66.20 Organización de Computadoras: Trabajo Práctico 1

Eirik Harald Lund, *Padrón Nro. 103081*,
eirikharald@hotmail.com
Julian Quino, *Padrón Nro. 94224*,
julian.quino2@gmail.com

Grupo Nro. ? - 1er. Cuatrimestre de 2018
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

09/04/2018

Resumen

Usando el lenguaje C, desarrollamos un programa que devuelve el traspuesto de una matriz de entrada. La salida es escribe a un archivo. Escribimos también un programa en la versión del lenguaje Assembly de MIPS32. Además de la matriz, la primera línea del archivo debe tener dos números que dan el tamaño de la matriz.

1. Documentación

El programa necesita abrir y leer un archivo. Después de hacer la transposición, el programa debe o imprimir el resultado por *stdout* o escribirlo a un archivo, dado por los argumentos. Todo eso se puede hacer con la biblioteca standard de C.

Los pasos del desarrollo eran:

1. Leer los argumentos y abrir el archivo
2. Leyendo el archivo, guardar las dimensiones de la matriz y recorrerla
3. Guardar la matriz en forma traspuesta

4. O imprimir por *stdout* o guardarla en un archivo

5. Durante todo, observar si el archivo de entrada tiene la forma correcta. Si no, manejarlo como un error

Los primeros dos pasos son bastante standard cuando se trabaja con C. El recorrido de la matriz implica comparar los datos leídos con la forma esperada. Si se encuenta algo erróneo, necesita terminar en una manera controlada.

Mientras se recorre el archivo, guarda los datos que se encuenten en la forma traspuesta. Entonces, cuando se termine, tiene la matriz nueva y puede continuar.

El cuarto paso se realiza usando métodos standard de C.

Finalmente, el quinto paso es parte de todo. El programa debía estar escrito para un formate específico de matrices, así que es importante que tenga verificaciones de los datos encontrados.

2. Comandos

Para compilar el programa en una versión de puro solo C se usa el siguiente comando:

```
$ gcc -Wall -g -o tp1 tp1-orga.c
```

Para el programa con una versión de MIPS32 y C se compila con el siguiente comando:

```
$ gcc -Wall -g -o tp1-MIPS transponer.S tp1-orga-MIPS.c
```

Esto da dos programas: *tp1* y *tp1-MIPS*. El primero sólo usa C, mientras la segunda usa la función de la transposición escrita en MIPS32 Assembly.

Para usar el programa, se puede usar un comando así:

```
$ ./tp1 -o - matriz
```

Esto imprimirá la matriz de salida por *stdout*, sin escribirla a un archivo. Si se quiere usar un archivo, puede usar un comando así:

```
$ ./tp1 -o salida matriz
```

Se puede por supuesto tambiíen reemplazar *tp1* con *tp1-MIPS*:

```
$ ./tp1-MIPS -o salida matriz
```

Para imprimir la información de ayuda, se usa uno de los siguientes:

```
$ ./tp1 -h
$ ./tp1 --help
```

Finalmente, los siguientes imprimirán la informaciíon de la versión:

```
$ ./tp1 -V
$ ./tp1 --version
```

3. Corridas de prueba

Se hicieron unas corridas de prueba para asegurar la exactitud y funcionalidad del programa. Usamos el mismo comando para cada una:

```
$ ./tp1 -o - matrix[#]
```

3.1. matrix1

El archivo original es

```
$ 1 7
$ 1 2 3 4 5 6 7
```

Usando el comando:

```
$ ./tp1 -o - matrix1
$ salida por stdout
$ 7 1
$ 1
$ 2
$ 3
$ 4
$ 5
$ 6
$ 7
```

Los datos están traspuestos, prueba exitosa.

3.2. matrix2

Entrada:

```
$ 3 5
$ 1234 5678 910 1112 1314
$ 20 300 4000 50000
$ 5 4 3 2
```

Salida:

```
$ salida por stdout
$ Error: faltan columnas en fila 3
```

El archivo de entrada dice que va a haber 5 columnas por tres filas, pero en la tercera fila sólo hay 4. Esto es un error y el programa lo encuentra. Prueba exitosa.

3.3. matrix3

El último archivo de prueba contiene:

```
$ 5 3
$ 1 2 34.5
$ 6 7 hola
$ 8 9 10
$ 11 12 13
```

Al fin eso da:

```
$ salida por stdout
$ Error de conversion en fila: 1, columna: 3
```

El programa sólo acepta números enteros. Como 34,5 no es entero, causa un error. El programa no llega hasta la "hola". Prueba exitosa.

