

Trabajo Práctico N° 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 implementó un programa que realiza el cálculo del máximo común divisor y del mínimo común múltiplo, mediante el uso del Algoritmo de Euclides. Para la implementación del algoritmo, se utilizó el lenguaje C, y con la particularidad de las funciones matemáticas fueron llevadas a cabo usando Assembler de MIPS. Por ende la compilación del programa, comprende el linkeo de estas funciones en Assembler.

Índice

1. Introducción	3
2. Desarrollo	3
2.1. Consideraciones:	3
2.2. Proceso de compilación:	4
3. Casos de prueba:	5
3.1. Comienzo de pruebas:	5
3.1.1. Test 1	5
3.1.2. Test 2	5
3.1.3. Test 3	5
3.1.4. Test 4	5
3.1.5. Test 5	5
3.1.6. Test 6	5
3.1.7. Test 7	6
3.1.8. Test 8	6
3.1.9. Test 9	6
3.1.10. Test 10	6
3.1.11. Test 11	6
3.1.12. Test 12	6
3.1.13. Test 13	6
3.1.14. Test 14	6
3.1.15. Test 15	6
3.1.16. Test 16	6
3.1.17. Test 17	6
4. Partes del código relevantes	7
4.1. MCD	7
4.2. MCM	7
5. Diagrama del Stack Frame	8
5.1. MCD	8
5.2. MCM	8
6. Conclusiones	9
7. El código fuente, en lenguaje C:	10
8. El código Assembler:	15

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 modificación que permite además expresar al máximo común divisor como una combinación 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 carpeta 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:

<code>-h,</code>	<code>--help</code>	Prints usage information.
<code>-V,</code>	<code>--version</code>	Prints version information.
<code>-o,</code>	<code>--output</code>	Path to output file.
<code>-d</code>	<code>--divisor</code>	Just the divisor
<code>-m</code>	<code>--multiple</code>	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 mínimo 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 código 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

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

Es función hoja (leaf), porque no llama a otras funciones. Por ser una función hoja no tiene ABA y la SRA no tiene ra. El tamaño del stack frame de la función mcd es de 16 words.

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	

Función no hoja (non leaf), porque llama a la función MCD. Por ser una función no hoja tiene ABA, dado que a través de esta área se guardan los parámetros que se utilizan para llamar a otras funciones. La función llamada (invocada) guarda en la ABA de la función que la llamó los parámetros antes de ser utilizados. Además en la saved register area (SRA) de la función llamante se guardará el valor de la dirección a la próxima instrucción al llamado de la función, luego de que la función llamada finalice retornando un valor (en el registro v0) o no.

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 debuggear 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:

common.c

```
1 #include <stdio.h>
2 #include "commonfunc.h"
3
4
5 int main(int argc, char **argv){
6     int alerta = validarArgumentos(argc, argv);
7     if (alerta == TODO_OK){
8         alerta = realizarAccion(argc, argv);
9     }
10    return alerta;
11 }
```

commonfunc.h

```
1 #define MAXIMO 4096
2 #define MINIMO 2
3
4 #define CAMPO_VACIO 0
5 #define PRIMERA_POSICION 0
6 #define SEGUNDA_POSICION 1
7 #define TERCERA_POSICION 2
8 #define INICIO_ARRAY 0
9
10 #define TODO_OK 0
11 #define ERROR_FUERA_DE_RANGO 1
12 #define ERROR_COMANDO_INVALIDO 2
13 #define ERROR_CREACION_DE_ARCHIVO 3
14
15 #define CORRECTO 0
16 #define OPCION 1
17 #define OPCION_OUTPUT_FILE 2
18 #define OPCION_GUION 3
19
20 #define DOS_ARGUMENTOS 2
21 #define TRES_ARGUMENTOS 3
22 #define CINCO_ARGUMENTOS 5
23 #define TODOS_LOS_ARGUMENTOS 6
24
25 #define UNA_POSICION 1
26
27 #define ALGUN_PROBLEMA 0
28
29 //Pre: Se ingresa el arreglo y su cantidad maxima de elementos.
30 //Pos: Los valores distintos a cero en el arreglo seran los numeros
    primos buscados.
31 void encontrarNumerosPrimos(int tope, int arreglo[]);
32
33
34 //Pre: Se ingresa el arreglo.
35 //Pos: Se imprimen por pantalla el o los resultados.
36 void imprimirPorPantalla(int resultado[]);
37
38 //Pre: Se ingresa el arreglo.
39 //Pos: Se crea un archivo de texto en el cual se guardaran los
    resultados.
40 int generarArchivo(int arreglo[]);
```

```

41
42 //Imprime por pantalla las instrucciones de uso
43 void mostrarAyuda();
44
45 //Imprime por pantalla la version
46 void mostrarVersion();
47
48 //Verifica que los argumentos sean -h, -V, -o N, - N;
49 //donde N es un entero mayor que 2 y menor que el maximo (yo puse
    4096).
50 //Devuelve: 1 si el entero se escapa del rango predeterminado.
51 //      2 si no se ingresaron comando validos.
52 //      0 si todos los argumentos son validos y se puede proceder.
53 int validarArgumentos(int argc, char **argv);
54
55 //Realizara alguna de las acciones:
56 // -h muestra el menu de ayuda.
57 // -V muestra la version.
58 // -o muestra por pantalla y crea un archivo.
59 // - muestra por pantalla.
60 int realizarAccion(int argc, char **argv);

