Trabajo práctico 1: conjunto de instrucciones MIPS

Augusto Arturi (#97498)
turitoh@gmail.com

Matias Rozanec (#97404)
rozanecm@gmail.com

Agustin Miguel Payaslian (#96885)
payas17@hotmail.com

Grupo Nro. # - 2do. Cuatrimestre de 2017

66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

5/10/2017

Resumen

El objetivo del siguiente trabajo es familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI, escribiendo un programa portable que resuelva el problema de minimo común múltiplo (mcm) y el máximo común divisor (mcd) entre dos números.

1. Introducción

Aquí se comenta en forma escueta como está constituido el presente informe, donde básicamente se encuentran dos secciones principales: Desarrollo y Conclusiones.
En Desarrollo se encuentran breves comentarios sobre la implementacion del algoritmo como tambien, las corridas de prueba del programa. En la seccion conclusiones se discuten los resultados obtenidos.

2. Desarrollo

2.1. Implementacion

A la hora de desarrollar el programa se utilizo el criterio establecido por la consigna del practico, por un lado realizar una API donde gran parte del programa es implementado en lenguaje C mientras que las funciones *mcd(m, n)* y *mcm(m, n)* son realizadas en assembly con e fin de proveer soporte especifico en nuestra plataforma principal de desarrollo, NetBSD/pmax. A la hora de brindar portabilidad entre sistemas operativos, tambien se debe realizar otra version la cual en su totalidad este desarrollada en C.

2.2. API

El algoritmo utilizado es el de Euclides, la estrategia de la cual se partio para realizar la implentacion en assembly fue realizar todas las validaciones fuera de las funciones *mcd(m, n)* y *mcm(m, n)*, utilizando estas dos unicamente para efectuar las operaciones necesarias con el fin de obtener los resultados deseados. Cabe destacar que primero se realizo la version portable, es decir aquella que en su totalidad fue hecha en C, luego se procedio haciendo la traduccion de la misma en asm.

La función mcd(m,n) fue realizada de manera recursiva.

Durante el desarrollo de la misma se respeta la ABI, los argumentos almacenados por el callee son almacenados en el argument area del stack.

2.3. Compilacion

```
root@:~# gcc -Wall -std=c99 -lm main.c my_assembly.S -o common
```

2.4. Corridas de prueba

A continuacion se demuestran las corridas de prueba con todos los casos posibles.

```
root@:~# ./common -h
Usage:
        common -h
        common -v
        common [options] M N
Options:
        -h, --help      Print 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

root@:~# ./common --version
API Version 1.0
root@:~# ./common -m -o 256 192
768
root@:~# ./common -d -o 256 192
64
root@:~# ./common -o EJEMPLO 256 192
```

En esta ultima corrida se guarda en un archivo (EJEMPLO.txt) los resultados de mcm y mcd.

A continuacion los ejemplos pedidos

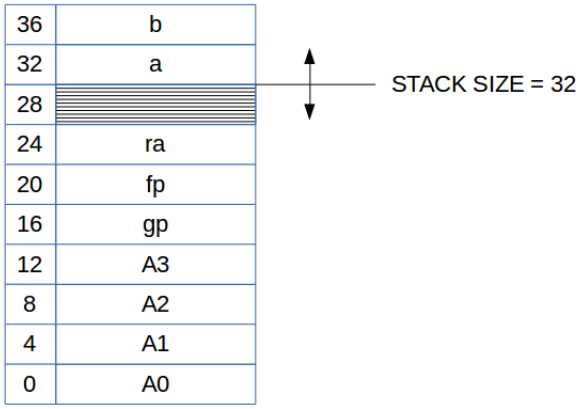
```
root@:~# ./common 5 10
10
5
root@:~# ./common 256 192
768
64
root@:~# ./common 1111 1294
1437634
1
```

