Trabajo Práctico Nº 1: Conjunto de instrucciones MIPS

Martinez Ariel, *Padrón Nro. 88573*
arielcorreofiuba@gmail.com.ar

Nestor Huallpa, *Padrón Nro. 88614*
huallpa.nestor@gmail.com

Pablo Sivori, *Padrón Nro. 84026]*
sivori.daniel@gmail.com

Entrega: 16/05/2017

1er. Cuatrimestre de 2017
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

En el presente trabajo práctico se describirán todos los pasos y conclusiones relacionadas al desarrollo e implementación de la codificacion y decodificacion de datos formateados en base 64.

Índice

1. Introducción

El objetivo del presente trabajo práctico es implementar las funciones encode y decode de datos base 64 en código assembly. Para ello nos conectamos con el emulador gxemul para realizar la codificación en mips 32 y posteriormente poder realizar las pruebas pertinentes.

2. Implementación

2.1. Lenguaje

Como lenguaje de implementación se eligió ANSI C ya que el mismo permite una alta portabilidad entre diferentes plataformas. El desarrollo del programa se realizó usando un editor de texto (gedit,vim, kwrite) y compilando los archivos fuente con GCC que viene en linux. Para compilar, ejecutar el siguiente comando:

$ make

2.2. Descripción del programa

La función `main` se encuentra en `tp0.c` y se encarga de interpretar las opciones y argumentos. En caso de ser una opción, como ayuda o versión, se imprime el mensaje correspondiente y finaliza la ejecución. Cuando no es una opción de ayuda o versión, se procede a procesar los datos de entrada. La salida de estas funciones proveen un codigo de error que sirve como salida del programa. Los mensajes de versión y ayuda se imprimen por stdout y el programa finaliza devolviendo 0 (cero) al sistema. Los mensajes de error se imprimen por la salida de errores (stderr) y el programa finaliza devolviendo 1 (uno) al sistema.

2.2.1. Errores posibles

1. El procesamiento de la entrada estándar causó el agotamiento del heap.
2. La invocación del programa es incorrecta.
3. Alguno de los archivos es inexistente.
4. Se produjo un error en la lectura del archivo a decodificar.
5. Se produjo un error en la escritura del archivo, donde se encuentra el resultado de la decodificación.

Cuando se produce un error en la codificación o decodificación, se devuelve un código distinto de 0 el cual sirve como indice para ver la descripción del error, la cual se encuentra en el vector de errores msgerr. En caso de que se devuelva 0, en esta posición el vector msgerr contendrá el mensaje "No hubo errores".

2.3. Desarrollo de actividades

1. Se instaló en un linux un repositorio de fuentes (GIT) para que al dividir las tareas del TP se pudiese hacer una unión de los cambios ingresados por cada uno de los integrantes más fácilmente.

2. Cada persona del grupo se comprometió a que sus cambios en el código fuente y los cambios obtenidos del repositorio que pudiesen haber subido los otros integrantes del grupo, sean compilados los sistemas operativos Linux y el NetBSD, asegurando así portabilidad entre plataformas planteada en el enunciado.

3. Se estableció que todos los integrantes en mayor o menor medida, contribuyan en el desarrollo de todas las partes del código para que nadie quede en desconocimiento de lo que se hizo en cada sección. Una parte se dedico al desarrollo de la función encode, mientras que la otra parte se focalizó en el desarrollo de la función decode. De esta manera los integrantes del grupo realizaron la programación en codigo assembly de ambas funciones familiarizandose de esta manera con el lenguaje.

4. Se propuso antes de iniciar el desarrollo de cada función, diagramar los stacks frames de las funciones que utilizan el encode y decode, respetando la convención de la abi dada por la catedra a través de la bibliografía brindada por la misma.

5. Solo se desarrollo el código assembly de las funciones encode y decode, lo cual para poder generar el resto del codigo assembly a partir del código fuente, dentro de NETBSD, se utilizó gcc con la siguiente opción:

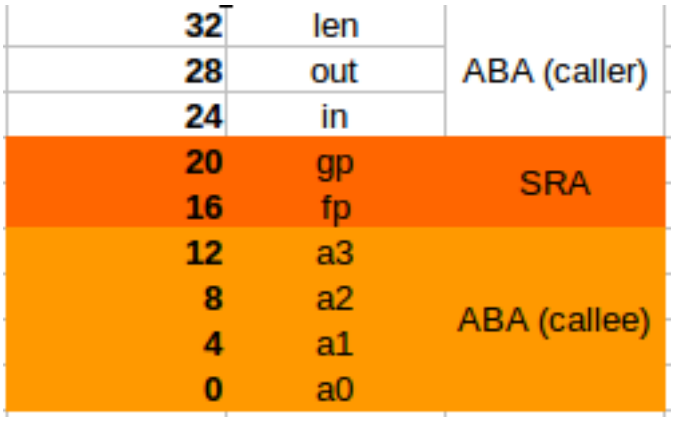
   gcc -S main.c

6. Para crear el presente informe se debe utilizar el comando make en el directorio informe.

2.4. Stack Frame de funciones

Función Encode

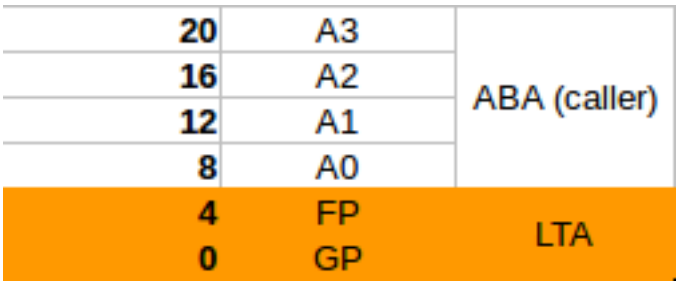
Para la función encode, se utilizo la función bloqueToBase64 cuyo stack frame es el siguiente:



