.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo Práctico 1

[66.20/86.37] Organización de Computadoras
Curso 2
Segundo cuatrimestre de 2020

Alumnos	Padrón	Correo electrónico	Slack
Gómez, Joaquín	103735	joagomez@fi.uba.ar	Joaquín Gomez
Grassano, Bruno	103855	bgrassano@fi.uba.ar	Bruno Grassano
Romero, Adrián	103371	adromero@fi.uba.ar	Adrián Romero

Índice

1. Introducción

El propósito del trabajo es familiarizarse con el conjunto de instrucciones MIPS32 y con el concepto de ABI.

Para ello buscamos escribir un programa en C que calcule el mínimo común múltiplo (MCM) y máximo común divisor (MCD) entre dos números utilizando el algoritmo de Euclides.

También se incluirán las implementaciones de las funciones MCM y MCD en assembly MIPS32.

2. Diseño e implementación

Nuestra propuesta al problema planteado es crear una función que calcule el MCD utilizando el algoritmo de Euclides y luego obtener el MCM con la relación:

$$MCM(a,b) = \frac{a \cdot b}{MCD(a,b)}$$

Procedemos a explicar el algoritmo de Euclides con un ejemplo.

2.1. El algoritmo de Euclides

Supongamos que queremos buscar el MCD entre 192 y 78 mediante el algoritmo de Euclides.

Empezamos por calcular:

- 192 mod 78 = 36

Dado que el resultado de 192 mod 78 no fue 0 sabemos que 192 no es divisible por 78 y por lo tanto debemos continuar buscando el MCD(192,78).

Para encontrar MCD(192,78) podemos buscar MCD(78,36), pues por propiedad de MCD se cumple que MCD(192,78) = MCD(78, 192 mod 78) = MCD(78,36).

Calculamos entonces:

- 78 mod 36 = 6

Dado que el resultado de 78 mod 36 no fue 0 procedemos a buscar MCD(36,6) pues MCD(78,36) = MCD(36,6)

Calculamos:

- 36 mod 6 = 0

Como el resultado de 36 mod 6 = 0 podemos decir que el MCD(36,6) = 6

Además tenemos que: MCD(36,6) = MCD(78,36) = MCD(192,78) = 6 que es lo que buscábamos.

2.2. Implementación del MCD en C

Nuestra implementación propuesta en C para calcular el MCD es la siguiente:

```
unsigned int mcd(unsigned int m, unsigned int n){
    unsigned int maximoEncontrado;
    while(n!=0){
        maximoEncontrado = n;
        n = m % n;
        m = maximoEncontrado;
    }
    return m;
}
```

Hacemos notar que decidimos realizar una implementación iterativa del algoritmo.

Buscamos calcular mcd(m, n). Para ilustrar el comportamiento de nuestra implementación utilizaremos el mismo ejemplo que en la sección anterior, es decir m = 192 y n = 78.

Buscamos calcular entonces mcd(m = 192, n = 78).

En una primera iteración las variables tomaran los siguientes valores:

- maximoEncontrado = n = 78
- n = m mod n = 192 mod 78 = 36
- m = maximoEncontrado = 78

Ahora verificamos la condición de corte, dado que n es distinto de 0 sabemos que 192 no es divisible por 78. Luego debemos proceder a calcular mcd(78, 36) y justamente las variables m y n tienen estos valores: m = 78 y n = 36

En una segunda iteración las variables tomaran los siguiente valores:

- maximoEncontrado = n = 36
- n = m mod n = 78 mod 36 = 6
- m = maximoEncontrado = 36

Nuevamente como el valor de n es distinto de 0 tenemos que calcular mcd(m = 36, n = 6) para la siguiente iteración:

- maximoEncontrado = n = 6
- n = m mod n = 36 mod 6 = 0
- m = maximoEncontrado = 6

Finalmente como n = 0, estaríamos intentando calcular MCD(m = 6, n = 0) y sabemos que por propiedad MCD(m = 6, n = 0) = 6 y por lo tanto en la función devolvemos el valor final de m.