Aqui ejemplos con datos de entrada incorrectos, los primeros dos por rango invalido y el ultimo por ingresar un valor negativo.

```
root@:~# ./common 111111111111 123456
strtol: Result too large or too small
root@:~# ./common 0 10
root@:~# ./common -2 10
common: unknown option -- 2
```

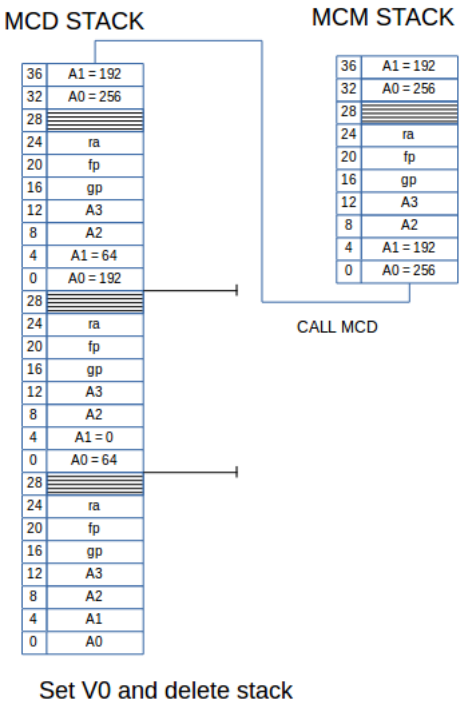
2.5. Diagrama de stack

Se muestra el stack creado, el mismo es identico para las funciones mcm o mcd, ya que las dos son **no hoja**



A continuacion un caso en el cual se hace un seguimiento del stack para el siguiente comando:

./common 256 198



3. Conclusiones

En este trabajo practico se incorporaron nuevos conceptos de importante valor tanto para el curso de la materia como para la carrera en si. Por un lado se llevo a la practica la programacion con el conjunto de instrucciones de MIPS32, lo cual a la hora de hacerlo en una computadora conlleva una dificultad extra, la cual es el ensamblado y compilacion, verificar todas aquellas advertencias obtenidas y corregirlas, ya que algunas de estas nos permitieron lograr los resultados esperados a la hora de correr el ejecutable. Ademas, la implentacion se realizo teniendo en cuenta la ABI propuesta, la cual se tuvo en cuenta en todo momento a lo largo del practico.

Un factor que se considera de gran valor fue el de realizar el diagrama de stack, al ser recursiva la funcion se observa como éste aumenta su tamaño con gran facilidad dejando explicito lo explicado en clase con otros ejemplos como el caso de un factorial recursivo. Se considera que el seguimiento del stack para el ejemplo dado ayuda a la comprension tanto de la ABI como del manejo de memoria.

Finalmente se considera que una vez realizado este trabajo se logra comprender como funciona realmente una computadora a bajo nivel y que dificultades implica a la hora de implementar una solucion en lenguaje ensamblador, se analizo el codigo generado por gcc sin optimizaciones (-O0) y se observa que el manejo de memoria es distinto al realizado, generando este instrucciones innecesarias de load y store, al evitarlas por nuestra propia cuenta estamos optimizando el codigo, tambien se intento ver una comparacion de tiempos de la version portable vs API, sin embargo los resultados no fueron los esperados ya que ambas se ejecutaron tan rapido que el clock no observa diferencia entre estas.

4. Apendice A, codigo fuente

4.1. Version 1, portable

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <limits.h>
#include <errno.h>

/* define max input. Number is max supported for 32 bit unsigned int */
#define MAX_INPUT UINT_MAX

void print_help_data(){
    printf("Usage:\n "
                    "\tcommon -h \n"
                    "\tcommon -v \n"
                    "\tcommon [options] M N \n"
                    "Options:\n"
                    "\t-h, --help \tPrint usage information.\n"
                    "\t-V, --version \tPrints version information.\n"
                    "\t-o, --output\tPath to output file.\n"
                    "\t-d  --divisor\tJust the divisor\n"
                    "\t-m  --multiple\tJust the multiple\n"
                    "Examples:\n"
```