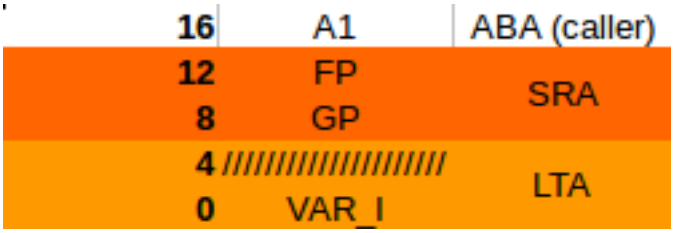
La función principal para realizar la codificación del mensaje a base 64 tiene el siguiente stack frame:

Función Decode

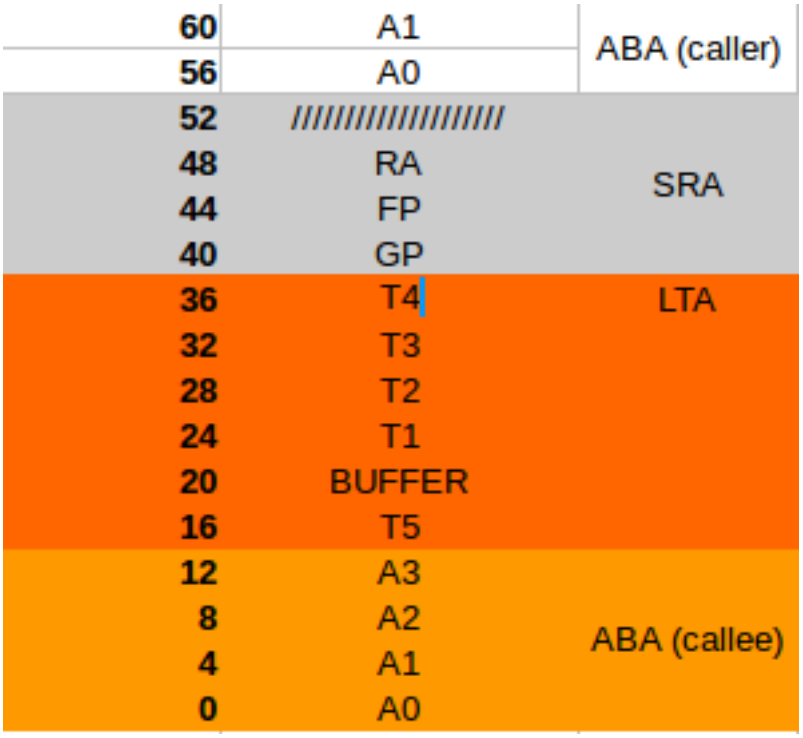
Para la función decode, se utilizo la función int search_b64(char caracter), la cual devuelve la posición en el vector de caracteres base64, del caracter codificado en base 64, utilizado para el proceso de decodificación. El stack frame utilizado para dicha función es:



También se utilizó la función read characters, la cual lee los caracteres de un archivo con codificación base 64. Su stack frame es:



Finalmente se muestra el stack frame de la función decode:



2.5. Casos de prueba

1. Codificamos **Man** ingresando por stdin y devolviendo por stdout
   TWFuCg==
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, el siguiente comando:
   echo Man — ../tp0 -a encode

2. Codificamos **Man** ingresando por stdin. Luego decodificamos por stdout obteniendo como resultado **Man**.
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, el siguiente comando:
   echo -n Man — ../tp0 — ../tp0 -a decode

3. Verificamos bit a bit, obteniendo por stdout
   0000000 x y z \n
   0000004
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, el siguiente comando:
   echo xyz — ../tp0 — ../tp0 -a decode — od -t c

4. Codificamos 1024 bytes y chequeamos que no haya más de 76 unidades de longitud.
   El resultado es una secuencia de palabras eQp5CnkK que se repiten 9 veces por linea, teniendo un total de 17 lineas.
   En la última línea, la secuencia finaliza con las palabras eQp5Cg==
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, el siguiente comando:
   yes — head -c 1024 — ../tp0 -a encode

5. Verificamos que la cantidad de bytes decodificados sea 1024.
   1024
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, el siguiente comando:
   yes — head -c 1024 — ../tp0 -a encode — ../tp0 -a decode — wc -c

6. Codificamos el contenido del archivo de **entrada.txt**, el cual contiene la palabra **Man**. Guardamos la salida en el archivo **salida.txt**, el cual contiene como resultado de la codificación **TWFuCg==**. Luego decodificamos la salida de este archivo y lo mandamos a otro archivo de entrada (**entrada2.txt**). Vemos que en este último archivo se encuentra la palabra que se había pasado como entrada en el primer archivo, verificandose de esta manera la codificación y decodificación.
   Para esta prueba ejecutamos por consola, estando posicionados en la carpeta pruebas, los siguientes comandos:
   ../tp0 -i entrada.txt -o salida.txt -a encode
   ../tp0 -i salida.txt -o entrada2.txt -a decode

7. Corremos el script **test_enunciado.sh**. El programa verifica con diff la codificación, mostrando las diferencias en caso de error. Finalmente esperamos superar la centena de corridas para cortar su ejecución haciendo ctrl+c.

