66.20 - Organización de Computadoras
Trabajo Práctico 1 - Conjunto de instrucciones MIPS

Nicolás Alvarez, Padrón 93503
Nicolás Fernandez, Padrón 88599
Andrew Parlane

26 de abril de 2018

Índice

1. Resumen

El objetivo de este trabajo se basa en aplicar los conocimientos aprendidos en clase acerca de las instrucciones MIPS32 y el concepto de ABI. Para eso desarrollaremos un programa en C que utilizará una función desarrollada en lenguaje assembler MIPS32.

2. Introducción

El programa que desarrollaremos se encargará de tomar una matriz de números enteros y devolverá su matriz transpuesta. El programa recibirá como argumento el nombre del archivo donde está especificada la matriz, y dará el resultado por stdout o lo escribirá en un archivo, según las opciones de entrada. De haber errores, los mensajes de error saldrán exclusivamente por stderr.

3. Compilacion y ejecución del programa

Esta sección describe los mecanismos para la compilación y ejecución del programa, tanto en entorno MIPS como en Linux.

En una primera etapa compilamos y trabajamos en Linux para poder realizar las pruebas. Adicionalmente a esto utilizamos Valgrind para chequear que no tengamos problemas de leaks o free en nuestro código.

Para compilar usando el Makefile:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  host# make
  ```

  Y para la machina de MIPS:

  ```
  host# gmake
  host# gmake asm
  ```

Para ejecutar:

- Estando parados en la carpeta donde se encuentran el archivo ejecutable, corremos para cada archivo o fragmento de prueba lo siguiente:

  ```
  ./tp1_c [options] file
  ```

  Y en la machina MIPS:

  ```
  ./tp1_asm [options] file
  ```

- Si se quiere ver el help, que especifica las opciones disponibles al momento de invocar el programa:

  ```
  ./tp1_c -h
  ```

- Si se quiere ver la versión del programa:

  ```
  ./tp1_c -V
  ```

4. Implementación

4.1. Implementación en C

El trabajo se estructuró en tres archivos, uno que poseerá la mayor parte del programa (procesamiento de argumentos, usage, etc.) y lo otros dos la función ***transponer*** en código C y en código assembler MIPS32:

- **main.c** : Define el proceso principal de ejecución, la validación de los parámetros pasados al programa y además los métodos para parseo y salida del programa.
- **transpose.c** : Define la función transponer en código c, que recibe una matriz y devuelve su transpuesta.

- **transpose.S** : Define la función transponer en código assembler, que recibe una matriz y devuelve su transpuesta.

A continuación enumeramos las funciones definidas en el programa que se usarán luego de la verificación y validación de los parámetros de entrada:

- **usage**
  parámetros: FILE *stream
  const char *nuestroNombre

  descripción: función que muestra el help de la aplicación.

- **leerLongLong**
  parámetros: FILE *f
  long long *ll
  bool *OK
  bool *eof
  bool *newLine

  descripción: función que lee un entero de la matriz.

- **leerLinea**
  parámetros: FILE *f
  long long *data
  uint columnasEsperados
  bool *eof

  descripción: función que lee una línea completa de la matriz.

- **leerEntrada**
  parámetros: const char *archivo
  uint *filas
  uint *columnas

  descripción: función que lee el archivo de entrada e inicia el procesamiento de la matriz.

- **escribirSalida**
  parámetros: const char *archivo
  uint filas
  uint columnas
  long long *salida

  descripción: función que escribe el resultado a un archivo o stdout.

- **transponer**
  parámetros: unsigned int *filas
  uint *columnas
  long long *entrada
  long long *salida

  descripción: función lee una matriz y devuelva su matriz transpuesta.

5. Pruebas

Realizamos las pruebas en GXEmul para cada uno de los archivos pedidos.
- **matrix1**
- **matrix2**
- **matrix3**

```
$ ./tp1_c -o - pruebas/matrix1
7 1
1
2
3
4
5
6
7

$ ./tp1_c -o - pruebas/matrix2
Not enough entries on line. Expecting 5, found 4

$ ./tp1_c -o - pruebas/matrix3
Found invalid character .
```

También se realizó unas pruebas con otros archivos para detectar otros casos posibles en el archivo de entrada.