```
                    "\tcommon -o - 256 192\n");
}

void print_version_info(){
    printf("Version 1.0\n");
}

int mcd(int a, int b) {
    /* Precondition: a > b */

    if (b == 0){
        return a;
    } else {
        return mcd(b, a % b);
    }
}

int mcm(const int a, const int b){
    return abs(a * b) / mcd(a, b);
}

short int check_range(unsigned long int a, unsigned long int b){
    /* This function tests the input numbers to be in range [2, MAX_INPUT].
     * Return values:
     *      0: Success.
     *      1: Failure.
     * */
    if ((a >= 2) && (a <= MAX_INPUT) && (b >= 2) && (b <= MAX_INPUT)){
        /* Success */
        return 0;
    }

    /* failure */
    return 1;
}

void process_input_numbers(unsigned long int *a, unsigned long int *b,
                           char *input1, char *input2){
    /* Process input numbers */
    errno = 0;
    char* endptr = NULL;
    *a = strtoul(input1, &endptr, 10);
    /* error checking performed according to strtol man page */
    if ((errno == ERANGE && (*a == LONG_MAX || *a == LONG_MIN))
        || (errno != 0 && *a == 0)) {
        perror("strtol");
        exit(EXIT_FAILURE);
    }

    *b = strtoul(input2, &endptr, 10);
    /* error checking performed according to strtol man page */
    if ((errno == ERANGE && (*b == LONG_MAX || *b == LONG_MIN))
        || (errno != 0 && *b == 0)) {
        perror("strtol");
        exit(EXIT_FAILURE);
    }

    /* make sure a > b
     * although this is not strictly necessary, we make it this way so it
       is
     * easier to follow */
    if (a < b){
```

```
        unsigned long int aux = *a;
        *a = *b;
        *b = aux;
    }

    if (check_range(*a, *b) == 1){
        exit(EXIT_FAILURE);
    }
}

void print_mcm(unsigned int a, unsigned int b, char* path){
    /* According to ASCII table, 45 is the '-' char.
     * Source: http://www.asciitable.com/ */
    if (*path == 45){
        /* output to stdout */
        printf("%i\n", mcm(a, b));
    }else{
        FILE* file = fopen(path, "a");
        /* error checking */
        if (file == NULL){
            exit(EXIT_FAILURE);
        }

        fprintf(file, "%i\n", mcm(a, b));

        fclose(file);
    }
}

void print_mcd(unsigned int a, unsigned int b, char* path){
    /* According to ASCII table, 45 is the '-' char.
     * Source: http://www.asciitable.com/ */
    if (*path == 45){
        /* output to stdout */
        printf("%i\n", mcd(a, b));
    }else{
        FILE* file = fopen(path, "a");
        /* error checking */
        if (file == NULL){
            exit(EXIT_FAILURE);
        }

        fprintf(file, "%i\n", mcd(a, b));

        fclose(file);
    }
}

int main(int argc, char* argv[]){
    unsigned long int a, b;
    static struct option long_options[] = {
            {"help",        no_argument,                0, 'h' },
            {"version",     no_argument,                0, 'v' },
            {"output",      required_argument,  0, 'o' },
            {"divisor",     no_argument,            0, 'd' },
            {"multiple",        no_argument,            0, 'm' },
            {0,                     0,                                  0,
                 0  }
    };
    int option_index = 0;
    int c = getopt_long(argc, argv, "hvo:dm", long_options, &option_index);
    switch (c){
```