3. El código fuente, en lenguaje C

```
/**
*    6620 - Organizacion del computador
*    Trabajo Practico 0
*    Alumnos:
*            88614 - Nestor Huallpa
*            88573 - Ariel Martinez
*            84026 - Pablo Sivori
*/

#include <getopt.h>
#include "base64.h"
#include <unistd.h>

static char* ENCODE = "encode";
static char* DECODE = "decode";

typedef struct {
        char* accion;
        char* entrada;
        char* salida;
} Parametro;

Parametro manejarArgumentosEntrada(int argc, char** argv)
{
        int siguiente_opcion;
        int option_index;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hva:i:o:"; /* "hva::i:o:" */

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largas[] =
        {
        { "help",       no_argument,            0,  'h'},
        { "version",    no_argument,            0,  'V'},
                { "action",     required_argument,  0,  'a'}, /*optional_argument*/
                { "input",      required_argument,  0,  'i'},
                { "output",     required_argument,  0,  'o'},
        { 0, 0, 0, 0 }
    };


        Parametro parametro;
        parametro.accion        = ENCODE;
        parametro.entrada       = "";
        parametro.salida        = "";

    while (1)
    {
        siguiente_opcion = getopt_long (argc, argv, op_cortas, op_largas, &option_index);

        if (siguiente_opcion == -1)
                break;

        switch (siguiente_opcion)
        {
            case 'h' :

                printf("Usage:\n");
                printf("\ttp0 -h\n");
                printf("\ttp0 -V\n");
                printf("\ttp0 [ options ]\n");

                printf("Options:\n");
                printf("\t-V, --version       Print version and quit.\n");
                printf("\t-h, --help          Print this information.\n");
                printf("\t-i, --input         Location of the input file.\n");
                printf("\t-o, --output        Location of the output file.\n");
                printf("\t-a, --action        Program action: encode (default) or decode.\n");

                printf("Examples:\n");
                printf("\ttp0 -a encode -i ~/input -o ~/output\n");
                printf("\ttp0 -a encode\n");
                exit(0);
                    break;

            case 'v' :
                printf("Tp0: Version 0.1: Grupo: Nestor Huallpa, Ariel Martinez, Pablo Sivori \n");
                exit(0);
                break;

            case 'a' :
                if ( optarg )
                        parametro.accion = optarg;
                break;

            case 'i' :
                if ( optarg )
                                        parametro.entrada = optarg;
                break;

            case 'o' :
                if ( optarg )
                                        parametro.salida = optarg;
                break;
```

```
            }
        }

        return parametro;

    }

    int main(int argc, char** argv) {
            Parametro p = manejarArgumentosEntrada(argc, argv);

            int isEntradaArchivo = strcmp(p.entrada,"");
            int isSalidaArchivo = strcmp(p.salida,"");
            FILE* archivoEntrada = (isEntradaArchivo!=0)?fopen(p.entrada, "rb"):stdin; //Si la entrada esta vacia lee st
            FILE* archivoSalida = (isSalidaArchivo!=0)?fopen ( p.salida, "w" ):stdout; //Si la salida esta vacia escribe

            int returnCode = 0;

            if (archivoEntrada == NULL) {
                    fprintf(stderr, "ERROR: NO EXISTE LA ENTRADA.\n");
                    exit (1);
            }

            if ( strcmp(p.accion, ENCODE) == 0 )
            {
                    /* Codificar entrada */
            int fileDescriptorEntrada = fileno(archivoEntrada);
            int fileDescriptorSalida = fileno(archivoSalida);
                    returnCode = base64_encode(fileDescriptorEntrada, fileDescriptorSalida);
                    if (returnCode!=0) fprintf(stderr, "Error: %s\n", errmsg[returnCode]);
            } else if ( strcmp(p.accion, DECODE) == 0 ) {
            int infd = fileno(archivoEntrada);
            int outfd = fileno(archivoSalida);
                    returnCode = decodificar(infd, outfd);
            if (returnCode!=0) fprintf(stderr, "Error: %s\n", errmsg[returnCode]);
            } else {
                    fprintf(stderr, "ERROR: SE DEBE INGRESAR UN ARGUMENTO CORRECTO PARA LA OPCION -i.\n");
                    exit(1);
            }

        /* Cierro los archivos de entrada y salida si no son stdin y stdout */
        if(isEntradaArchivo!=0){
                    fclose(archivoEntrada);
            }
            if(isSalidaArchivo!=0){
                    fclose  ( archivoSalida );
            }

            return returnCode;
    }
```

4. El código MIPS32 generado por el compilado

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

#########FUNCION DECODE############

#STATICS VAR DEFINITIONS FUNCTION DECODIFICAR
#define SF_SIZE_DECODIFICAR        56
#define SF_DECODIFICAR_GP_POS      40
#define SF_DECODIFICAR_FP_POS      44
#define SF_DECODIFICAR_RA_POS      48


#STATICS VAR DEFINITIONS FUNCTION READ CHARACTERS
#define SF_SIZE_READCH              16
#define SF_READCH_VAR_I             4
#define SF_READCH_GP_POS            8
#define SF_READCH_FP_POS            12