```
$ cat matrix_tabs
4   3
1   2   3
                4           5           6
7 8 9
10                                          11 12

$ ./tp1_c -o - pruebas/matrix_tabs
3 4
1 4 7 10
2 5 8 11
3 6 9 12

$ cat matrix_negativo
4 2
2 1
0 -1
-2 -3
-4 -5

$ ./tp1_c -o - pruebas/matrix_tabs
2 4
2 0 -2 -4
1 -1 -3 -5

$ cat matrix_long_long
5 4
9223372036854775807 0 1234567891011121314 1
1516171819202122232 4252627282930313233 2 3
4 3435363738394041424 3444546474849505152 6
5354555657585960616 2636465666768697071 7273747576777787980 8182838485868788899
0 0 0 0

$ ./tp1_c -o - pruebas/matrix_long_long
4 5
9223372036854775807 1516171819202122232 4 5354555657585960616 0
0 4252627282930313233 3435363738394041424 2636465666768697071 0
1234567891011121314 2 3444546474849505152 7273747576777787980 0
1 3 6 8182838485868788899 0
```

Se incluirán en la entrega más archivos que fueron usados para probar la robustez del programa. También el Makefile incluye una regla "prueba"que ambos de tp1_c y tp1_asm por cada prueba, comparando el código de salida con lo que es esperado, y el matrix resultado con lo que es esperado.

6. Diagrama del stack del programa

A continuación se podrá ver el diagrama de cómo quedaría el stack justo después de entrando la función transponer. Usando *objdump* encontramos que el stack de main es 88 bytes con 16 bytes por la SRA. La función ***leerLongLong*** tiene cinco argumentos, así la ABA de main debería estar 24 bytes. Los último 48 bytes están la LTA.