```

commonfunc.c

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include "commonfunc.h"
5 #include "mathfunc.h"
6
7 void imprimirPorPantalla(int resultado[]) {
8     printf("%d\n", resultado[PRIMERA_POSICION]);
9     if (resultado[SEGUNDA_POSICION] != CAMPO_VACIO) {
10         printf("%d\n", resultado[SEGUNDA_POSICION]);
11     }
12 }
13
14 int generarArchivo(int arreglo[]) {
15     FILE *archivo;
16     archivo = fopen("out.txt", "w");
17     if(archivo != ALGUN_PROBLEMA){
18         int i;
19         for (i = INICIO_ARRAY; i <= 1; i++){
20             if (arreglo[i] != CAMPO_VACIO){
21                 fprintf ( archivo, "%d\n", arreglo[i]);
22             }
23         }
24         fclose(archivo);
25         return TODO_OK;
26     } else {
27         return ERROR_CREACION_DE_ARCHIVO;
28     }
29     return TODO_OK;
30 }
31
32 void mostrarVersion() {
33     printf("Common version 1.0 \n");
34 }
35
36 void mostrarAyuda() {
37     printf("Usage: \n");

```

```

38 printf("    common -h\n");
39 printf("    common -V\n");
40 printf("    common [options] M N\n");
41 printf("Options:\n");
42 printf("    -h,    --help    Prints usage information.\n");
43 printf("    -V,    --version Prints version information.\n");
44 printf("    -o,    --output  Path to output file.\n");
45 printf("    -d     --divisor Just the divisor\n");
46 printf("    -m     --multiple Just the multiple\n");
47 printf("Examples:\n");
48 printf("    common -o - 256 192\n");
49 }
50
51 //Se verifica que los parámetros ingresados sigan el formato
52 //predeterminado
53 int validarArgumentos(int argc, char **argv){
54     if ((argc > TODOS_LOS_ARGUMENTOS) || (argc < DOS_ARGUMENTOS)){
55         return ERROR_COMANDO_INVALIDO;
56     }
57     if (argc != TRES_ARGUMENTOS){
58         if (strcmp(argv[OPCION], "-h") != CORRECTO){
59             if (strcmp(argv[OPCION], "-V") != CORRECTO){
60                 if (argc > TRES_ARGUMENTOS){
61                     if (((strcmp(argv[OPCION], "-d") != CORRECTO) &&
62                         (strcmp(argv[OPCION], "-m") != CORRECTO) &&
63                         (strcmp(argv[OPCION], "-o") != CORRECTO)) ||
64                         ((strcmp(argv[OPCION_OUTPUT_FILE], "-o") != CORRECTO))
65                 ){
66                     return ERROR_COMANDO_INVALIDO;
67                 }
68                 else {
69                     if ((atoi(argv[argc-1]) > MAXIMO) ||
70                         (atoi(argv[argc-1]) < MINIMO)) {
71                         return ERROR_FUERA_DE_RANGO;
72                     }
73                     if ((atoi(argv[argc-2]) > MAXIMO) ||
74                         (atoi(argv[argc-2]) < MINIMO)) {
75                         return ERROR_FUERA_DE_RANGO;
76                     }
77                 }
78             }
79         }
80         else {
81             if (argc > DOS_ARGUMENTOS) {
82                 return ERROR_COMANDO_INVALIDO;
83             }
84         }
85     }
86     else {
87         if (argc > DOS_ARGUMENTOS) {
88             return ERROR_COMANDO_INVALIDO;
89         }
90     }
91 }
92 else {
93     if ((atoi(argv[argc-1]) > MAXIMO) || (atoi(argv[argc-1]) <
94         MINIMO)){
95         return ERROR_FUERA_DE_RANGO;
96     }
97 }

```