#define BUFFER_SIZE                 1


.text
.abicalls
.align 2
.globl decodificar
.ent decodificar

decodificar:
        .frame          $fp, SF_SIZE_DECODIFICAR, ra
        .set            noreorder
        .cpload         t9
        .set            reorder
        subu            sp, sp, SF_SIZE_DECODIFICAR
        sw                  ra, SF_DECODIFICAR_RA_POS(sp)
        sw                  $fp, SF_DECODIFICAR_FP_POS(sp)
    .cprestore    SF_DECODIFICAR_GP_POS
        move            $fp, sp
        sw                  a0, 56($fp)             #Guardo FD in
    sw              a1, 60($fp)             #Guardo FD out

obtener_enteros:                            #LEO LOS 4 PRIMEROS CARACTERES
        lw                  a0, 56($fp)
        lw                  a1, 60($fp)
    jal              read_characters        # Leemos 4 caracteres

    li               t0, 0
    blt              v0,t0,error_lectura    # Salto si retorno < 0, a informar error
    beqz             v0,fin_success         # Salto si es EOF
    li               t0, 4
    blt              v0,t0,err_lect_incomp  # Salto si 0< retorno < 4, a informar error

    li               t5, 3                  # Cantidad de bytes maximo por bloque de salida
    sw               t5, 16($fp)            # Esta cantidad puede ir de 1 a 3

    la               t0, buffer             # Cargo direccion de buffer con los 4 elementos leidos
    sw               t0, 20($fp)            # Guardo el buffer en stack

    lb               t0, 0(t0)              #Leo caracter primera posicion
    move             a0, t0
    jal              search_b64             #Obtengo valorEntero1
    bnez             a3,error_busqueda
    move             t1, v0                 #Resultado valorEntero 1
    sw               t1, 24($fp)

    lw               t0, 20($fp)
    lb               t0, 1(t0)              #Leo el segundo caracter
    move             a0, t0
    jal              search_b64
    bnez             a3, error_busqueda
    move             t1, v0                 #Resultado valorEntero 2
    sw                 t1, 28($fp)

    lw                 t0, 20($fp)
    lb               t0, 2(t0)              #Leo el tercer caracter

    la               t9, basis_64           # levando basis_64
    lb               t6, 64(t9)             # Obtengo el caracter '='
    beq              t0, t6, descontar_uno  # saltar si el caracter leido es =

    move             a0, t0
    jal              search_b64
    bnez             a3, error_busqueda
    move             t1, v0                 #Resultado valorEntero 3
    sw                 t1, 32($fp)
    j                leer_cuarto_byte
descontar_uno:
    lw               t5, 16($fp)            # Leo Contador LEN_OUT de bytes tranformados
    subu             t5, t5, 1              # Resto en uno a LEN_OUT de bytes tranformados
    sw               t5, 16($fp)

leer_cuarto_byte:
    lw             t0, 20($fp)
    lb               t0, 3(t0) #Leo el cuarto caracter

    la               t9, basis_64           # levando basis_64
```

```
    lb              t6, 64(t9)              # Obtengo el caracter '='
    beq             t0, t6, descontar_segundo  # saltar si el caracter leido es =

    move            a0, t0
    jal             search_b64
    bnez            a3, error_busqueda
    move            t4, v0 #Resultado valorEntero 4
    sw               t4, 36($fp)
    j               continuar_deco
descontar_segundo:
    lw              t5, 16($fp)             # Leo Contador LEN_OUT de bytes tranform
    subu            t5, t5, 1             # Resto en uno a LEN_OUT de bytes tranfo
    sw              t5, 16($fp)

continuar_deco:

    lw              t5, 16($fp)
    la              t6, buffer_write

    li              t1, 3                   # Voy a obtner 3 bytes de salida
    beq             t5,t1,continuar_deco4   # Salto pa decodificar 4 y obtner 3 byte

    li              t1, 2
    beq             t5,t1,continuar_deco3   # Salto pa decodificar 3 y obtner 2 byte

    li              t1, 1
    beq             t5,t1,continuar_deco2   # Salto pa decodificar 3 y obtner 2 byte

continuar_deco4:
    lw              t1,24($fp)
    lw              t2,28($fp)
    lw              t3,32($fp)
    lw              t4,36($fp)

    bltz            t1,fin_decodificar
    li              t5,63
    bgt             t1,t5,fin_decodificar

    li              t5,64
    bltz            t2,fin_decodificar
    bgt             t2,t5,fin_decodificar

    bltz            t3,fin_decodificar
    bgt             t3,t5,fin_decodificar

    bltz            t4,fin_decodificar
    bgt             t4,t5,fin_decodificar

    li              t5,2
    sll             s0,t1,t5 #valorEntero1<<2
    li              t5,4
    srl             s1,t2,t5 #valorEntero2>>4
    or              s0,s0,s1 #valorEntero1<<2 | valorEntero2>>4
    sb              s0, 0(t6)

    li              t5,4
    sll             s0,t2,t5 #valorEntero2<<4
    li              t5,2
    srl             s1,t3,t5 #valorEntero3>>2
    or              s0,s0,s1 #valoreEntero2<<4 | valorEntero3>>2
    sb              s0, 4(t6)

    li              t5,6
    sll             s0,t3,t5 #valorEntero3<<6
    or              s0,s0,t4 #valorEntero3<<6 | valorEntero4
    sb              s0, 8(t6)
    j               escribir_decode

continuar_deco3:
    lw              t1,24($fp)
    lw              t2,28($fp)
    lw              t3,32($fp)

    bltz            t1,fin_decodificar
    li              t5,63
    bgt             t1,t5,fin_decodificar

    li              t5,64
    bltz            t2,fin_decodificar
    bgt             t2,t5,fin_decodificar

    bltz            t3,fin_decodificar
    bgt             t3,t5,fin_decodificar

    li              t5,2
    sll             s0,t1,t5 #valorEntero1<<2
    li              t5,4
    srl             s1,t2,t5 #valorEntero2>>4
    or              s0,s0,s1 #valorEntero1<<2 | valorEntero2>>4
    sb              s0, 0(t6)

    li              t5,4
    sll             s0,t2,t5 #valorEntero2<<4
    li              t5,2
    srl             s1,t3,t5 #valorEntero3>>2
    or              s0,s0,s1 #valoreEntero2<<4 | valorEntero3>>2
    sb              s0, 4(t6)
    j                escribir_decode
```

