Trabajo Práctico Nº 1: Conjunto de instrucciones MIPS

Sebastian Ripari, *Padrón Nro. 96.453*
sebastiandanielripari@hotmail.com

Cesar Emanuel Lencina, *Padrón Nro. 96.078*
cesar_1990@live.com

Pablo Sivori, *Padrón Nro. 84.026*
sivori.daniel@gmail.com

2do. Cuatrimestre de 2017
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

5 de octubre de 2017

Resumen

Se implemento un programa que realiza el calculo del maximo comun divisor y del minimo comun multiplo, mediante el uso del Algoritmo de Euclides. Para la implementacion del algortimo, se utilizo el lenguaje C, y con la particularidad de las funciones matematicas fueron llevadas a cabo usando Assembler de MIPS. Por ende la compilacion del programa, comprende el linkeo de estas funciones en Assembler.

1. Introducción

El *algoritmo de Euclides* es un método antiguo y eficaz para calcular el máximo común divisor (MCD). Fue originalmente descrito por Euclides en su obra Elementos. El *algoritmo de Euclides extendido* es una ligera modificacion que permite ademas expresar al maximo comun divisor como una combinacion lineal.

2. Desarrollo

2.1. Consideraciones:

1. Antes de realizar la codificación de las funciones MCD y MCM en código MIPS 32 se hizo un gráfico del stack frame de ambas, teniendo en cuenta que la función MCD se la dibujo sin ABA y sin el registro ra (return address), y lo contrario con la función MCM la cual invoca a la anterior, resultando ser una función no hoja. Siguiendo los lineamientos del stack frame (SF) se crean 4 áreas en el SF de cada función (SRA, FRA, LTA y ABA).

2. Cada area del stack frame debe tener su padding si lo necesita, para que sean múltiplos de 8 bytes.

3. Con el stack de ambas funciones comenzamos pasar el código de C a lenguaje MIPS 32, respetando la convención de la ABI.

4. En el código MIPS 32 utilizamos el include mips/regdef.h para utilizar las constantes sp, fp, t1, ..., etc y de esta manera no tener que utilizar los números de registros `($0,$1,....,$32)`.

5. Se utiliza la directiva .globl mcd y .globl mcm para que ambas funciones puedan ser llamadas desde otro archivo.

6. Se crean constantes para definir el tamaño del stack frame de las funciones así como también la posición de los registros fp, sp y ra. De esta manera si hay algún cambio en la posición donde se encuentran en el stack, solo modificamos el valor de la constante.

7. Ponemos el align 2 para que las instrucciones estén alineadas 4 bytes. De esta manera el program counter avanzará de a 4 bytes.

8. Para facilitar el paso anterior, se creo una carpeta compartida al netbsd, la cual contenia los archivos con el código fuente a ejecutar. Para esto

montamos la carpepeta una vez realizado el tunel. Para esto ejecutamos el comando:
sshfs -p 2222 -C root@127.0.0.1:/root/tp1/
/home/username/workspace/orga6620_mounted/.
De esta manera podemos editar los archivos directamente y luego compilarlos en netbsd sin tener la necesidad de hacer una transferencia (copy) de los archivos modificados a esta carpeta.

2.2. Proceso de compilación:

Para compilar el programa utilizamos el siguiente Makefile:

```
# Build version
VERSION = 0

# Compiler and env set up
CC=gcc
CFLAGS = -Wall -O0
OBJ = common.o commonfunc.o

# Rules
default: clean bin

bin: $(CC) $(CFLAGS) -o common common.c commonfunc.c mathfunc.S
-D_VERSION_=¨$(VERSION)¨

clean: $(RM) common
```

En CFLAGS ponemos -O0 para apagar las optimizaciones.

Desde la terminal nos posicionamos en la carpeta tp1_orga6620 y corremos el comando make. De esta manera ira a la regla default ejecutando el clean y luego el bin para generar el archivo common (nombre del programa ejecutable). Esto se corre en netbsd para generar el ejecutable que incluya el código MIPS 32 de las funciones, el cual se encuentra en el archivo mathfunc.S.