```
        case 'h':
            print_help_data();
            break;
        case 'v':
            print_version_info();
            break;
        case 'o':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, optarg);
            print_mcd(a, b, optarg);
            break;
        case 'd':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcd(a, b, "-");
            break;
        case 'm':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, "-");
            break;
        default:
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, "-");
            print_mcd(a, b, "-");
    }

    exit(EXIT_SUCCESS);
}
```

4.2. Version 2, API

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <errno.h>
#include "my_assembly.h"

/* define max input. Number is max supported for 32 bit unsigned int */
#define MAX_INPUT 2147483647
#define LONG_MAX 4294967295

void print_help_data(){
    printf("Usage:\n "
                    "\tcommon -h \n"
                    "\tcommon -v \n"
                    "\tcommon [options] M N \n"
                    "Options:\n"
                    "\t-h, --help \tPrint usage information.\n"
                    "\t-V, --version \tPrints version information.\n"
                    "\t-o, --output\tPath to output file.\n"
                    "\t-d  --divisor\tJust the divisor\n"
                    "\t-m  --multiple\tJust the multiple\n"
                    "Examples:\n"
                    "\tcommon -o - 256 192\n");
}

void print_version_info(){
    printf("API Version 1.0\n");
}
```

```c
short int check_range(unsigned long int a, unsigned long int b){
    /* This function tests the input numbers to be in range [2, MAX_INPUT].
     * Return values:
     *      0: Success.
     *      1: Failure.
     * */
    if ((a >= 2) && (a <= MAX_INPUT) && (b >= 2) && (b <= MAX_INPUT)){
        /* Success */
        return 0;
    }

    /* failure */
    return 1;
}

void process_input_numbers(unsigned long int *a, unsigned long int *b,
                           char *input1, char *input2){
    /* Process input numbers */
    errno = 0;
    char* endptr = NULL;
    *a = strtoul(input1, &endptr, 10);
    /* error checking performed according to strtol man page */
    if ((errno == ERANGE && (*a == LONG_MAX))
        || (errno != 0 && *a == 0)) {
        perror("strtol");
        exit(EXIT_FAILURE);
    }

    *b = strtoul(input2, &endptr, 10);
    /* error checking performed according to strtol man page */
    if ((errno == ERANGE && (*b == LONG_MAX ))
        || (errno != 0 && *b == 0)) {
        perror("strtol");
        exit(EXIT_FAILURE);
    }

    /* make sure a > b
     * although this is not strictly necessary, we make it this way so it
       is
     * easier to follow */
    if (a < b){
        unsigned long int aux = *a;
        *a = *b;
        *b = aux;
    }

    if (check_range(*a, *b) == 1){
        exit(EXIT_FAILURE);
    }
}

void print_mcm(unsigned int a, unsigned int b, char* path){
    /* According to ASCII table, 45 is the '-' char.
     * Source: http://www.asciitable.com/ */
    if (*path == 45){
        /* output to stdout */
        printf("%i\n", mcm(a, b));
    }else{
        FILE* file = fopen(path, "a");
        /* error checking */
        if (file == NULL){
```

```
            exit(EXIT_FAILURE);
        }

        fprintf(file, "%i\n", mcm(a, b));

        fclose(file);
    }
}

void print_mcd(unsigned int a, unsigned int b, char* path){
    /* According to ASCII table, 45 is the '-' char.
     * Source: http://www.asciitable.com/ */
    if (*path == 45){
        /* output to stdout */
        printf("%i\n", mcd(a, b));
    }else{
        FILE* file = fopen(path, "a");
        /* error checking */
        if (file == NULL){
            exit(EXIT_FAILURE);
        }

        fprintf(file, "%i\n", mcd(a, b));

        fclose(file);
    }
}

int main(int argc, char* argv[]){
    unsigned long int a, b;
    static struct option long_options[] = {
            {"help",        no_argument,                0, 'h' },
            {"version",     no_argument,                0, 'v' },
            {"output",      required_argument,  0, 'o' },
            {"divisor",     no_argument,            0, 'd' },
            {"multiple",        no_argument,            0, 'm' },
            {0,                     0,                                  0,
                 0  }
    };
    int option_index = 0;
    int c = getopt_long(argc, argv, "hvo:dm", long_options, &option_index);
    switch (c){
        case 'h':
            print_help_data();
            break;
        case 'v':
            print_version_info();
            break;
        case 'o':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, optarg);
            print_mcd(a, b, optarg);
            break;
        case 'd':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcd(a, b, "-");
            break;
        case 'm':
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, "-");
            break;
        default:
```

```
            process_input_numbers(&a, &b, argv[argc-2], argv[argc-1]);
            print_mcm(a, b, "-");
            print_mcd(a, b, "-");
    }

    exit(EXIT_SUCCESS);
}
```

