if (has_errors(&cmd_options)) {
66          show_error(&cmd_options);
67          return 1;
68      }
69
70      return process(&cmd_options);
71  }
```

B.0.2. mcd.c

```c
/*
  Máximo Comun Divisor
*/
#ifdef MIPS
extern unsigned int mcd(unsigned int m, unsigned int n);
#else
unsigned int mcd(unsigned int m, unsigned int n)
{
    if (m == 0) return m;
    if (n == 0) return n;
    if (m == n) return m;

    unsigned int a = m;
    unsigned int b = n;
    unsigned int mcd_v = 0;
    if( n > m )
    {
        a = n;
        b = m;
    }

    do
    {
        mcd_v = b;
        b = a % b;
        a = mcd_v;
    } while( b != 0 );

    return mcd_v;
}
#endif
```

B.0.3. mcm.c

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

/*
  Mínimo Comun Multiplo
*/
#ifdef MIPS
extern unsigned int mcm(unsigned int m, unsigned int n);
#else
unsigned int mcm(unsigned int m, unsigned int n)
{
    unsigned int mcm_v = 0;
    unsigned int mcd_v = 0;

    unsigned int a = m;
    unsigned int b = n;
    if( n > m ) {
        a = n;
        b = m;
    }

    mcd_v = mcd(a, b);
    mcm_v = ( a / mcd_v ) * b;

    return mcm_v;
}
#endif
```

B.0.4. command.c

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include "command.h"
#include "limits.h"

extern unsigned int mcm(unsigned int m, unsigned int n);
extern unsigned int mcd(unsigned int m, unsigned int n);

void command_create(command_options_st *opt) {
    create_file(&opt->output_file);
    opt->error_condition = OK;
    opt->output_path = 0;
    opt->multiple = TRUE;
    opt->divisor = TRUE;
    opt->m = MIN_NUMBER;
    opt->n = MIN_NUMBER;
}

void set_output_file(command_options_st *opt, const char *output) {
    if ( strcmp(output,"-") != 0 ) {
        opt->output_path = output;
    }
}

void set_divisor_only(command_options_st *opt) {
    opt->multiple = FALSE;
}

void set_multiple_only(command_options_st *opt) {
    opt->divisor = FALSE;
}

void set_numbers(command_options_st *opt, int argc, char** argv) {

    if( argc >= 3 ) {
        char* arg_number_m = argv[argc-2];
        char* arg_number_n = argv[argc-1];

        if (strcmp(arg_number_m,"") == 0) arg_number_m = "0";
        if (strcmp(arg_number_n,"") == 0) arg_number_n = "0";
        unsigned int m = strtoul(arg_number_m, NULL, 10);
        unsigned int n = strtoul(arg_number_n, NULL, 10);

        if( m < MIN_NUMBER || m > INT_MAX )
        {
                set_error(opt, INVALID_NUMBERS);
        }
        else
        {
                opt->m = m;
        }

        if( n < MIN_NUMBER || n > INT_MAX )
        {
                set_error(opt, INVALID_NUMBERS);
        }
        else
        {
                opt->n = n;
        }
    }
    else
    {
        set_error(opt, INVALID_ARGUMENT);
```