```

97     if ((atoi(argv[argc-2]) > MAXIMO) || (atoi(argv[argc-2]) <
MINIMO)){
98         return ERROR_FUERA_DE_RANGO;
99     }
100 }
101 return TODO_OK;
102 }
103
104
105 //Se realiza la accion previamente validada, ya sea mostrar la
version, la ayuda, mcm o mcd.
106 int realizarAccion(int argc, char **argv){
107
108     int mensajeDeError = TODO_OK;
109     int resultado[DOS_ARGUMENTOS];
110     resultado[PRIMERA_POSICION] = CAMPO_VACIO;
111     resultado[SEGUNDA_POSICION] = CAMPO_VACIO;
112
113     if (strcmp(argv[OPCION], "-h") == CORRECTO){
114         mostrarAyuda();
115     }
116     else{
117         if (strcmp(argv[OPCION], "-v") == CORRECTO){
118             mostrarVersion();
119         }
120         else{
121             int numeroBajo = atoi(argv[argc-2]);
122             int numeroAlto = atoi(argv[argc-1]);
123             if (strcmp(argv[OPCION], "-d") == CORRECTO){
124                 resultado[PRIMERA_POSICION] = mcd(numeroBajo, numeroAlto);
125             }
126             else{
127                 if (strcmp(argv[OPCION], "-m") == CORRECTO){
128                     resultado[PRIMERA_POSICION] = mcm(numeroBajo, numeroAlto)
;
129                 }
130                 else {
131
132                     resultado[PRIMERA_POSICION] = mcd(numeroBajo, numeroAlto)
;
133                     resultado[SEGUNDA_POSICION] = mcm(numeroBajo, numeroAlto)
;
134                 }
135             }
136             mensajeDeError = generarArchivo(resultado);
137             if (((argc > TRES_ARGUMENTOS) && (strcmp(argv[OPCION_GUION],
"-") == CORRECTO)) ||
138                 (argc == TRES_ARGUMENTOS)){
139                 imprimirPorPantalla(resultado);
140             }
141         }
142     }
143     return mensajeDeError;
144 }

```

mathfunc.h

```

1 //Pre: Se ingresan dos numeros.
2 //Pos: Se retorna el maximo comun divisor.
3 int mcd(int numeroBajo, int numeroAlto);
4

```

```

5
6 //Pre: Se ingresan dos numeros.
7 //Pos: Se retorna el minimo comun multiplo.
8 int mcm(int numeroBajo, int numeroAlto);

```

mathfunc.c

```

1 #include "mathfunc.h"
2
3 //Halla el maximo comun divisor entre los valores pasados por
  parametro.
4 int mcd(int numeroBajo, int numeroAlto){
5     int auxiliar;
6     while (numeroBajo > 0){
7         auxiliar = numeroBajo;
8         numeroBajo = numeroAlto % numeroBajo;
9         numeroAlto = auxiliar;
10    }
11    return numeroAlto;
12 }
13
14 //Halla el minimo comun multiplo entre los valores pasados por
  parametro.
15 int mcm(int numeroBajo, int numeroAlto){
16     int numerador = numeroBajo * numeroAlto;
17     int mcdDenominador = mcd(numeroBajo, numeroAlto);
18     int resultadoMcm = numerador / mcdDenominador;
19     return resultadoMcm;
20 }

```

8. El código Assembler:

```
1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3
4 #####FUNCION HOJA MCD#####
5
6 #STATICS VAR DEFINITIONS FUNCTION MCD
7 #define SF_SIZE_MCD 16
8 #define SF_MCD_FP_POS 12
9 #define SF_MCD_GP_POS 8
10
11 .text
12 .abicalls
13 .align 2
14 .globl mcd
15 .ent mcd
16
17
18 mcd:
19     #####INICIALIZACION DEL STACK FRAME DE LA FUNCION MCD###
20     .frame $fp, SF_SIZE_MCD, ra
21     .set noreorder
22     .cpload t9
23     .set reorder
24     subu sp, sp, SF_SIZE_MCD
25     sw $fp, SF_MCD_FP_POS(sp)
26     .cprestore SF_MCD_GP_POS
27     move $fp, sp
28     #####FIN INICIALIZACION DEL STACK FRAME DE LA FUNCION MCD###
29     #guardo los argumentos en la aba de la func que me llamo
30     sw a0, 16($fp)
31     sw a1, 20($fp)
32     #Si a0 > 0, ingresa al loop
33     bgtz a0, iterando_mcd
34     #Asigno en v0 el valor del parametro numeroAlto
35     move v0, a1
36     #Si a0 <=0, salta a fin
37     j fin_mcd
38
39 iterando_mcd:
40     #guardo en la posicion 0 del stack frame el valor auxiliar
41     sw a0, 0($fp)
42     #Calculo el resto de la division numeroAlto/numeroBajo
43     div a1, a0
44     #Guardo el valor del modulo en numeroBajo
45     mfhi t0
46     #Obtengo el valor auxiliar, guardado en la posicion 0 del stack
47     lw t1, 0($fp)
48     #El resto de la division lo guardo en a0
49     move a0, t0
50     #Ahora el numeroAlto es el valor auxiliar
51     move a1, t1
52     #Si a0 es mayor a 0, sigue en el loop. Caso contrario va a
53     fin_mcd
54     bgtz a0, iterando_mcd
55
56 fin_mcd:
57     #El numero alto es lo que retorna la funcion
58     move v0, a1
59     #####RESTAURO LOS REGISTROS###
60     move sp, $fp
```