2.3. Implementación del MCM en C

La implementación en C que realizamos para obtener el MCM es la siguiente:

```
unsigned int mcm(unsigned int m, unsigned int n){
    unsigned long producto = (unsigned long)m*(unsigned long)n;
    if(producto > UINT_MAX){
        return 0;
    }
    return (m*n)/mcd(m,n);
}
```

Algo a destacar de la misma es que se verifica si ocurre un overflow con respecto al limite del int (unsigned) al realizar la multiplicación de los dos números. Si esto ocurre se devuelve un 0 para indicar que ocurrió un error.

Después lo que queda es la relación que se tiene entre el MCM y el MCD. A través de ella se obtiene lo pedido.

2.4. Detalles de la linea de comandos

Para ejecutar el programa se tienen diferentes opciones validas. Las mismas se pueden ver pidiendo la ayuda del programa. Se muestran a continuación algunas opciones.

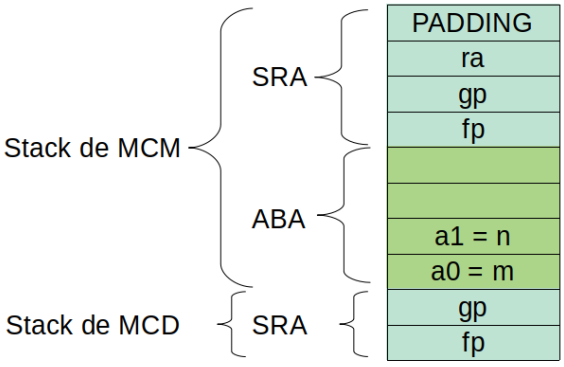
- ./tp -h
- ./tp -v
- ./tp -o nombreArchivo M N
- ./tp -m M N
- ./tp -d M N

Para la ejecución de las mismas se puede enviar la palabra completa, por ejemplo $-help$ o $--help$.

- -o requiere de un nombre de archivo a continuación, esto es para indicar que se quiere la salida del programa en un archivo en especial. Si se le envía '-' se considera que no se quiere un archivo, si no que se le mande a stdout. Se obtiene el mismo resultado si no se le manda -o.
- M es el primer numero que se debe enviar.
- N es el segundo numero a enviar.
- -m y -d sirven para indicar que se quiere obtener solamente uno de los resultados, ya sea el múltiplo o el divisor. Si se mandan ambos en la misma ejecución del programa se considera como un error y se le indica al usuario. Si se quieren obtener ambos números no se debe mandar alguno de ellos.

3. Diagramas de stack

Mostramos el stack de las funciones mcm y mcd para nuestra implementación en MIPS32:



Explicamos el diagrama:

Dado que mcm es una función no hoja (pues invoca a mcd) debe guardar en su SRA el valor del registro ra, además de los registros gp y fp. Una convención del ABI es que el SRA tenga una cantidad de bytes múltiplo de 8 y por lo tanto debemos agregar 4 bytes de padding al SRA. De esta forma el SRA de mcm tiene 16 bytes de tamaño.

Como ya mencionamos, mcm es una función no hoja y que invoca a mcd. Al invocar a mcd envía los parámetros m y n. Estos parámetros se envían por los registros a0 y a1 respectivamente

y sera responsabilidad de la función callee salvarlos en el ABA de la función caller. En este caso es responsabilidad de mcd almacenar estos registros en el ABA de mcm. Para que esto sea posible mcm debe dejar 16 bytes disponibles para el ABA, a pesar de que solo se utilizaran 8 de estos 16 bytes.

Respecto de mcd podemos decir que es una función hoja y que por lo tanto no salva el registro ra en su SRA. Además tampoco tendrá ABA.

En la implementación MIPS32 que hemos propuesto para estas funciones no hemos utilizado el LTA. Esto es porque las funciones eran simples por lo que nos pareció conveniente utilizar únicamente registros para almacenar las variables temporales y locales.

4. Proceso de compilación

Para poder realizar la compilación del trabajo se recomienda utilizar el *makefile* mediante la siguiente linea en la terminal.

```
make tp
```

Este *makefile* compila con una variedad de argumentos y distingue si se esta utilizando la arquitectura mips o alguna otra. Dependiendo del caso compila para el sistema que corresponda. Se tiene también la opción de utilizar *make clean*. La cual limpia los archivos generados por el make.