```
67      }
68  }
69
70  void set_error(command_options_st *opt, char error_condition) {
71      opt->error_condition = error_condition;
72  }
73
74  int has_errors(command_options_st *opt) {
75      return (opt->error_condition != OK) ? ERROR : OK;
76  }
77
78  void show_error(command_options_st *opt) {
79      char *error_message = NULL;
80      bool should_show_help = false;
81      if (opt->error_condition == INVALID_ARGUMENT) {
82          error_message = "Argumentos Invalidos!\n\n";
83          should_show_help = true;
84      } else if (opt->error_condition == NO_ARGUMENTS) {
85          error_message = "No se recibieron Argumentos!\n\n";
86          should_show_help = true;
87      } else if (opt->error_condition == INVALID_NUMBERS) {
88          error_message = "Los numeros deben ser mayores a 2 y menores a INT_MAX!\n\n";
89         should_show_help = true;
90      }
91
92      fprintf(stderr, "%s", error_message);
93      if (should_show_help) {
94          show_help();
95      }
96  }
97
98  void show_help() {
99      printf("Options:\n");
100     printf("\t-h,\t--help\t\tPrint usage information.\n");
101     printf("\t-V,\t--version\tPrint version information.\n");
102     printf("\t-o,\t--output\tPath to output file.\n");
103     printf("\t-d,\t--divisor\tJust the divisor.\n");
104     printf("\t-m,\t--multiple\tJust the multiple.\n");
105     printf("Examples:\n");
106     printf("\tcommon -o - 256 192\n");
107 }
108
109 void show_version() {
110     printf("Version: 1.0\n");
111 }
112
113
114 char process(command_options_st *opt) {
115     if (open_file_write(&opt->output_file, opt->output_path) == ERROR) {
116         return ERROR;
117     }
118
119     unsigned int result;
120     if (opt->multiple) {
121         result = mcm(opt->m, opt->n);
122         if( result > INT_MAX ) {
123                 char* overflow_msg = "overflow";
124                 file_write_text(&opt->output_file,overflow_msg);
125         } else {
126                 file_write_number(&opt->output_file, result);
127         }
128     }
129
130     if (opt->divisor) {
131         result = mcd(opt->m, opt->n);
132         if( result > INT_MAX ) {
133                 char* overflow_msg = "overflow";
```

```
134                 file_write_text(&opt->output_file,overflow_msg);
135         } else {
136                 file_write_number(&opt->output_file, result);
137         }
138     }
139
140     close_file(&opt->output_file);
141     return OK;
142 }
```

B.0.5. command.h

```
#ifndef __COMMAND_H__
#define __COMMAND_H__

#include "file.h"
#include "constants.h"

typedef struct {
    File output_file;
    const char *output_path;
    char error_condition;
    char multiple;
    char divisor;
    unsigned int m;
    unsigned int n;
} command_options_st;

void command_create(command_options_st *opt);

void set_output_file(command_options_st *opt, const char *output);

void set_divisor_only(command_options_st *opt);

void set_multiple_only(command_options_st *opt);

void set_numbers(command_options_st *opt, int argc, char **argv);

void set_error(command_options_st *opt, char error_condition);

int has_errors(command_options_st *opt);

void show_error(command_options_st *opt);

void show_help();

void show_version();

char process(command_options_st *opt);

#endif
```

B.0.6. constants.h

```c
#ifndef __TP1_CONSTANTS_H__
#define __TP1_CONSTANTS_H__

// BOOLEAN VALUES
#define TRUE 1
#define FALSE 0

// RESULT VALUES
#define OK 0
#define ERROR 1

// ERROR CODES
#define INVALID_ARGUMENT -1
#define NO_ARGUMENTS -2
#define INVALID_NUMBERS -3

#define MIN_NUMBER 2

#endif
```

B.0.7. file.c

```c
#include "file.h"
#include <stdlib.h>
#include <string.h>
#include <errno.h>

void create_file(File *file) {
    file->file = 0;
    file->eof = 0;
}

char open_file_write(File *file, const char *route) {
    if (route == NULL) {
        file->file = stdout;
    } else {
        file->file = fopen(route, "wb");
        if (file->file == NULL) {
            int err = errno;
            fprintf(stderr, "Error al abrir archivo: %s\n", strerror(err));
            return ERROR;
        }
    }
    return OK;
}


int close_file(File *file) {
    if (file->file == stdin || file->file == stdout) return OK;

    int result = fclose(file->file);
    if (result == EOF) {
        int err = errno;
        fprintf(stderr, "Error al cerrar archivo: %s\n", strerror(err));
        return ERROR;
    }
    return OK;
}

void file_write_number(File *file, unsigned int number) {
    fprintf(file->file, "%d\n", number);
}

void file_write_text(File *file, const char* text) {
    fprintf(file->file, "%s\n", text);
}
```

```
45
46 int file_eof(File *file) {
47     return file->eof;
48 }
```

B.0.8. file.h