my_assembly.h

```
int mcd(int a, int b);

int mcm(const int a, const int b);
```

my_assembly.S

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

.globl mcd
.globl mcm


.text
.abicalls
.align 2
.ent mcm


mcm:

    #define mcm_frame_size 32
    #define mcm_frame_ra 24
    #define mcm_frame_fp 20
    #define mcm_frame_gp 16
    #define mcm_frame_2arg 36
    #define mcm_frame_1arg 32

    .frame $fp, mcm_frame_size, ra
    /*CREATE STACK FRAME*/
    subu sp, sp, mcm_frame_size
    .cprestore 16
    sw gp, mcm_frame_gp(sp)
    sw $fp, mcm_frame_fp(sp)
    sw ra, mcm_frame_ra(sp)
    move  $fp, sp /* frame pointer at the bottom */

    sw a0, mcm_frame_1arg($fp) /*save first argument (int a)*/
    sw a1, mcm_frame_2arg($fp) /*save second argument (int b)*/

    lw t3, mcm_frame_1arg($fp) /* use temporary for a0 value */
    lw t4, mcm_frame_2arg($fp) /* use temporary for a1 value */

    mul t3, t3, t4 /* (a * b) */

    la t5, mcd /* load address of mcd function */
    jalr t5     /* jump and link to mcd */

    div t3, t3, v0 /* (a * b) / mcd(a, b); */

    move v0, t3 /* set result */


```

```
47      /* PREPARE TO DELETE STACK */
48      lw gp, mcm_frame_gp(sp)
49      lw $fp, mcm_frame_fp(sp)
50      lw ra, mcm_frame_ra(sp)
51      addu sp, sp, mcm_frame_size
52      jr ra
53
54  .end mcm
55
56  .text
57  .abicalls
58  .align 2
59  .ent mcd
60
61  mcd:
62
63      #define mcd_frame_size 32
64      #define mcd_frame_ra 24
65      #define mcd_frame_fp 20
66      #define mcd_frame_gp 16
67      #define mcd_frame_2arg 36
68      #define mcd_frame_1arg 32
69
70      .frame $fp, mcd_frame_size, ra
71      /*CREATE STACK*/
72      subu sp, sp, mcd_frame_size
73      .cprestore 16
74      sw gp, mcd_frame_gp(sp)
75      sw $fp, mcd_frame_fp(sp)
76      sw ra, mcd_frame_ra(sp)
77      move  $fp, sp /* From here, we use frame pointer */
78
79      sw a0, mcd_frame_1arg($fp) /*save first argument (int a)*/
80      sw a1, mcd_frame_2arg($fp) /*save second argument (int b)*/
81
82      beq a1, zero, return_a /*if b == 0 then return a*/
83
84      lw t0, mcd_frame_2arg($fp) /* use temporary for a1 value */
85      lw t1, mcd_frame_1arg($fp) /* use temporary for a0 value */
86
87      move a0, t0 /* use b as first argument */
88      remu a1, t1, t0 /* a %b, used as second argument*/
89
90      jal mcd
91
92  return_a:
93      move v0, a0
94
95      /* PREPARE TO DELETE STACK */
96      lw gp, mcd_frame_gp(sp)
97      lw $fp, mcd_frame_fp(sp)
98      lw ra, mcd_frame_ra(sp)
99      addu sp, sp, mcd_frame_size
100     jr ra
101
102
103 .end mcd
```

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.