3. Casos de prueba:

Ejecutamos desde la carpeta tp1_orga6620/pruebas/ el script pruebas.sh, en netbsd con gxemul, con el siguiente comando bash pruebas.sh. De esta manera obtenemos el siguiente resultado:

```
root@:~/tp1/tp1_orga6620/Pruebas# bash pruebas.sh—
```

3.1. Comienzo de pruebas:

3.1.1. Test 1

Mostramos el mensaje de ayuda usando la opcion -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 19
```

3.1.2. Test 2

Mostramos la versión del common usando la opción -V.
Common version 1.0

3.1.3. Test 3

Mostramos por stdout el máximo común divisor entre 5 y 10.
5

3.1.4. Test 4

Mostramos por stdout el minimo común múltiplo entre 5 y 10.
10

3.1.5. Test 5

Mostramos por stdout el mcm y el mcd entre 5 y 10.
5
10

3.1.6. Test 6

Mostramos por stdout el mcd entre 256 y 192.
64

3.1.7. Test 7

Mostramos por stdout el mcm entre 256 y 192.
768

3.1.8. Test 8

Mostramos por stdout el mcd y el mcm entre 256 y 192.
64
768

3.1.9. Test 9

Mostramos por stdout el mcd entre 1111 y 1294.
1

3.1.10. Test 10

Mostramos por stdout el mcm entre 1111 y 1294.
1437634

3.1.11. Test 11

Mostramos por stdout el mcd y el mcm entre 1111 y 1294.
1
1437634

3.1.12. Test 12

Ingresamos un comando invalido (./common -i 5 10).

3.1.13. Test 13

Ingresamos un argumento extra en la opción -h (./common -h 10).

3.1.14. Test 14

Ingresamos un comando invalido (./common aaa bbb ccc).

3.1.15. Test 15

No pasamos ningún parámetro (./common).

3.1.16. Test 16

No pasamos el parámetro número en la opción -d (./common -d).

3.1.17. Test 17

Pasamos letras en lugar del número (./common -d -o - sss).

4. Partes del codigo relevantes

4.1. MCD

#guardo en la aba de la funcion que me llamo los argumentos sw a0, SF_MCM_A0_POS($fp)
sw a1, SF_MCM_A1_POS($fp)

Esto es para respetar la convención de la ABI, dado que la función llamada deberá preservar los valores de la función llamante.

4.2. MCM

```
    # multiplico numeroBajo*numeroAlto = numerado
mult a0, a1

# guardo en t0 el resultado de la multiplicacion anterior (numerador) mflo
t0

# guardo en el stack el numerador
sw t0, 16($fp)

# llamo a la funcion mcd con los parametros a0 y a1
jal mcd

    Sigue:
#guardo en el stack el valor de mcd, valor que retorno la funcion (es el deno-
minador)
sw v0, 20($fp)

#recupero del stack el valor de numeroBajo*numeroAlto
lw t0, 16($fp)

#Hago (numeroBajo*numeroAlto)/(mcd(numeroBajo,numeroAlto))
div t0,v0
mflo t1

#Guardo en V1 el resultado de mcm
move v0,t1

#Guardo en el stack el resultado de mcm
sw t1,24($fp)
```

Aquí es importante destacar que antes de hacer el llamado a la función mcd, se guarda el valor de la variable local de la función (t0) en el stack frame de la función. Esto es porque la función mcd puede utilizar este registro y en consecuencia si modifica el valor, luego cuando retorne a la siguiente instrucción de mcm, obtendremos un error en el cálculo, dado que se perdió el valor original de la variable local.

5. Diagrama del Stack Frame

5.1. MCD

5.2. MCM

Texto de la otra sección. En la figura 1 se muestra un ejemplo de cómo presentar las ilustraciones del informe.

Figura 1: Facultad de Ingeniería − Universidad de Buenos Aires.

6. Conclusiones

Este trabajo práctico nos sirvió para ver cómo utilizar correctamente el stack de una función, respetando la convención de la ABI. También notamos que es necesario tener presente el esquema del stack de cada función dado que nos resultó muy importante a la hora de realizar el código de ambas funciones en MIPS 32. Otro hito importante fue el uso de gdb, mediante el cual pudimos debuguear el programa para solucionar algunos inconvenientes presentados en las funciones hechas en MIPS 32. Con gdb pudimos ir corroborando los valores que iban tomando los registros al momento de ejecutar una instrucción, y de esta manera poder detectar la línea que teníamos que corregir si había algún valor que no era el esperado.

Referencias

[1] Algoritmo de Euclides, https://es.wikipedia.org/wiki/Algoritmo_de_Euclides.