```
#   +---------------+   |------------MAIN------------
# 92    |       |   |   \
#   +---------------+   |
# 88    |   ra  |   |
#   +---------------+   |   SRA MAIN
# 84    |   S8  |   |
#   +---------------+   |
# 80    |   gp  |   |   /
#   +---------------+
# 76    |  LTA  |   |   \
#   +---------------+   |
# ..    |  LTA  |   |
#   +---------------+   |   LTA MAIN
# ..    |  LTA  |   |
#   +---------------+   |
# 32    |  LTA  |   |   /
#   +---------------+
# 28    |       |   |   \
#   +---------------+   |
# 24    |       |   |
#   +---------------+   |
# 20    |salida |   |
#   +---------------+   |   ABA MAIN
# 16    |entrada|   |
#   +---------------+   |
# 12    |columnas|  |
#   +---------------+   |
# 08    | filas |   |   /
#   +---------------+   |----------TRANSPONER----------
# 04    |   fp  |   |   \
#   +---------------+   |   SRA TRANSPONER
# 00    |   gp  |   |   /
#   +---------------+
```

7. Conclusiones

El desarrollo de este trabajo nos permitió llevar a la práctica los conocimientos adquiridos acerca de la estructura MIPS32. Debimos respetar la ABI de esta arquitectura, respetando los tamaños de las distintas secciones: SRA (Saved Register Area), LTA (Local and Temp Area) y ABA (Arg Building Area).

Para otorgar portabilidad a esta arquitectura desarrollamos la función transponer en lenguaje assembler MIPS32. Pudimos comprabar que fue un éxito al realizar la compilación de programa con la función ***transponer*** desarrollada en assembler en el archivo ***transpose.S***

Una observación que se puede apreciar en el diagrama del stack es que la función transponer posee un stack de 8 bytes ya que es una función *hoja* por lo cual no tendrá una ABA ni una LTA y no se deberá guardar el registro RA, mientras que el stack del main mide 88 bytes ya que reserva espacio de ABA para los argumentos que se usarán en llamar otras funciones, y como no es una función *hoja* guarda algunas variables temporales en la LTA.

8. Código

8.1. main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <getopt.h>
#include <ctype.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include <errno.h>

#define MAJOR_VERSION   1
#define MINOR_VERSION   0

/*
 *   +---------------+   |------------MAIN------------
 * 84    |       |   |   \
 *   +---------------+   |
 * 80    |   ra  |   |
 *   +---------------+   |   SRA MAIN
 * 76    |   S8  |   |
 *   +---------------+   |
 * 72    |   gp  |   |   /
 *   +---------------+
 * 68    |  LTA  |   |   \
 *   +---------------+   |
 * ..    |  LTA  |   |
 *   +---------------+   |   LTA MAIN
 * ..    |  LTA  |   |
 *   +---------------+   |
 * 24    |  LTA  |   |   /
 *   +---------------+
 * 20    |  ABA  |   |   \
 *   +---------------+   |
 * 16    |  ABA  |   |
 *   +---------------+   |
 * 12    |  ABA  |   |
 *   +---------------+   |   ABA MAIN
 *  8    |  ABA  |   |
 *   +---------------+   |
 *  4    |  ABA  |   |
 *   +---------------+   |
 *  0    |  ABA  |   |   /
 *   +---------------+
 */

// declaración adelante
// transponer es en transponer.c o transponer.s
extern int transponer(unsigned int filas,
                      unsigned int columnas,
                      long long *entrada,
                      long long *salida);

static const struct option long_options[] =
{
    {"help",    no_argument,        0, 'h' },
```

```
    {"version", no_argument,        0, 'V' },
    {"output",  required_argument,  0, 'o' },
    {0,         0,                  0,  0  }
};

static void usage(FILE *stream, const char *nuestroNombre)
{
    fprintf(stream,
           "Usage:\n"
           "  %s -h\n"
           "  %s -V\n"
           "  %s [options] filename\n"
           "Options:\n"
           "  -h, --help Prints usage information.\n"
           "  -V, --version Prints version information.\n"
           "  -o, --output Path to output file.\n"
           "\n"
           "Examples:\n"
           "  %s -o - mymatrix\n",
           nuestroNombre, nuestroNombre, nuestroNombre, nuestroNombre);
    // necesitamos usar nuestroNombre, nuestroNombre, nuestroNombre, nuestroNombre
    // porque no soportamos %1$s
}

// lea carácter por carácter deshaciendo whitespace
// hasta encontrar [0-9-]. Después comenzar a leer números
// [0-9]. Para cuando obtenemos EOF, \n, \r, ' ', \t.
// Es un error si encontramos algún otro carácter.
// devolver no 0 si hay un error
// *OK = 1 -> hay un integer valido en *ll
// *eof = 1 -> no hay más a leer
// *newLine = 1 -> encontramos nueva línea
static bool leerLongLong(FILE *f, long long *ll, bool *OK, bool *eof, bool *newLine)
{
    *OK = false;
    *eof = false;
    *newLine = false;

    // soportamos signed 64 bits:
    // máx = 0x7FFF_FFFF_FFFF_FFFF =  9223372036854775807
    // min = 0x8000_0000_0000_0000 = -9223372036854775808
    // así max input legal es 20 cáracters +1 por NULL terminator
#define MAX_CHARS 20
    char buff[MAX_CHARS + 1];
    uint idx = 0;

    bool comenzandoLeerInt = false;
    while (1)
    {
        int res = fgetc(f);
        if (res == EOF)
        {
            *eof = true;
            // si tenemos algo en buff, convertimos ahora