Si se quiere utilizar la linea se pueden usar las siguientes lineas dependiendo de la arquitectura.

```
gcc main.c euclides.c -o tp
gcc main.c euclides.c mcd.S mcm.S -DUSE_MIPS -o tp
```

5. Portabilidad

El programa fue hecho en gran parte en el lenguaje C. Las funciones mcm y mcd se realizaron tanto en C como en assembly. Esto busca dar un soporte genérico en entornos que no posean una versión más específica.

El programa lo hemos creado en el sistema operativo Ubuntu y en la arquitectura de MIPS emulada por QEMU.

6. Casos de prueba

Con el objetivo de testear el correcto funcionamiento del programa hemos creado un archivo bash con pruebas automatizadas.

Las pruebas en el archivo buscan evaluar el comportamiento del programa comparando el resultado obtenido en cada prueba con el resultado esperado.

Estas pruebas son:

- Se buscan el mcd y mcm de 5 10 (Prueba 1)
- Se busca solo el mcd de 5 10 (Prueba 2)
- Se busca solo el mcm de 5 10 (Prueba 3)
- Se buscan el mcd y mcm de 256 192 (Prueba 4)
- Se busca solo el mcd de 256 192 (Prueba 5)
- Se busca solo el mcm de 256 192 (Prueba 6)

- Se buscan el mcd y mcm de 1111 1294 (Prueba 7)
- Se busca solo el mcd de 1111 1294 (Prueba 8)
- Se busca solo el mcm de 1111 1294 (Prueba 9)
- No se le manda ningún argumento al TP (Prueba 10)
- Se le manda un solo numero al TP (Prueba 11)
- Se le manda un numero negativo al TP (Prueba 12)
- Se le manda un numero que no es valido (1) al TP (Prueba 13)
- Se le manda un numero con el que ocurre overflow (Prueba 14)
- Se le manda un argumento que excede la representación del int(Prueba 15)
- Se mandan números con las cuales la multiplicación de mcm da overflow (Prueba 16)
- Se envían juntos los parámetros -m y -d (Prueba 17)
- Se envían el parámetro -o sin argumento (Prueba 18)
- Se envían el parámetro -o sin argumento junto con un solo numero (Prueba 19)
- Se envían el parámetro -o sin argumento junto con dos números (Prueba 20)
- Se envían palabras en lugar de números (Prueba 21)
- Se envían caracteres especiales en lugar de números (Prueba 22)

Para correr las pruebas ejecutar:

```
bash pruebas.sh
```

Se deja el archivo realizado con las pruebas en el apéndice.

7. Conclusiones

A lo largo de la resolución del trabajo practico fueron surgiendo diferentes conclusiones. Las mismas se detallan a continuación.

- Se pudo observar la diferencia que ocurre al apreciar el código generado por el compilador y el realizado por nosotros. Las diferencias que se observaron consisten principalmente en la cantidad de directivas agregadas y de instrucciones que no resultan necesarias al momento de realizar los cálculos.
- Se aprendió sobre las convenciones de la ABI en el armado de las funciones del máximo común divisor y mínimo común múltiplo. Esto es tanto para el caso de las funciones hoja y de las no hojas, ya que hay que tener distintas consideraciones para cada caso. Esto abarca los ABA, SRA, y los LTA.
- Se aprendieron diferentes propiedades del mcm y mcd, siendo una de ellas la inversión que tienen entre si, de forma tal de obtener el otro.

8. Referencias

Realease en github,
https://github.com/brunograssano/Organizacion-de-computadoras-fiuba/releases/tag/v1.1.0
Algoritmo de Euclides, http://http://es.wikipedia.org/wiki/ Algoritmo_de_Euclides.

