TP1
Organización de Computadoras

Ramiro Sánchez — Lukas De Angelis Riva — Matías Merlo
104095 — 103784 — 104093

November 12, 2020

1 Introducción

El objetivo de este trabajo prácitco es familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI[1]. Escribiendo un programa portable que resuelva el máximo común divisor y/o mínimo común múltiplo entre dos números implementando el algoritmo de Euclides.

2 Detalles de diseño

Al ejecutar el programa por consola es necesario especificar los parámetros que se utilizarán como se explica en la sección 3. Para realizar únicamente el mcm o el mcd será necesario especificarlo con los flags pertinentes, en caso de no especificar: se harán ambos.

Se especifica con -o el archivo de salida seguido de los dos números con los cuales operar. Si se envía la cadena "-" como nombre de archivo de salida: Será interpretado como salida estándar y por lo tanto se imprimirá el resultado por stdout.

Los dos números a operar deben estar seguidos al nombre del archivo especificado con el flag -o, en caso contrario se devolverá un error. Hay una única excepción y es ejecutar el programa únicamente con la entrada de los dos números, pero se prohibe la entrada de flags: resultando que se imprima por stdout el mcd y el mcm.
La decisión de prohibir la entrada de flags cuando se ejecuta con la forma especial:

```
./tp1 256 192
```

Fue para simplificar el parseo[2] de la entrada y no enfocarse en eso a la hora de realizar el trabajo.

Luego todas las demás formas admiten cualquier orden en los flags, siempre y cuando los parámetros numéricos acompañen al flag -o.

En caso de que algún numero ingresado no cumpla las condiciones necesarias:

- No sea numérico.
- No pertenezca al rango [2, UINT_MAX[3]]

Se enviarán por stderr mensajes especificando el error.

Además, si al calcular el mcm la multiplicación de los números m y n provocan un overflow[4], se informará de esta situación por stderr.

[1] ABI: Application Binary Interface; Interfaz binaria de aplicaciones
[2] Parser: Analizador sintáctico. Parsear: Analizar sintácticamente
[3] UINT_MAX: 4294967295U
[4] Overflow: Desbordamiento de búfer

3 Recomendaciones y pautas de uso

Las opciones que acepta el programa son:

```
-h, --help      Muestra ayuda por pantalla.
-V, --version   Muestra la version del programa.
-o, --output    Direccion al archivo de salida.
-d, --divisor   Imprimir solo el mcd.
-m, --multiple  Imprimir solo el mcm.
```

Una forma de tener una salida por pantalla es pasando "-" como nombre del archivo de salida.

Ejemplo:

```
> ./tp1 -o - 256 192
64
768
```

Para especificar un archivo de salida el flag -o admite como primer parámetro el nombre de un archivo donde será escrito el resultado.
Si el archivo ya existe: se sobrescribirá.

Ejemplo:

```
> ./tp1 -o archivo.txt 256 192
> cat archivo.txt
64
768
```

Se da prioridad a las opciones -h y -v.
Esto quiere decir que si se ejecutara el programa de la siguiente manera:

```
> ./tp1 -o - 256 192 -d -h
```

La salida por pantalla será la de la opción de ayudas.
Ídem con la opción de versión:

```
> ./tp1 -o - 256 192 -d -v
```

Como caso excepcional, permitimos que se manden como argumentos del programa solamente 2 números. En este caso se imprimirán tanto el mcd como el mcm por stdout, además de que esta forma especial no admite más argumentos. Por ejemplo:

```
> ./tp1 256 192
64
768
```

Pero si se ejecutara la siguiente linea:

```
> ./tp1 256 192 -d
```

Se envía por stderr un mensaje informando un error al parsear la entrada.[5]

[5] La explicación de por qué tomamos esta decisión se dio en la sección de Detalles de diseño.

4 Pruebas

Se escribieron pruebas en shell del programa que podrá comprobar ejecuntando el comando

```
> sh tests
```

Los tests comprueban el correcto funcionamiento de la implementación en los casos que consideramos borde:

- Test01: Calcula mcd con 2 números validos.
- Test02: Calcula mcm con 2 números validos.
- Test03: Calcula mcd y mcm con 2 números validos y salen por stdout.
- Test04: Calcula mcd y mcm con 2 números validos y salen por un archivo.
- Test05: Ingresa un número de más.
- Test06: Ingresa un número de menos.
- Test07: Ingresa una letra en vez de un número.
- Test08: Ingresa un número mayor a UINT_MAX.
- Test09: La multiplicación da overflow.
- Test10: Ingresa un 1.
- Test11: Ingresa un 0.
- Test12: Ingresa un argumento incorrecto.
- Test13: Ingresa un número negativo.

5 Conclusiones

Con esta primera entrega del trabajo práctico Nº1 hemos entendido la importancia de utilizar la ABI, algunos de los motivos que encontramos fueron:

- Gracias a la estandarización se facilita la comunicación entre diferentes módulos y subrutinas.
  Por ejemplo: Los llamados a subrutinas. En el llamado al mcd desde el código generado por gcc o el llamado a mcd desde la subrutina de mcm se realiza de la misma forma, lo que quiere decir que mcd recibe parámetros mediante una interfaz, y por contraparte devuelve resultados también mediante una interfaz, facilitando la vuelta.
- Facilita además la forma de guardar la información, sea mediante registros "saved" o registros temporales. De forma tal que una subrutina deja el ambiente de la rutina que la llamó de forma intacta.

6 Archivos entregados

Los archivos .c .h y .s entregados se encuentran en el repositorio de github, cuyo link está en la sección de Links. Los archivos con código entregados son:

- TP1.c
- TP1_MIPS.S
- TP1_x86_64.S
- makefile
- common.h
- common.c
- mcd.S
- mcm.S

El archivo TP1_MIPS.S tiene el código assembly generado por

```
objdump -S tp1
```

cuando el ejecutable tp1 fue compilado en la arquitectura MIPS

A diferencia del archivo TP1_x86_64.S que es el código assembly generado nuevamente por

```
objdump -S tp1
```

cuando el ejecutable tp1 fue compilado en la máquina host

Para compilar el programa:

```
make tp1
```

7 Links

Link al repositorio de Github:
https://github.com/RamiroSanchez-dev/Organizacion-de-Computadoras