            if (*OK)
            {
                buff[idx] = '\0';
                *ll = strtoll(buff, NULL, 10);
```

```
        if (errno != 0)
        {
            fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff, strerror(e
            return false;
        }
        return true;
    }
    else if (comenzandoLeerInt)
    {
        // solo podríamos estar aquí si leemos
        // un '-' y después nada, eso es un error
        fprintf(stderr, "Found invalid entry \"-\"\n");
        return false;
    }
    else
    {
        // eof but no error
        return true;
    }
}

char c = (char)res;
if (c == '\r' || c == '\n')
{
    *newLine = true;
}

if (!comenzandoLeerInt)
{
    // Todavía no cemenzamos a leer el int
    if (c == ' ' || c == '\t')
    {
        // ignoramos
        continue;
    }
    else if (*newLine)
    {
        // nuevo línea, terminamos.
        return true;
    }
    else if (c >= '0' && c <= '9')
    {
        // válido
        buff[idx++] = c;
        comenzandoLeerInt = true;
        *OK = true;
    }
    else if (c == '-')
    {
        // también válido pero el int todavía no es OK
        // porque necesitamos un número después de un -
        buff[idx++] = c;
        comenzandoLeerInt = true;
    }
    else
    {
        // error
        fprintf(stderr, "Found invalid character %c\n", c);
```

```
                return false;
            }
        }
        else
        {
            // ya estamos leyendo data
            if (c == ' ' || c == '\t' ||
                *newLine)
            {
                // terminamos
                if (*OK)
                {
                    buff[idx] = '\0';
                    *ll = strtoll(buff, NULL, 10);
                    if (errno != 0)
                    {
                        fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff, strerr
                        return false;
                    }
                    return true;
                }
                else
                {
                    // solo podríamos estar aquí si leemos
                    // un '-' y después nada, eso es un error
                    fprintf(stderr, "Found invalid entry \"-\"\n");
                    return false;
                }
            }
            else if (c == '-')
            {
                // un - aquí no es válido porque estámos en el medio
                // de un int.
                fprintf(stderr, "Found \"-\" in the middle of an integer\n");
                return false;
            }
            else if (c >= '0' && c <= '9')
            {
                // válido
                if (idx >= MAX_CHARS)
                {
                    fprintf(stderr, "Integer read was too large to fit into a long long\n");
                    return false;
                }
                buff[idx++] = c;
                *OK = true;
            }
            else
            {
                // error
                fprintf(stderr, "Found invalid character %c\n", c);
                return false;
            }
        }
    }
}

static bool leerLinea(FILE *f, long long *data, uint columnasEsperados, bool *eof)
```

```
{
    uint32_t count = 0;
    while (1)
    {
        bool OK;
        bool newLine;
        long long ll;
        if (!leerLongLong(f, &ll, &OK, eof, &newLine))
        {
            // error
            return false;
        }

        if (OK)
        {
            // leemos un integer
            if (count >= columnasEsperados)
            {
                // error - hay mas columnas de las esperadas
                fprintf(stderr, "Too many entries on line. Expecting %u\n", columnasEsperados);
                return false;
            }

            data[count] = ll;
            count++;
        }

        if (*eof)
        {
            if (count != columnasEsperados)
            {
                // error - hay menos columnas de las esperadas
                fprintf(stderr, "Not enough entries on line. Expecting %u, found %u\n", columnasEspera
                return false;
            }
            else
            {
                return true;
            }
        }

        if (newLine)
        {
            if (count == 0) // permitimos newLines antes de data comenzando
            {
                continue;
            }
            else if (count != columnasEsperados)
            {
                // error - hay menos columnas de las epseradas.
                fprintf(stderr, "Not enough entries on line. Expecting %u, found %u\n", columnasEspera
                return false;
            }
            else
            {
                return true;
            }
        }
```

```
    }
    return true;
}

static long long *leerEntrada(const char *archivo, uint *filas, uint *columnas)
{
    FILE *f = fopen(archivo, "r");
    if (f == NULL)
    {
        fprintf(stderr, "%s: No such file or directory\n", archivo);
        return NULL;
    }

    long long primerLinea[2];

    bool eof;
    if (!leerLinea(f, primerLinea, 2, &eof))
    {
        fclose(f);
        return NULL;
    }