9. Apéndices

9.1. Pruebas

Se muestra el archivo bash realizado.

```
#!/bin/bash
espacios() {
    echo
    echo
}
newline=$'\n'

echo Comienza la ejecucion de las pruebas

espacios

        echo -e "\e[1m PRUEBA 1 \e[0m" - Se buscan el mcd y mcm de 5 10
        echo

        echo RESULTADO ESPERADO:
        resultadoEsperado="$(printf "5\n10")"
        echo $resultadoEsperado
        echo RESULTADO OBTENIDO:
        resultadoObtenido=$(./tp 5 10 )
        echo $resultadoObtenido
        echo
        if [ "$resultadoEsperado" == "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

        echo -e "\e[1m PRUEBA 2 \e[0m" - Se busca solo el mcd de 5 10
        echo

        echo RESULTADO ESPERADO:
        resultadoEsperado="5"
        echo $resultadoEsperado
        echo RESULTADO OBTENIDO:
        resultadoObtenido=$(./tp -d 5 10 )
        echo $resultadoObtenido
        echo
        if [ "$resultadoEsperado" == "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

        echo -e "\e[1m PRUEBA 3 \e[0m" - Se busca solo el mcm de 5 10
        echo

        echo RESULTADO ESPERADO:
```

```
resultadoEsperado="10"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp -m 5 10 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 4 \e[0m" - Se buscan el mcd y mcm de 256 192
echo

echo RESULTADO ESPERADO:
resultadoEsperado="$(printf "64\n768")"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp 256 192 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 5 \e[0m" - Se busca solo el mcd de 256 192
echo

echo RESULTADO ESPERADO:
resultadoEsperado="64"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp -d 256 192 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 6 \e[0m" - Se busca solo el mcm de 256 192
echo

echo RESULTADO ESPERADO:
resultadoEsperado="768"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
```

```
resultadoObtenido=$(./tp -m 256 192 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 7 \e[0m" - Se buscan el mcd y mcm de 1111 1294
echo

echo RESULTADO ESPERADO:
resultadoEsperado="$(printf "1\n1437634")"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp 1111 1294 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 8 \e[0m" - Se busca solo el mcd de 1111 1294
echo

echo RESULTADO ESPERADO:
resultadoEsperado="1"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp -d 1111 1294 )
echo $resultadoObtenido
echo
if [ "$resultadoEsperado" == "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi

echo -e "\e[1m PRUEBA 9 \e[0m" - Se busca solo el mcm de 1111 1294
echo

echo RESULTADO ESPERADO:
resultadoEsperado="1437634"
echo $resultadoEsperado
echo RESULTADO OBTENIDO:
resultadoObtenido=$(./tp -m 1111 1294 )
echo $resultadoObtenido
echo
```

```
        if [ "$resultadoEsperado" == "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

        echo -e "\e[1m PRUEBA 10 \e[0m" - No mandamos ningun argumento al tp y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp"
        echo "Luego: echo $ ?"
        echo " "
        ./tp
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 11 \e[0m" - Le enviamos un solo numero al tp y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp 8"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 8
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 12 \e[0m" - Le enviamos un numero negativo al tp y
                                            devolvemos distinto de 0
        echo " "
```

```
        echo "Ejecutamos: ./tp 8 -15"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 8 -15
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"

        fi
espacios

        echo -e "\e[1m PRUEBA 13 \e[0m" - Le enviamos un uno y otro numero al tp y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp 8 1"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 8 1
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 14 \e[0m" - Le enviamos un numero cuyo mcm da overflow y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp 8 4294967295"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 8 4294967295
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
```

```
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 15 \e[0m" - Le enviamos un numero fuera del rango del int al tp
                                            y devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp 2 4294967300"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 8 4294967300
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 16 \e[0m" - Le enviamos otros 2 numeros cuyo mcm da overflow
                                            y devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp 3 4294967295"
        echo "Luego: echo $ ?"
        echo " "
        ./tp 3 4294967295
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 17 \e[0m" - Le enviamos -m -d y devolvemos distinto de 0
```

```
        echo " "
        echo "Ejecutamos: ./tp -m -d 2 9"
        echo "Luego: echo $ ?"
        echo " "
        ./tp -m -d 2 9
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 18 \e[0m" - Le enviamos -o sin parametro y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp -o"
        echo "Luego: echo $ ?"
        echo " "
        ./tp -o
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 19 \e[0m" - Le enviamos -o con un solo numero y
                                             devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp -o 9"
        echo "Luego: echo $ ?"
        echo " "
        ./tp -o 9
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
```

```
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 20 \e[0m" - Le enviamos -o con dos numeros y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp -o 9 16"
        echo "Luego: echo $ ?"
        echo " "
        ./tp -o 9 16
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi

espacios

        echo -e "\e[1m PRUEBA 21 \e[0m" - Le enviamos palabras en lugar de numeros y
                                            devolvemos distinto de 0
        echo " "
        echo "Ejecutamos: ./tp esto falla"
        echo "Luego: echo $ ?"
        echo " "
        ./tp esto falla
        resultadoObtenido=$(echo $?)
        echo "Ejecutamos y obtenemos:"
          echo "RESULTADO ESPERADO:"
          echo "Distinto de 0"
        echo "RESULTADO OBTENIDO:"
        printf "$resultadoObtenido\n"
        if [ "0" != "$resultadoObtenido" ];
        then
                        echo -e "\e[32m PRUEBA SUPERADA \e[0m"
        else
                        echo -e "\e[31m PRUEBA FALLADA \e[0m"
        fi
espacios

        echo -e "\e[1m PRUEBA 22 \e[0m" - Le enviamos caracteres especiales y
```

```
                                        devolvemos distinto de 0
echo " "
echo "Ejecutamos: ./tp @ !"
echo "Luego: echo $ ?"
echo " "
./tp @ !
resultadoObtenido=$(echo $?)
echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  echo "Distinto de 0"
echo "RESULTADO OBTENIDO:"
printf "$resultadoObtenido\n"
if [ "0" != "$resultadoObtenido" ];
then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
fi
```