```
continuar_deco2:
    lw              t1,24($fp)
    lw              t2,28($fp)

    bltz            t1,fin_decodificar
    li              t5,63
    bgt             t1,t5,fin_decodificar

    li              t5,64
    bltz            t2,fin_decodificar
    bgt             t2,t5,fin_decodificar

    li              t5,2
    sll             s0,t1,t5 #valorEntero1<<2
    li              t5,4
    srl             s1,t2,t5 #valorEntero2>>4
    or              s0,s0,s1 #valorEntero1<<2 | valorEntero2>>4
    sb              s0, 0(t6)

escribir_decode:
    la              t6, buffer_write            # Cargamos resultados a escribir
    lw              a0, 60($fp)                 # Vamos por el PRIMERO caracter del bloque. Leo fd_out
    move            a1, t6                      # Cargo direccion del word a escribit
    li              a2, 1
    li              v0, SYS_write
    syscall
    bnez             a3, error_escritura

escribir_caracter_dos:

    li              t6, 2
    lw              t5, 16($fp)             # Leo Contador LEN_OUT de bytes tranformados
    blt             t5, t6, fin_success
    la              t6, buffer_write            # Cargamos resultados a escribir
    lw              a0, 60($fp)                 # Vamos por el SEGUNDO caracter del bloque
    addiu           a1, t6, 4                   # Calculo direccion del siguiente word, para escritura
    li              a2, 1                       # solo imprimimos el primer byte del word
    li              v0, SYS_write
    syscall
    bnez            a3, error_escritura

escribir_caracter_tres:

    li              t6, 3
    lw              t5, 16($fp)             # Leo Contador LEN_OUT de bytes tranformados
    blt             t5, t6, fin_success

    la              t6, buffer_write            # Cargamos resultados a escribir
    lw              a0, 60($fp)                 # Vamos por el TERCER caracter del bloque
    addiu           a1, t6, 8                   # Calculo direccion del siguiente word, para escritura
    li              a2, 1                       # solo imprimimos el primer byte del word
    li              v0, SYS_write
    syscall
    bnez            a3, error_escritura
    j              obtener_enteros

fin_success:
    li              v0, 0
    j              fin_decodificar
err_lect_incomp:
    li              v0, 6
    j              fin_decodificar
error_busqueda:
    li              v0, 1 #Indice 0 donde esta el mensaje de error de busqueda del caracter en el vector de errores
    j              fin_decodificar
error_lectura:
    li              v0, 2 #Indice 1 donde esta el mensaje de error de lectura  en el vector de errores
    j               fin_decodificar
error_escritura:
    li              v0, 3 #Indice 2 donde esta el mensaje de error de escritura en el vector de errores
fin_decodificar:
    move            sp, $fp
    lw              $fp,SF_DECODIFICAR_FP_POS(sp)   #Destruimos el frame
    lw              gp,SF_DECODIFICAR_GP_POS(sp)
    lw              ra,SF_DECODIFICAR_RA_POS(sp)
    addu            sp,sp, SF_SIZE_DECODIFICAR
    jr              ra
    .end            decodificar


    .text
    .abicalls
    .align 2
    .globl read_characters
    .ent read_characters
read_characters:
     .frame         $fp, SF_SIZE_READCH, ra
     .set           noreorder
     .cpload        t9
     .set           reorder
     subu           sp, sp, SF_SIZE_READCH
     sw             $fp, SF_READCH_FP_POS(sp)       #save fp
     .cprestore     SF_READCH_GP_POS
     move             $fp, sp                     #fp->sp
     sw             a0, 16($fp)                     #save arg(infd)
     sw             zero, SF_READCH_VAR_I($fp)      # i=0
for_read_decode:
     la             t0, buffer                  # Carga la dir donde guardar
     lw             t1, SF_READCH_VAR_I($fp)        # Cargo i
```

```
    addu        t0, t0, t1                  # Obtengo dir de buffer[i]
    lw          a0, 16($fp)
    move        a1, t0
    li          a2, BUFFER_SIZE
    li          v0, SYS_read
    syscall                                 # read from file
    blez        v0, fin_lectura_deco        # Salto si es menor o igual q 0
    lw          t0, SF_READCH_VAR_I($fp)    # Cargo i
    addiu       t0, t0, 1                   # i++
    sw          t0, SF_READCH_VAR_I($fp)    #Guardo i
    li          t1, 4
    blt         t0, t1, for_read_decode     # Salto si i<4
fin_lectura_deco:
    lw          v0, SF_READCH_VAR_I($fp)    # Cargo i
    move        sp,$fp
    lw          gp, SF_READCH_GP_POS(sp)    #restore gp
    lw          $fp, SF_READCH_FP_POS(sp)   #restore fp
    addu        sp,sp, SF_SIZE_READCH
    jr          ra
    .end        read_characters


#
# Funcion busqueda de posicion de caracter en vector basis 64
#

#define SIZE_SF_SEARCH_B64 8
#define LOCATE_FP_SF_SEARCH_B64 4
#define LOCATE_ARG0_SF_SEARCH_B64 8
#define LOCATE_ARG3_SF_SEARCH_B64 20
#define RETURN_SUCCESS  0
#define RETURN_ERROR    -1