    long long *llFilas = &primerLinea[0];
    long long *llColumnas = &primerLinea[1];

    // Validar filas y columnas
    // no pueden ser menor a cero
    // ni mas grande que 0xFFFFFFFF
    if (*llFilas < 0 || *llColumnas < 0 ||
        *llFilas > 0xFFFFFFFF ||
        *llColumnas > 0xFFFFFFFF)
    {
        fprintf(stderr, "Invalid number of rows / columns\n");
        fclose(f);
        return NULL;
    }

    *filas = *(uint *)llFilas;
    *columnas = *(uint *)llColumnas;

    // numero de elementos = filas * columnas
    // cada uno es un long long, así:
    long long *entrada = malloc(*filas * *columnas * sizeof(long long));
    if (entrada == NULL)
    {
        fprintf(stderr, "Failed to malloc %u bytes\n", (unsigned int)(*filas * *columnas * sizeof(long
        fclose(f);
        return NULL;
    }

    uint i;
    for (i = 0; i < *filas; i++)
    {
        if (!leerLinea(f, &entrada[i * *columnas], *columnas, &eof))
        {
            fclose(f);
            free(entrada);
            return NULL;
```

```
        }
    }

    // debería estar todo, comprobar que no hay más data
    while (!eof)
    {
        if (!leerLinea(f, NULL, 0, &eof))
        {
            fclose(f);
            free(entrada);
            return NULL;
        }
    }

    fclose(f);
    return entrada;
}

static bool escribirSalida(const char *archivo, uint filas, uint columnas, long long *salida)
{
    FILE *f;

    if (archivo == NULL)
    {
        // stdout
        f = stdout;
    }
    else
    {
        // archivo
        f = fopen(archivo, "w");
        if (f == NULL)
        {
            fprintf(stderr, "Failed to open %s for writing\n", archivo);
            return NULL;
        }
    }

    fprintf(f, "%u %u\n", filas, columnas);
    uint i;
    for (i = 0; i < filas; i++)
    {
        uint c;
        for (c = 0; c < columnas; c++)
        {
            fprintf(f, "%lld ", salida[(i * columnas) + c]);
        }
        fprintf(f, "\n");
    }

    if (archivo != NULL)
    {
        fclose(f);
    }

    return true;
}
```

```c
int main(int argc, char **argv)
{
    // usamos argv[0] como el nombre del aplicación
    // pero solo queremos el archivo, no la ruta
    const char *nuestroNombre = basename(argv[0]);

    // escribir la salida a un archivo si vemos -o (y el argumento no es -)
    const char *oArchivo = NULL;

    // clear errors
    opterr = 0;

    // parse short options
    while (1)
    {
        // obtener el siguiente argumento
        int option_index = 0;
        int c = getopt_long(argc, argv, "hVo:", long_options, &option_index);

        if (c == -1)
        {
            // no hay más
            break;
        }

        switch (c)
        {
            case 'h':
            {
                usage(stdout, nuestroNombre);
                // no seguimos despues de -h
                return 0;
            }
            case 'V':
            {
                printf("%s: Version %u.%u\n", nuestroNombre, MAJOR_VERSION, MINOR_VERSION);
                // no seguimos despues de -V
                return 0;
            }
            case 'o':
            {
                // si vemos "-o -" la salida es stdout
                // si no, la salida es el archivo en optarg
                if (strcmp(optarg, "-") != 0)
                {
                    oArchivo = optarg;
                }
                break;
            }
            case '?':
            {
                if (optopt == 'o')
                {
                    fprintf(stderr, "Option '-%c' requires an argument.\n\n", optopt);
                }
                else if (isprint(optopt))
                {
                    // es un argumento, pero no es uno que esperamos
```

```
                fprintf (stderr, "Unknown option '-%c'.\n\n", optopt);
            }
            else
            {
                // solo muestra el usage
            }
            usage(stderr, nuestroNombre);
            return 1;
        }
        default:
        {
            usage(stderr, nuestroNombre);
            return 1;
        }
    }
}

if (optind == argc)
{
    fprintf(stderr, "filename is required\n\n");
    usage(stderr, nuestroNombre);
    return 1;
}

if ((optind + 1) != argc)
{
    fprintf(stderr, "Too many arguments\n\n");
    usage(stderr, nuestroNombre);
    return 1;
}

// leer archivo
uint filas;
uint columnas;
long long *entrada = leerEntrada(argv[optind], &filas, &columnas);
if (entrada == NULL)
{
    // falla, leerArchivo escribí el error
    return 1;
}

// malloc la salida
long long *salida = malloc(filas * columnas * sizeof(long long));
if (salida == NULL)
{
    fprintf(stderr, "Failed to allocate %u bytes for output\n", (unsigned int)(filas * columnas *
    free(entrada);
    return 1;
}

// transponer
if (transponer(filas, columnas, entrada, salida) != 0)
{
    fprintf(stderr, "Failed to transpose the matrix\n");
    free(entrada);
    free(salida);
    return 1;
}
```

```
    // escribir el resultado
    bool res = escribirSalida(oArchivo, columnas, filas, salida);
    free(entrada);
    free(salida);

    return res ? 0 : 1;
}
```