```
1  #ifndef __FILE_H__
2  #define __FILE_H__
3
4  #include <stdio.h>
5  #include "constants.h"
6
7  typedef struct {
8      FILE *file;
9      char eof;
10 } File;
11
12 void create_file(File *file);
13
14 char open_file_write(File *file, const char *route);
15
16 int close_file(File *file);
17
18 void file_write_number(File *file, unsigned int number);
19
20 void file_write_text(File *file, const char *text);
21
22 int file_eof(File *file);
23
24 #endif
```

B.0.9. mcm.S

```
#include <sys/regdef.h>

.text
.abicalls
.align  2
.globl  mcm
.ent  mcm

# Pipeline magic.
.set        noreorder
.cpload     t9
.set        reorder

/* STACK DESIGN - SIZE: 48
      ==========
  52  |   a1   |
      ---------- ABA CALLER (8)
  48  |   a0   |
      ==========
  44  | PADDING|
      ----------
  40  |   ra   |
      ---------- SRA (16)
  36  |   gp   |
      ----------
  32  |   fp   |
      ==========
  28  |   b    |
      ----------
  24  |   a    |
      ---------- LTA(16)
  20  | mcd_v  |
      ----------
  16  |  mcm_v |
      ==========
  12  |   a3   |
      ----------
  8   |   a2   |
      ---------- ABA CALLEE(16)
  4   |   a1   |
      ----------
  0   |   a0   |
      ==========
*/


#  Argument-building area callee (ABA callee).
#define A0_CALLEE 0
#define A1_CALLEE 4
#define A2_CALLEE 8
#define A3_CALLEE 12

# Local and Temporary Area (LTA).
#define MCM_V 16
#define MCD_V 20
#define A_LTA 24
#define B_LTA 28

# Saved-registers area (SRA).
#define FP 32
#define GP 36
#define RA 40
#define PADDING 44

#  Argument-building area caller (ABA caller).
#define M_A0 48
```

```
67 #define N_A1 52
68
69
70 #define SIZE_SF PADDING + 4
71
72
73 # Debugger metadata.
74   .frame  fp,SIZE_SF,ra
75
76 // mcm(unsigned int m, unsigned int n)
77 mcm:
78         # Allocate memory for the stack.
79         subu  sp,sp,SIZE_SF
80
81         # SRA beginning area.
82         .cprestore GP
83
84         # Save the callee-saved registers used by the caller in the SRA.
85         sw  fp,FP(sp)
86         sw  gp,GP(sp)
87         sw  ra,RA(sp)
88         # We must set the $fp to the beginning of the stack.
89         move fp,sp
90
91         # Now we save the arguments that were loaded by the caller
92         # in the area reserved by the caller.
93         sw  a0,M_A0(fp)
94         sw  a1,N_A1(fp)
95
96
97         li       t0, 0
98         sw      t0,MCM_V(fp)
99         li       t1, 0
100        sw      t1,MCD_V(fp)
101        lw      t0,M_A0(fp)
102        sw      t0,A_LTA(fp)    # a = m
103        lw      t1,N_A1(fp)
104        sw      t1,B_LTA(fp)   # b = n
105        lw      t3,N_A1(fp)
106        lw      t4,M_A0(fp)
107        sltu    t4,t4,t3       # m < n
108        beqz    t4,m_high_n
109
110        lw      t2,N_A1(fp)
111        sw      t2,A_LTA(fp)   # a = n
112        lw      t3,M_A0(fp)
113        sw      t3,B_LTA(fp)   # b = m
114 m_high_n:
115        lw      a1,B_LTA(fp)
116        lw      a0,A_LTA(fp)
117        jal     mcd
118
119        move    t0, v0         # t0 = mcd(a, b)
120        sw      t0,MCD_V(fp)  # mcd_v = mcd(a, b)
121        lw      t1,A_LTA(fp)  # t1 = a
122        lw      t2,MCD_V(fp)
123        divu    t0,t1,t2       # div: a / mcd_v
124        mfhi    t5
125        mflo    t3
126        lw      t4,B_LTA(fp)
127        mul     t4,t3,t4       # ( a / mcd_v ) * b
128        sw      t4,MCM_V(fp)  # mcm_v = ( a / mcd_v ) * b
129        move    v0,t4          # Return mcm_v value
130
131 stack_unwinding:
132
133        lw        ra, RA(sp)
```