9.2. Código C

9.2.1. main.c

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <limits.h>
#include "euclides.h"
#include <errno.h>

#define DIVISOR 'd'
#define MULTIPLO 'm'
#define OUTPUT 'o'
#define VERSION 'v'
#define AYUDA 'h'
#define MAX_NOMBRE_ARCHIVO 256
#define MODO_ESCRITURA "w"

const int ERROR = -1, VACIO=0, TERMINO = -1;
extern int errno;

void mostrarAyuda(){
  printf("Uso: \n");
  printf(" tp -h\n");
  printf(" tp -v\n");
  printf(" tp [opciones] primerNumero segundoNumero\n");
  printf("Opciones: \n");
  printf(" -v, --version    Imprime la version y termina el programa.\n");
  printf(" -h, --help       Imprime esta informacion.\n");
  printf(" -o, --output     Indica que le sigue la direccion al archivo de salida.\n");
  printf(" -m, --multiple   Entrega solo el multiplo.\n");
```

```
  printf(" -d, --divisor    Entrega solo el divisor.\n");
  printf("Ejemplos: \n");
  printf("        tp -o archivoSalida 256 192\n");
  printf("        tp -o - 256 192 (En este caso se toma como que se quiere stdout)\n");
  printf("        tp 256 192\n");
  printf("        tp -m 256 192\n");
  printf("        tp -d 256 192\n");
  printf("        tp -o -m archivoSalida 256 192\n");
}

void mostrarVersion(){
         printf("Version 1.0.0\n");
}

configuracion_t manejarParametros(int cantidadArgumentos, char* argumentos[],
        char archivoOutput[MAX_NOMBRE_ARCHIVO]){
  static struct option opcionesLargas[] = {
     {"multiple", no_argument, 0, 'm'},
     {"divisor", no_argument, 0, 'd'},
     {"help", no_argument, 0, 'h'},
     {"output", required_argument, 0, 'o'},
     {"version", no_argument, 0, 'v'},
     {0, 0, 0, 0}
  };
  configuracion_t configuracion = {false,false,false,false,0,0};
  int argumento;
  int indiceOpcion = 0;
  bool pidioAyuda = false, pidioVersion = false;

  while((argumento = getopt_long(cantidadArgumentos, argumentos, "o:hvdm",opcionesLargas,
                &indiceOpcion))!=TERMINO){
      switch (argumento) {
          case DIVISOR:
              configuracion.soloDivisor = true;
              break;
          case MULTIPLO:
              configuracion.soloMultiplo = true;
              break;
          case OUTPUT:
              strcpy(archivoOutput,optarg);
              break;
          case VERSION:
              if(!pidioVersion){
                  configuracion.pidioOtraOpcion = true;
                  pidioVersion = true;
                  mostrarVersion();
              }
              break;
          case AYUDA:
              if(!pidioAyuda){
                  configuracion.pidioOtraOpcion = true;
                  pidioAyuda = true;
                  mostrarAyuda();
              }
```