    .text
    .align  2
    .globl      search_b64
    .ent        search_b64
search_b64:
    .frame      $fp,SIZE_SF_SEARCH_B64, ra
    .set        noreorder
    .cpload     t9
    .set        reorder
    subu        sp, sp, SIZE_SF_SEARCH_B64
    .cprestore 0
    sw          $fp, LOCATE_FP_SF_SEARCH_B64(sp)
    move        $fp, sp
    sw          a0, LOCATE_ARG0_SF_SEARCH_B64($fp)

    li          v0, 0           # Usamos V0 para el resultado
    li          t2, 66          # Para saber si no esta en el array el caracter buscado
    la          t0, basis_64
search_loop:
    lb          t1,0(t0)
    beq         a0,t1,search_b64_return
    beq         v0,t2,error_return
    addi        t0,t0,1     #Avanzo de a 1 word
    addi        v0, v0, 1
    j           search_loop

search_b64_return:
    li          a3,RETURN_SUCCESS #Success
    sw          a3,20($fp)
    j           destroy_frame

error_return:
    li          a3,RETURN_ERROR                     #Error
    sw          a3,LOCATE_ARG3_SF_SEARCH_B64($fp)
    li          v0,RETURN_ERROR                     #Error. No se encontro en el array el caracter buscado

destroy_frame:
    move        sp, $fp
    lw          $fp, LOCATE_FP_SF_SEARCH_B64(sp)        # Destruimos el frame.
    lw          a0, LOCATE_ARG0_SF_SEARCH_B64(sp)
    lw          a3, LOCATE_ARG3_SF_SEARCH_B64(sp)
    lw          gp, 0(sp)
    addu        sp, sp, SIZE_SF_SEARCH_B64
    jr          ra
    .end        search_b64


##########FIN FUNCION DECODE############

##########FUNCION ENCODE############

    .text
    .align 2
    .globl bloqueToBase64
    .ent bloqueToBase64
bloqueToBase64:
    .frame      $fp, 24, ra
    .set        noreorder
    .cpload     t9
    .set        reorder
    subu        sp, sp, 24
    .cprestore  20
    sw          $fp, 16(sp)
    move        $fp, sp
    sw          a0, 24($fp)         # guardo in
```

```
    sw          a1, 28($fp)          # guardo out
    sw          a2, 32($fp)          # guardo len
    lbu         t1, 0(a0)           # Leo in[0]
    srl         t1, t1, 2           # Me quedo con los 6 bits mas significativos de in[0].
    la          t9, basis_64
    addu        t2, t9, t1          # Obtengo direccion basis_64 + indice_caracter
    lbu         t3, 0(t2)           # Lee codificacion basis_64[indice_caracter]
    sb          t3, 0(a1)           # Guardo codificacion en out[0]
    lbu         t1, 0(a0)           # Leo in[0]
    lbu         t2, 1(a0)           # Leo in[1]
    andi        t3, t1, 0x03        # (in[0] & 0x03)
    sll         t3, t3, 4           # (in[0] & 0x03) << 4
    andi        t4, t2, 0xf0        # (in[1] & 0xf0)
    srl         t4, t4, 4           # (in[1] & 0xf0) >> 4
    or          t4, t4, t3          # Concateno dos bits y seis bits extraidos

    la          t9, basis_64
    addu        t5, t9, t4          # Obtengo direccion basis_64 + indice_caracter
    lbu         t6, 0(t5)           # Lee codificacion basis_64[indice_caracter]
    sb          t6, 1(a1)           # Guardo codificacion en out[1]
    lw          t1, 32($fp)         # leo LEN
    li          t2, 1
    ble         t1, t2, menor_igual_2  # Saltar si no cumple len > 1
    lbu         t1, 1(a0)           # Leo in[1]
    lbu         t2, 2(a0)           # Leo in[2]
    andi        t1, t1, 0x0f        # Calculo (in[1] & 0x0f)
    sll         t1, t1, 2           # Calculo en t1 (in[1] & 0x0f) << 2)
    andi        t2, t2, 0xc0
    srl         t2, t2, 6           # ((in[2] & 0xc0) >> 6))
    or          t3, t1, t2          # (((in[1] & 0x0f) << 2) | ((in[2] & 0xc0) >> 6))
    la          t9, basis_64
    addu        t5, t9, t3          # Obtengo direccion basis_64 + indice_caracter
    lbu         t6, 0(t5)           # Lee codificacion basis_64[indice_caracter]
    sb          t6, 2(a1)           # Guardo codificacion en out[2]
    j           siguiente_codigo
menor_igual_2:
    la          t9, basis_64
    lbu         t6, 64(t9)          # Obtengo el caracter '='
    sb          t6, 2(a1)           # Guardo codificacion en out[2]
siguiente_codigo:
    lw          t1, 32($fp)         # leo LEN
    li          t2, 2
    ble         t1, t2, menor_igual_3  # Saltar si no cumple len > 2
    lbu         t2, 2(a0)           # Leo in[2]
    andi        t2, t2, 0x3f        # Calculo indice = (in[2] & 0x3f)
    la          t9, basis_64        # Obtengo 1er dir de basis_64
    addu        t9, t9, t2          # Obtengo direccion basis_64[indice]
    lbu         t6, 0(t9)           # Obtengo codigo basis_64[indice]
    sb          t6, 3(a1)           # Guardo codificacion en out[3]
    j           retornar_transform
menor_igual_3:
    la          t9, basis_64
    lbu         t6, 64(t9)          # Obtengo el caracter '='
    sb          t6, 3(a1)           # Guardo codificacion en out[3]
retornar_transform:
    move        sp, $fp
    lw          $fp, 16(sp)
    lw          gp, 20(sp)
    addiu       sp, sp, 24
    jr          ra
    .end        bloqueToBase64



#define ENCODE_STACK_SIZE       64
#define ENCODE_VAR_RA           60
#define ENCODE_VAR_GP           56
#define ENCODE_VAR_FP           52
#define WRITE_FILENO_IN         28
#define WRITE_FILENO_OUT        32

#define VAR_READ_CODE          40
#define VAR_RETCODE            24
#define VAR_LEN                20
#define VAR_I                  16