```
134         lw          fp, FP(sp)
135         lw          gp, GP(sp)
136         addu        sp, sp, SIZE_SF
137         jr          ra
138
139 .end         mcm
```

B.0.10. mcd.S

```
#include <sys/regdef.h>

.text
.align  2

.globl  mcd
.ent  mcd


/* STACK DESIGN - SIZE: 24
      ===========
  28  |   a1    |
      ---------- ABA CALLER (8)
  24  |   a0    |
      ===========
  20  |   gp    |
      ----------  SRA(8)
  16  |   fp    |
      ===========
  12  |PADDING_0|
      ----------
  8   |  mcd_v  |
      ---------- LTA(16)
  4   |   b     |
      ----------
  0   |   a     |
      ===========
*/


# Local and Temporary Area (LTA).
#define TEMP_0 0
#define TEMP_1 4
#define TEMP_2 8
#define PADDING_0 12

# Saved-registers area (SRA).
#define FP 16
#define GP 20

#  Argument-building area caller (ABA caller).
#define M_A0 24
#define N_A1 28


#define SIZE_SF 24

// mcd(unsigned int m, unsigned int n)
mcd:
        # Allocate memory for the stack.
        subu  sp,sp,SIZE_SF


        # Save the callee-saved registers used by the caller in the SRA.
        sw  fp,FP(sp)
        sw  gp,GP(sp)
        # We must set the $fp to the beginning of the stack.
        move fp,sp

        # Now we save the arguments that were loaded by the caller
        # in the area reserved by the caller.
        sw  a0,M_A0(fp)
        sw  a1,N_A1(fp)


        # case if (m == 0)
```

```
67          lw      t0,M_A0(fp)
68          bnez    t0,case_n_0
69
70          # return m;
71          lw      v0,M_A0(fp)
72          b       stack_unwinding
73
74  case_n_0:
75          # case  if (n == 0)
76          lw      t0,N_A1(fp)
77          bnez    t0,case_equal
78
79          # return n;
80          lw      v0,N_A1(fp)
81          b       stack_unwinding
82
83  case_equal:
84          # case  if (m == n)
85          lw      t0,M_A0(fp)
86          lw      t1,N_A1(fp)
87          bne     t1,t0,next_line
88
89          # return m;
90          lw      v0,M_A0(fp)
91          b       stack_unwinding
92
93  next_line:
94          lw      t0,M_A0(fp)    # t0 = m
95          sw      t0,TEMP_0(fp)  # a  = m
96          lw      t1,N_A1(fp)    # t1 = n
97          sw      t1,TEMP_1(fp)  # b = n
98          lw      t0,M_A0(fp)
99          lw      t1,N_A1(fp)
100         sltu    t0,t0,t1   # m < n
101         beqz    t0,cont_do
102
103
104         lw      t2,N_A1(fp)
105         sw      t2,TEMP_0(fp)   # a = n
106         lw      t3,M_A0(fp)
107         sw      t3,TEMP_1(fp)   # b = m
108 cont_do:
109         lw      t0,TEMP_1(fp)
110         sw      t0,TEMP_2(fp)   # mcd_v = b
111         lw      t1,TEMP_0(fp)   # t1 = a
112         lw      t2,TEMP_1(fp)   # t2 = b
113         divu    t0,t1,t2
114         mfhi    t3
115         sw      t3,TEMP_1(fp)  # b = a % b
116         lw      t0,TEMP_2(fp)
117         sw      t0,TEMP_0(fp)  # a = mcd_v
118         lw      t4,TEMP_1(fp)
119         bnez    t4,cont_do     # continue loop with b != 0
120
121         lw      t0,TEMP_2(fp)
122         move     v0,t0
123 stack_unwinding:
124
125         lw        fp, FP(sp)
126         lw        gp, GP(sp)
127         addu      sp, sp, SIZE_SF
128         jr        ra
129
130 .end        mcd
```