8.2. transpose.c

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

int transponer(unsigned int filas, unsigned int columnas, long long *entrada, long long *salida)
{
    uint f;
    for (f = 0; f < filas; f++)
    {
        uint c;
        for (c = 0; c < columnas; c++)
        {
            salida[(c * filas) + f] = entrada[(f * columnas) + c];
        }
    }

    return 0;
}
```

8.3. transpose.S

```
 #   +---------------+
 # 20    |salida |   |   \
 #   +---------------+   |
 # 16    |entrada|   |
 #   +---------------+   |   ABA MAIN
 # 12    |columnas|  |
 #   +---------------+   |
 # 08    | filas |   |   /
 #   +---------------+   |----------TRANSPONER----------
 # 04    |   fp  |   |   \
 #   +---------------+   |   SRA TRANSPONER
 # 00    |   gp  |   |   /
 #   +---------------+

#include <mips/regdef.h>

    .text
    .align  2
    .globl  transponer
    .ent    transponer

# int transponer(unsigned int filas,
#                unsigned int columnas,
#                long long *entrada,
#                long long *salida)
```

```
transponer:
    subu    sp, sp, 8
    sw      gp,  0(sp)
    sw      $fp, 4(sp)
    sw      a0,  8(sp)
    sw      a1, 12(sp)
    sw      a2, 16(sp)
    sw      a3, 20(sp)

    # a0 = filas
    # a1 = columnas
    # a2 = &entrada[0]
    # a3 = &salida[0]

    move    v0, zero            # siempre devolvemos 0 no hay errores posibles

    beqz    a0, fin             # if (filas == 0) return 0;
    beqz    a1, fin             # if (columnas == 0) return 0;

    move    t0, zero            # uint f = 0;

filaLoop:                       # do {
    move    t1, zero            #   uint c = 0;

columnaLoop:                    #   do {
    ld      t2, 0(a2)           #     (t2,t3) = *entrada;
    addu    a2, a2, 8           #     entrada++;

    mul     t4, t1, a0          #     t4 = (c * filas);
    addu    t4, t4, t0          #     t4 += f;
    mul     t4, t4, 8           #     t4 = offset en salida
    addu    t4, t4, a3          #     t4 = &salida[(c * filas) + f]
    sd      t2, 0(t4)           #     salida[(c * filas) + f] = (t2,t3)

    addu    t1, t1, 1           #     c++;
    bne     t1, a1, columnaLoop #   } while (c != columnas)

    addu    t0, t0, 1           #   f++;
    bne     t0, a0, filaLoop    # } while (f != filas)

fin:
    lw      gp, 0(sp)
    lw      $fp, 4(sp)
    addu    sp, sp, 8
    jr      ra

    .end transponer
```

8.4. Makefile

```
C_TARGET = tp1_c
ASM_TARGET = tp1_asm

MACHINE = $(shell uname -m)