```
                break;
            default:
                fprintf(stderr, "Puede ver ayuda enviando el parametro -h \n");
        }
    }

    if(optind<cantidadArgumentos){
      errno = 0;
      unsigned long numeroCompleto = strtoul(argumentos[optind], NULL, 10);
      if(numeroCompleto>UINT_MAX || errno == ERANGE){
        configuracion.overflow = true;
      }
        configuracion.primerNumero = (unsigned int)numeroCompleto;
        optind++;
    }
    if(optind<cantidadArgumentos){
      errno = 0;
      unsigned long numeroCompleto = strtoul(argumentos[optind], NULL, 10);
      if(numeroCompleto>UINT_MAX || errno == ERANGE){
        configuracion.overflow = true;
      }
      configuracion.segundoNumero = (unsigned int)numeroCompleto;
    }
    return configuracion;
}

int main(int cantidadArgumentos, char* argumentos[]){
    int estado = 0;
    char archivoOutput[MAX_NOMBRE_ARCHIVO] = "";
    if(cantidadArgumentos == 1){
      fprintf(stderr, "No se enviaron argumentos. Puede ver ayuda mandando -h\n");
      return ERROR;
    }
    configuracion_t configuracion = manejarParametros(cantidadArgumentos,argumentos,archivoOutput);

    if(configuracion.overflow){
        fprintf(stderr, "Los numeros enviados no pueden ser mayores al limite del int.
                  Puede ver ayuda mandando -h\n");
        return ERROR;
    }

    if(!configuracion.pidioOtraOpcion &&
          (configuracion.primerNumero<2 || configuracion.segundoNumero<2)){
        fprintf(stderr, "Los numeros tienen que ser mayor o igual a 2.
                  Puede ver ayuda mandando -h\n");
        return ERROR;
    }

    if(!configuracion.pidioOtraOpcion && configuracion.soloDivisor && configuracion.soloMultiplo){
        fprintf(stderr, "Uso mal las opciones, no es valido mandar -d y -m juntos.
                  Puede ver ayuda mandando -h\n");
        return ERROR;
    }
```

```
  if(strlen(archivoOutput)>VACIO && archivoOutput[0]!='-' && !configuracion.pidioOtraOpcion){
      FILE* fileOutput = fopen(archivoOutput,MODO_ESCRITURA);
      if(fileOutput==NULL){
          fprintf(stderr, "No se pudo abrir el archivo enviado.\n");
          return ERROR;
      }
      estado = buscarNumeros(configuracion,fileOutput);
      fclose(fileOutput);
  }
  else if(!configuracion.pidioOtraOpcion){
      estado = buscarNumeros(configuracion,stdout);
  }

  return estado;
}
```

9.2.2. euclides.c

```
#include <limits.h>
#include "euclides.h"
#define ERROR -1
#define EXITO 0

#ifndef USE_MIPS

unsigned int mcd(unsigned int m, unsigned int n){
    unsigned int maximoEncontrado;
    while(n!=0){
        maximoEncontrado = n;
        n = m % n;
        m = maximoEncontrado;
    }
    return m;
}

unsigned int mcm(unsigned int m, unsigned int n){
    unsigned long producto = (unsigned long)m*(unsigned long)n;
    if(producto > UINT_MAX){
        return 0;
    }
    return (m*n)/mcd(m,n);
}

#endif


void ordenarNumeros(configuracion_t* configuracion){
    if(configuracion->primerNumero < configuracion->segundoNumero){
        unsigned int aux = configuracion->primerNumero;
        configuracion->primerNumero = configuracion->segundoNumero;
        configuracion->segundoNumero = aux;
    }
}
```