```

60      lw      gp, SF_MCD_GP_POS(sp)
61      lw      $fp, SF_MCD_FP_POS(sp)
62      ###DESTRUYO EL STACK FRAME###
63      addu     sp, sp, SF_SIZE_MCD
64      ###RETORNO A LA SIGUIENTE INSTRUCCION DE LA FUNC QUE LLAMO A
        CMD###
65      jr      ra
66      .end          mcd
67
68      #####FIN FUNCION HOJA MCD#####
69
70      #####FUNCION NO HOJA MCM#####
71
72      #STATICS VAR DEFINITIONS FUNCTION MCM
73      #define SF_MCM_A1_POS      52
74      #define SF_MCM_A0_POS      48
75      #define SF_SIZE_MCM      48
76      #define SF_MCM_FP_POS      40
77      #define SF_MCM_GP_POS      36
78      #define SF_MCM_RA_POS      32
79
80      .text
81      .abicalls
82      .align 2
83      .globl mcm
84      .ent mcm
85
86      mcm:
87      ###INICIALIZACION DEL STACK FRAME DE LA FUNCION MCM###
88      .frame      $fp, SF_SIZE_MCM, ra
89      .set        noreorder
90      .cload      t9
91      .set        reorder
92      subu        sp, sp, SF_SIZE_MCM
93      sw          ra, SF_MCM_RA_POS(sp)
94      sw          $fp, SF_MCM_FP_POS(sp)
95      .cprestore  SF_MCM_GP_POS
96      move        $fp, sp
97      ###FIN INICIALIZACION DEL STACK FRAME DE LA FUNCION MCM###
98
99      #guardo en la aba de la func que me llamo los argumentos
100     sw          a0, SF_MCM_A0_POS($fp)
101     sw          a1, SF_MCM_A1_POS($fp)
102     #multiplico numeroBajo*numeroAlto = numerador
103     mult        a0, a1
104     #guardo en t0 el resultado de la multiplicacion anterior (
        numerador)
105     mflo        t0
106     #guardo en el stack el numerador
107     sw          t0, 16($fp)
108     #llamo a la funcion mcd con los parametros a0 y a1
109     jal         mcd
110
111     sigue:
112     #guardo en el stack el valor de mcd, valor que retorno la
        funcion (es el denominador)
113     sw          v0, 20($fp)
114     #recupero del stack el valor de numeroBajo*numeroAlto
115     lw          t0, 16($fp)
116     #Hago (numeroBajo*numeroAlto)/(mcd(numeroBajo, numeroAlto))
117     div         t0, v0
118     mflo        t1

```



```

119      #Guardo en V1 el resultado de mcm
120      move      v0,t1
121      #Guardo en el stack el resultado de mcm
122      sw        t1,24($fp)
123
124      ###FIN MCM###
125      ###RESTAURO LOS REGISTROS###
126      move      sp, $fp
127      lw        ra, SF_MCM_RA_POS(sp)
128      lw        gp, SF_MCM_GP_POS(sp)
129      lw        $fp, SF_MCM_FP_POS(sp)
130      ###DESTRUYO EL STACK FRAME###
131      addu      sp, sp, SF_SIZE_MCM
132      ###REITORNO A LA SIGUIENTE INSTRUCCION DE LA FUNC QUE LLAMO A
          CMD###
133      jr        ra
134      .end      mcm

```

../../mathfunc.S

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 ABI¹, escribiendo un programa portable que resuelva el problema descrito 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 T_EX/ L^AT_EX.

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].

¹Application 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 $mcm(m, n) = \frac{m \cdot n}{mcd(m, n)}$, 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 digital².

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.

²No 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