LIBS =
CC = gcc
C_FLAGS = -Wall -g
```

```
ASM_FLAGS = -Wall -g

default: $(C_TARGET)
all: default

C_OBJECTS = main.c.o \
                    transpose.c.o

ASM_OBJECTS = main.c.o \
                          transpose.S.o

HEADERS = $(wildcard *.h)

ifeq (£(MACHINE), pmax)

define HACE_PRUEBA_ASM
        @echo Probando código ASM con pruebas/$(strip $(1))
        @./pruebaScript.sh $(ASM_TARGET) $(strip $(1)) $(2)
endef

else

define HACE_PRUEBA_ASM
        @#Hace nada porque no estamos pmax
endef

endif

# Macro con dos argumentos
#  1) El nombre de archivo a probar en pruebas/
#  2) Código de salida esperado
#
# Si el código de salida es igual al esperada,
# combrobamos la salida con un archivo que
# tiene el mismo nombre del input en resultados/

define HACE_PRUEBA

        @echo Probando código C con pruebas/$(strip $(1))
        @./pruebaScript.sh $(C_TARGET) $(strip $(1)) $(2)
        $(call HACE_PRUEBA_ASM, $(1), $(2))
endef


%.c.o: %.c $(HEADERS)
        $(CC) $(C_FLAGS) -c $< -o $@

%.S.o: %.S $(HEADERS)
        $(CC) $(ASM_FLAGS) -c $< -o $@

$(C_TARGET): $(C_OBJECTS)
        $(CC) $(C_OBJECTS) $(C_FLAGS) $(LIBS) -o $@

$(ASM_TARGET): $(ASM_OBJECTS)
        $(CC) $(ASM_OBJECTS) $(ASM_FLAGS) $(LIBS) -o $@

C: $(C_TARGET)
```

```
ifeq ($(MACHINE), pmax)

ASM: $(ASM_TARGET)

else

ASM:

endif

c: C

asm: ASM

prueba: C ASM
	@# Primero el básico
	-$(call HACE_PRUEBA, matrix1, 0)
	@# Con finales de líneas diferentes
	-$(call HACE_PRUEBA, matrix_crlf, 0)
	-$(call HACE_PRUEBA, matrix_cr_only, 0)
	-$(call HACE_PRUEBA, matrix_lf_only, 0)
	@# Espacio blanco extra
	-$(call HACE_PRUEBA, matrix_filas_blancas, 0)
	-$(call HACE_PRUEBA, matrix_tabs, 0)
	@# Numeros negativos o grandes, pero en el rango de signed long long
	-$(call HACE_PRUEBA, matrix_negativo, 0)
	-$(call HACE_PRUEBA, matrix_long_long, 0)
	@# Inválidos
	-$(call HACE_PRUEBA, matrix2, 1)
	-$(call HACE_PRUEBA, matrix3, 1)
	-$(call HACE_PRUEBA, matrix_filas_negativos, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_columnos, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_filas, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_largo_int, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_largo_int2, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_negativo_int, 1)
	-$(call HACE_PRUEBA, matrix_demasiado_negativo_int2, 1)
	-$(call HACE_PRUEBA, matrix_no_suficiente_filas, 1)
	-$(call HACE_PRUEBA, archivo_que_no_existe, 1)

clean:
	-rm -f *.o
	-rm -f $(C_TARGET)
	-rm -f $(ASM_TARGET)
	-rm -f salida stdout stderr

.PHONY: default all C c ASM asm clean
```

8.5. pruebaScript.sh

```
#!/bin/sh

# Argumentos:
#  £1 applicación
#  £2 entrada en pruebas/ sin la ruta
#  £3 código de salida esperada

./$1 -o salida pruebas/$2 > stdout 2> stderr
```

```
export RES_CODE=$?
if [ $RES_CODE -eq $3 ]; then
    if [ $RES_CODE -ne 0 ]; then
        echo "  OK";
    else
        diff -w salida pruebas/esperados/$2 > /dev/null;
        if [ $? -eq 0 ]; then
            echo "  OK";
        else
            echo "  Transpuesta no es como esperada";
        fi
    fi
else
    echo "  Código de salida no es esperado";
fi
```

9. Enunciado

**Ver hojas anexadas*