```c
int buscarNumeros(configuracion_t configuracion,FILE* salida){
    unsigned int minimoComunMultiplo;
    unsigned int maximoComunDivisor;

    ordenarNumeros(&configuracion);

    unsigned int m = configuracion.primerNumero;
    unsigned int n = configuracion.segundoNumero;

    if(configuracion.soloMultiplo){
        minimoComunMultiplo = mcm(m,n);
        if(minimoComunMultiplo == 0){
            fprintf(stderr, "Hubo problemas por overflow al intentar calcular el mcm\n");
            return ERROR;
        }
        fprintf(salida,"%i",minimoComunMultiplo);
    }
    else if(configuracion.soloDivisor){
        maximoComunDivisor = mcd(m,n);
        fprintf(salida,"%i",maximoComunDivisor);
    }
    else{
        minimoComunMultiplo = mcm(m,n);
        if(minimoComunMultiplo == 0){
            fprintf(stderr, "Hubo problemas por overflow al intentar calcular el mcm\n");
            return ERROR;
        }
        maximoComunDivisor = mcd(m,n);
        fprintf(salida,"%i\n%i",maximoComunDivisor,minimoComunMultiplo);
    }

    return EXITO;
}
```

9.3. Código S

9.3.1. mcd.S

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

#define SS 8/*Stack size*/

/*ABA CALLER*/
#define O_A1 (SS+4)
#define O_A0 (SS)

/*SRA (de 0 a 8)*/
#define O_FP 4
#define O_GP 0

/*LTA (no lo usamos)*/
/*ABA MCD (es hoja)*/

.globl mcd
```

```
.text
.align 2
.ent mcd
.cprestore 0_GP

mcd:

        subu sp, sp, SS

        /*SRA*/
        sw fp, 0_FP(sp)
        sw gp, 0_GP(sp)
        move fp, sp

        /*ABA del caller*/
        sw a0, 0_A0(sp)     /*Guardamos lo recibido en el ABA del caller*/
        sw a1, 0_A1(sp)

loop:
        beq a1, zero, finMcd    /* Si n == 0 -> finMcd*/
        add t0, zero, a1        /* t0 <- numero n*/
        div a0, a1              /* m/n */
        mfhi a1                 /* a1 <- m mod n (el modulo se guarda en $HI)*/
        add a0, t0, zero        /* a0 <- numero n */
        j loop


finMcd:
        add v0, zero, a0                        /*v0 <- numero mcd (Se devuelve por v0)*/

        lw gp, 0_GP(sp)                             /* Restauramos gp y fp*/
        lw fp, 0_FP(sp)

        /*Desarmar stack*/
        addiu sp, sp, SS

        jr ra

.end mcd
```

9.3.2. mcm.S

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

#define SS 32/*Stack size*/

/*ABA CALLER*/
#define 0_A1 (SS+4)
#define 0_A0 (SS)

/*SRA (de 24 a 36)*/
#define 0_RA 24
#define 0_GP 20
#define 0_FP 16
```

```
/*LTA (no se usa)*/
/*VARIABLES LOCALES*/
/*VARIABLES TEMPORALES*/

/*ABA  (de 0 a 16)*/

.extern mcd
.globl mcm
.text
.align 2
.ent mcm
.cprestore 0_GP

mcm:
        /*Hacemos espacio para el stack*/
        subu sp, sp, SS

        /*SRA*/
        sw ra, 0_RA(sp)     /* Guardamos el retorno */
        sw fp, 0_FP(sp)
        sw gp, 0_GP(sp)
        move fp, sp

        /*ABA CALLER*/
        sw a0, 0_A0(sp)     /*Guardamos lo recibido en el ABA del caller*/
        sw a1, 0_A1(sp)

        jal mcd

        lw a0, 0_A0(sp)     /* Obtenemos lo recibido devuelta*/
        lw a1, 0_A1(sp)

        add t0, zero, zero

        multu a0, a1        /*$LO <- m * n (se guarda el resultado en $LO)*/

        mfhi t1
        bne t1, zero, overflow

        mflo t0             /*t0 <- m*n (Obtenemos el resultado)*/

        div t0, v0          /* En v0 esta el resultado de mcd */

        mflo v0             /* Devolvemos tambien en v0 */

return:
        lw ra, 0_RA(sp)     /* Restauramos gp,fp y obtenemos la direccion de retorno*/
        lw gp, 0_GP(sp)
        lw fp, 0_FP(sp)

        /*Desarmar stack*/
        addiu sp, sp, SS
```

```
        jr ra

overflow:
        add v0, zero, zero
        j return

.end mcm
```

9.4. Enunciado

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

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`.