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

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.

Índice

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 denominador)
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

	24		
	20	a1	ABA Caller
	16	a0	
Stack Frame MCD	12	fp	SRA
	8	gp	
	4		LTA
	0	auxiliar	
	Es funcion Hoja		

5.2. MCM

	56		
	52	a1	ABA caller
	48	a0	
Stack Frame MCM	44		SRA
	40	fp	
	36	gp	
	32	ra	
	28		LTA
	24	numerador	
	20	mcdDenominador	
	16	resultadoMcm	
	12	a3	ABA callee
	8	a2	
	4	a1	
	0	a0	
	No es function Hoja		

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.

7. El código fuente, en lenguaje C:

```
#include <stdio.h>
#include "commonfunc.h"


int main(int argc, char **argv){
        int alerta = validarArgumentos(argc, argv);
        if (alerta == TODO_OK){
                alerta = realizarAccion(argc, argv);
        }
        return alerta;
}
```

8. El código Assembler:

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

#########FUNCION HOJA MCD############

#STATICS VAR DEFINITIONS FUNCTION MCD
#define SF_SIZE_MCD         16
#define SF_MCD_FP_POS       12
#define SF_MCD_GP_POS        8

.text
.abicalls
.align 2
.globl mcd
.ent mcd


mcd:
        ###INCIALIZACION DEL STACK FRAME DE LA FUNCION MCD###
        .frame          $fp, SF_SIZE_MCD, ra
        .set            noreorder
        .cpload         t9
        .set            reorder
        subu            sp, sp, SF_SIZE_MCD
        sw              $fp, SF_MCD_FP_POS(sp)
        .cprestore      SF_MCD_GP_POS
        move            $fp, sp
        ###FIN INICIALIZACION DEL STACK FRAME DE LA FUNCION MCD###

        sw              a0, 16($fp)             #guardo los argumentos en la aba de la
        sw              a1, 20($fp)

        bgtz            a0, iterando_mcd        #Si a0 > 0, ingresa al loop
        move            v0, a1                  #Asigno en v0 el valor del parametro
        j               fin_mcd                 #Si a0 <=0, salta a fin

iterando_mcd:

        sw              a0, 0($fp)              #guardo en la posicion 0 del stack fra
        div             a1, a0                  #Calculo el resto de la division nume
        mfhi            t0                      #Guardo el valor del modulo en numero
        lw              t1, 0($fp)              #Obtengo el valor auxiliar, guardado
        move            a0, t0                  #El resto de la division lo guardo en
        move            a1, t1                  #Ahora el numeroAlto es el valor auxi
        bgtz            a0, iterando_mcd        #Si a0 es mayor a 0, sigue en el loop
fin_mcd

fin_mcd:
        move            v0, a1                  #El numero alto es lo que retorna la
        ###RESTAURO LOS REGISTROS###
        move            sp,  $fp
        lw              gp,  SF_MCD_GP_POS(sp)
```

```
        lw              $fp, SF_MCD_FP_POS(sp)
        ###DESTRUYO EL STACK FRAME###
        addu            sp, sp, SF_SIZE_MCD
        ###RETORNO A LA SIGUIENTE INSTRUCCION DE LA FUNC QUE LLAMO A CMD###
        jr              ra
        .end            mcd

#########FIN FUNCION HOJA MCD############

#########FUNCION NO HOJA MCM#############

#STATICS VAR DEFINITIONS FUNCTION MCM
#define SF_MCM_A1_POS      52
#define SF_MCM_A0_POS      48
#define SF_SIZE_MCM        48
#define SF_MCM_FP_POS      40
#define SF_MCM_GP_POS      36
#define SF_MCM_RA_POS      32

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

mcm:
        ###INCIALIZACION DEL STACK FRAME DE LA FUNCION MCM###
        .frame          $fp, SF_SIZE_MCM, ra
        .set            noreorder
        .cpload         t9
        .set            reorder
        subu            sp, sp, SF_SIZE_MCM
        sw              ra, SF_MCM_RA_POS(sp)
        sw              $fp, SF_MCM_FP_POS(sp)
        .cprestore      SF_MCM_GP_POS
        move            $fp, sp
        ###FIN INICIALIZACION DEL STACK FRAME DE LA FUNCION MCM###

        sw              a0, SF_MCM_A0_POS($fp)          #guardo en la aba de la func
        sw              a1, SF_MCM_A1_POS($fp)

        mult            a0, a1                          #multiplico numeroBajo*numero
        mflo            t0                              #guardo en t0 el resultado de
        sw              t0, 16($fp)                     #guardo en el stack el numera
        jal             mcd                             #llamo a la funcion mcd con l

sigue:
        sw              v0, 20($fp)                     #guardo en el stack el valor
        lw              t0, 16($fp)                     #recupero del stack el valor
        div             t0,v0                           #Hago (numeroBajo*numeroAlto)
        mflo            t1
        move            v0,t1                           #Guardo en V1 el resultado de
        sw              t1,24($fp)                      #Guardo en el stack el resulta
```

```
###FIN MCM###
###RESTAURO LOS REGISTROS###
move            sp, $fp
lw              ra, SF_MCM_RA_POS(sp)
lw              gp, SF_MCM_GP_POS(sp)
lw              $fp, SF_MCM_FP_POS(sp)
###DESTRUYO EL STACK FRAME###
addu            sp, sp, SF_SIZE_MCM
###RETORNO A LA SIGUIENTE INSTRUCCION DE LA FUNC QUE LLAMO A CMD###
jr              ra
.end            mcm
```

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 GXemul [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 NetBSD [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, NetBSD/pmax.

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 NetBSD 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, en formato digital[2].

9. Fecha de entrega

La última fecha de entrega es el jueves 5 de octubre de 2017.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.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] No usar diskettes: son propensos a fallar, y no todas las máquinas que vamos a usar en la corrección tienen lectora. En todo caso, consultá con tu ayudante.

Referencias

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

[2] Documentación GDB, https://www.gnu.org/software/gdb/documentation

[3] Llamadas a funciones, func_call_conv.pdf, Yahoo Groups - Orga-Comp

[4] Conjunto de instrucciones, MIPSQuickRef.pdf, Yahoo Groups - Orga-Comp