    .text
    .align 2
    .globl base64_encode
    .ent base64_encode
base64_encode:
    .frame      $fp, ENCODE_STACK_SIZE, ra
    .set        noreorder
    .cpload     t9
    .set        reorder
    subu        sp, sp, ENCODE_STACK_SIZE
    .cprestore  ENCODE_VAR_GP
    sw          ra, ENCODE_VAR_RA(sp)       #save ra
    sw          $fp, ENCODE_VAR_FP(sp)      #save $fp
    move        $fp, sp
    li          t0, 1
    sw          t0, VAR_READ_CODE(sp)     # read_code
    sw          zero, VAR_RETCODE(sp)     # retcode
    sw          a0, WRITE_FILENO_IN($fp)     # Guardo file descriptor entrada
    sw          a1, WRITE_FILENO_OUT($fp)    # Guardo file descriptor salida
while_read_code:
    lw          t0, VAR_READ_CODE($fp)        # Leemos codigo retorno de read
```

```
    li          t1, 1                       # Se lee de 1 byte
    bne         t0, t1, return_codificar    # Mientras se lea un byte - while (read_code == 1)
    sw          zero, VAR_LEN($fp)          # len = 0
    lw          t1, VAR_LEN($fp)            # t1<---len
    sw          zero, VAR_I($fp)            # Incializamos i = 0 para leer arch entrada

for_entrada:
    lw          t2, VAR_I($fp)              # Loop for para lectura
    li          t3, 3                       # Leemos hasta 3 bytes
    bge         t2, t3, aplicar_codificacion # Si ya leimos 3, salto a aplicar_codificacion
    lw          a0, WRITE_FILENO_IN($fp)    # Cargamos parametro 1 file descriptor entrada
    la          a1, buffer_read             # Cargamos direccion del buffer de lectura
    li          a2, 1                       # Cargamos longitud de lectura
    li          v0, SYS_read                # Cargamos syscall READ
    syscall                                 # Seria read(fileDescriptorEntrada, byte_read, 1);
    bnez        v0, else_if_uno             # Si v0 es cero continuo, sino salto a 'else_if_uno'
    la          t6, array_in                # v0 fue cero, o sea que no leyo nada
    lw          t2, VAR_I($fp)              # Tomo valor de I
    addu        t7, t6, t2                  # Calculo direccion de in[i]
    sb          zero, 0(t7)                 # Guardo cero en in[i]
    sw          zero, VAR_READ_CODE($fp)    # Actualizamos codigo retorno de read
    j           continuar_for               # vemos si continuamos
else_if_uno:
    li          t4, 1                       # Vemos si leimos 1 byte
    bne         v0, t4, else_error_file_in  # Si no leimos 1 byte, salto por q dio error
    la          t5, buffer_read             # Cargo la direccion del buffer de lectura
    lbu         t5, 0(t5)                   # Obtengo el byte leido en t5
    la          t6, array_in                # Cargo la direccion del BLOQUE in[]
    lw          t2, VAR_I($fp)              # Tomo valor de I
    addu        t7, t6, t2                  # Calculo direccion de in[i]
    sb          t5, 0(t7)                   # Guardo el dato t5 en in[i]
    lw          t2, VAR_LEN($fp)            # Leo LEN
    add         t2, t2, 1                   # Actualizo LEN++
    sw          t2, VAR_LEN($fp)            # Guardo LEN
    j           continuar_for               # Siguiente iteracion
else_error_file_in:
    li          t4, 4                       # Cargo codigo de error
    sw          t4, VAR_RETCODE($fp)        #
    j           return_codificar
continuar_for:
    lw          t2, VAR_I($fp)              # t2<---i
    addiu       t2, t2, 1                   # i++ para el for de lectura
    sw          t2, VAR_I($fp)              # Guardo i++
    j           for_entrada


aplicar_codificacion:
    lw          t2, VAR_LEN($fp)            # Obtenemos cuanto vale len
    bgtz        t2, mapear_base64           # if len > 0 codificar
    j           while_read_code
mapear_base64:
    la          a0, array_in
    la          a1, array_out
    lw          a2, VAR_LEN($fp)
    jal         bloqueToBase64              # Llamamo a bloqueToBase64
    sw          zero, VAR_I($fp)            # i = 0
for_salida:
    lw          t2, VAR_I($fp)              # t2<---i
    li          t4, 4
    bge         t2, t4, otro_while          # Itererar para escribir en file desc salida
    la          t3, array_out               # Leo direccion de 'array_out'
    addu        t5, t3, t2                  # t5 = out[i]    bloque de salida
    lw          a0, WRITE_FILENO_OUT($fp)
    move        a1, t5
    li          a2, 1
    li          v0, SYS_write
    syscall                                 #write(fileDescriptorSalida, (void*)(&out[i]), 1);
    bgez        v0, siguiente_escritura
    li          t4, 5                        # Cargo codigo de error
    sw          t4, VAR_RETCODE($fp)         # retcode = 1
    j           return_codificar
siguiente_escritura:
    lw          t2, VAR_I($fp)              # t2<---i
    addiu       t2, t2, 1
    sw          t2, VAR_I($fp)
    j           for_salida
otro_while:                                 # Itero la siguiente terna de bytes
    j           while_read_code
return_codificar:
    lw          v0, VAR_RETCODE(sp)         # Liberamos el stack frame
    move        sp, $fp
    lw          $fp,  ENCODE_VAR_FP(sp)
    lw          gp, ENCODE_VAR_GP(sp)
    lw          ra, ENCODE_VAR_RA(sp)
    addu        sp, sp, ENCODE_STACK_SIZE
    jr           ra
    .end        base64_encode

#########FIN FUNCION ENCODE############


    .data
    .align 2
    buffer:             .space 4
    buffer_write:       .space 4
    basis_64:           .byte   'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
                                'R','S','T','U','V','W','X','Y','Z', 'a','b','c','d','e','f','g','h',
                                'i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y',
```

```
                                'z','0','1','2','3','4','5','6','7','8','9','+','/','='

.align      2
buffer_read:        .space 4
array_out:          .space 4
array_in:           .space 3

.rdata
.align 2
.globl errmsg
errmsg: .word msg_error_success, msg_error_busqueda, msg_error_lectura, msg_error_escritura,
              msg_error_lectura_codif, msg_error_escritura_codif, msg_error_lect_incomplet
.align 2
msg_error_success: .asciiz "No hubo errores en la decodificacion."
msg_error_busqueda: .asciiz "Hubo un error en la busqueda del caracter a decodificar."
msg_error_lectura: .asciiz "Hubo un error en la lectura del archivo a decodificar."
msg_error_escritura: .asciiz "Hubo un error en la escritura del archivo para la decodificacion."
msg_error_lectura_codif: .asciiz "Hubo un error en la lectura del archivo para codificar."
msg_error_escritura_codif: .asciiz "Hubo un error en la escritura del archivo cuando codificamos"
msg_error_lect_incomplet: .asciiz "Lectura incompleta de bloques a decodificar"
```

5. Conclusiones

1. La realización completa del TP llevó cierta dificultad al tener que realizarlo en el contexto solicitado: alta portabilidad, desarrollo en C, funciones en assembly respetando la convención de la ABI e informe hecho en LaTeX.

2. En el primer caso la dificultad radicaba en tener configurado y funcionando el GXEmul dentro de un Linux, y lograr que en ambos casos el programa compile y corra sin problemas.

3. Tuvimos que invertir tiempo para leer la bibliografía dada por la catedra para respetar la convención de la ABI para el desarrollo en assembly de las funciones solicitadas.

4. En cuanto al trabajo grupal en si mismo, no hubo inconvenientes de ningún tipo ya que al ser el grupo relativamente chico y tener conocimiento del manejo del versionado de un proyecto ante cambios ingresado por los integrantes (por medio del GIT), la introducción de modificaciones y correcciones fué fluida.

6. Enunciado del trabajo practico

Universidad de Buenos Aires, F.I.U.B.A.
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
`$Date: 2017/04/23 22:25:51 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El informe deberá ser entregado personalmente, por escrito, 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 6), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada caso.

4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior [1].

Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- `main.c`: contendrá todo el código necesario para el procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores (`stderr`). Desde aquí se llama a las funciones de encoding y decoding siguientes.

- `base64.S`: contendrá el código MIPS32 assembly con las funciones `base64_encode()` y `base64_decode()`, y las funciones y estructuras de datos auxiliares para realizar los cómputo de encoding y decoding, que los alumnos crean convenientes. También contendrá la definición en assembly de un vector equivalente al siguiente vector C: `const char* errmsg[]`. Dicho vector contendrá los mensajes de error que las funciones antes mencionadas puedan generar, y cuyo índice es el código de error devuelto por las mismas.
- Los header files pertinentes (al menos, `base64.h`, con los prototipos de las funciones mencionadas, a incluir en `main.c`), y la declaración del vector `extern const char* errmsg[]`).

A su vez, las funciones MIPS32 `base64_encode()` y `base64_decode()` antes mencionadas, coresponden a los siguientes prototipos C:

- `int base64_encode(int infd, int outfd)`
- `int base64_decode(int infd, int outfd)`

Ambas funciones reciben por `infd` y `outfd` los file descriptors correspondientes a los archivos de entrada y salida pre-abiertos por `main.c`, la primera función realizará el encoding a base 64 de su entrada, y la segunda función el decoding de base 64 se su entrada.

Ante un error, ambas funciones volverán con un código de error numérico (índice del vector de mensajes de error de `base64.h`), o cero en caso de realizar el procesamiento de forma exitosa.

5. Implementación

El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación:

5.1. ABI

Será necesario que el código presentado utilice la ABI explicada en clase ([2] y [3]).

5.2. Syscalls

Es importante aclarar que desde el código assembly no podrán llamarse funciones que no fueran escritas originalmente en assembly por los alumnos. Por lo contrario, desde el código C sí podrá (y deberá) invocarse código assembly.

Por ende, y atendiendo a lo planteado en la sección 4, los alumnos deberán invocar algunos de los system calls disponibles en NetBSD (en particular, `SYS_read` y `SYS_write`).

5.3. Casos de prueba

Es necesario que la implementación propuesta pase todos los casos incluidos tanto en el enunciado del trabajo anterior [1] como en el conjunto de pruebas suministrado en el informe del trabajo, los cuales deberán estar debidamente documentados y justificados.

5.4. Documentación

El informe deberá incluir una descripción detallada de las técnicas y procesos de desarrollo y debugging empleados, ya que forman parte de los objetivos principales del trabajo.

6. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño, desarrollo y debugging del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, (sección 5.3) con los comentarios pertinentes;
- El código fuente completoi, el cual deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas);
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

7. Fechas

- Entrega: 2/5/2017;
- Vencimiento: 16/5/2017.

Referencias

[1] Enunciado del primer trabajo práctico (TP0), primer cuatrimestre de 2017 (`http://groups.yahoo.com/groups/orga-comp/files/TPs/`).

[2] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

[3] MIPS ABI: Function Calling Convention, Organización de computadoras - 66.20 (archivo "func_call_conv.pdf", `http://groups.yahoo.com/groups/orga-comp/Material/`).