Ademas de las pruebas anteriores, se implemento un conjunto de pruebas automatizadas (es necesario tener las dos versiónes compiladas) que se puede correr con el comando:

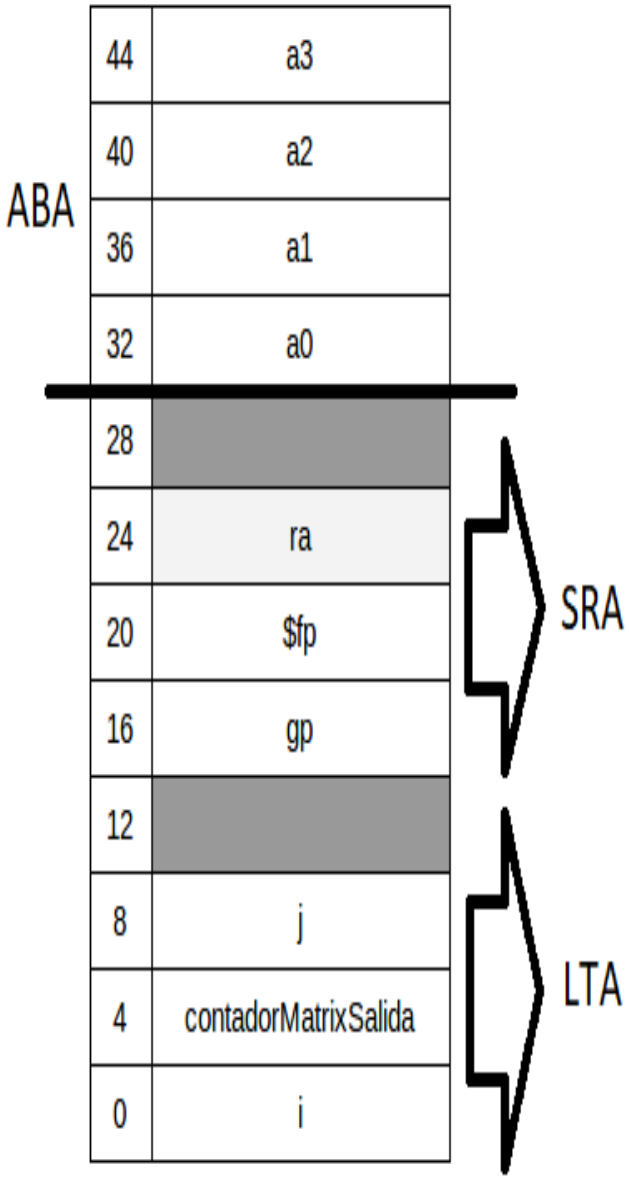
```
$ ./testing.sh
```

es necesario tener las dos versiónes compiladas.

4. Diagramas

Para mostrar cómo funciona el código Assembly, hay unos diagramas del *stack*.

La función trasponer en assembler, tiene la siguiente ABI.



El stack de la función transponer tiene un total de 32 bytes, los parametros a0, a1, a2 y a3 son obtenidos desde el stack del main.

ABI de la función main es la siguiente:

68	
64	ra
60	$fp
56	gp
52	&matrix
48	cantColumnas
44	cantFilas
40	ingresoGuion
36	&outputFile
32	&inputFile
28	&long_opt
24	&short_opt
20	&file
16	option
12	
8	
4	argv
0	argc

El stack de main tiene un total de 72 bytes.

5. Conclusión

En este trabajo tuvimos que hacer dos verciones de un programa; el primero sólo con C, el segundo con una parte en Assembly. Descubrimos lo que ya sabíamos, que la programación en Assembly lleva mucho más tiempo y da un código más largo. No obstante, esto también se le permite entender mejor cómo funciona la función y qué hace la computadora cuando está con un programa.

6. Código fuente

6.1. Versión C

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <getopt.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

#define ERROR -1
#define SALIDA_EXITOSA 0
FILE* validacionDeArchivoEntrada(int argc, char *argv
    [], bool ingresoGuion){
        FILE *auxi = NULL;
        auxi = fopen( argv[argc - 1], "r" );
        if(auxi == NULL){
                fprintf(stderr, "Error en el archivo
                    Entrada: %s\n", strerror(errno));
                return NULL;
        }
        return auxi;
}
void guardarMatriz(long long *matrix, int cantColumnas,
    int cantFilas, FILE *fp){
        fprintf( fp, "%i %i\n", cantFilas ,cantColumnas
            );
        int i=0;
```

```
        int contadorColumna = 0;
        for (i = 0; i < (cantColumnas*cantFilas); i++){
                if(contadorColumna == cantColumnas){
                        fprintf( fp, "\n");
                        contadorColumna = 0;
                }
                fprintf( fp, "%lli", matrix[i] );
                contadorColumna++;
                if(contadorColumna != cantColumnas){
                        fputc( ' ', fp );
                }
        }
        if ( NULL != fp ) fputc( '\n', fp);
        else puts( "\n" );
}
int transponer(unsigned int filas, unsigned int
   columnas, long long *entrada, long long *salida){
        int i = 0;
        int contadorMatrixSalida = 0;
        for(i = 0; i < columnas; i++){
                int j = 0;
                for(j = 0; j < filas; j++){
                        salida[contadorMatrixSalida] =
                           entrada[j*columnas+i];
                        contadorMatrixSalida++;
                }
        }
        return SALIDA_EXITOSA;
}

int llenarFila(char *fila,long long *vectorFila,int *
   cantColumnas,int contadorFilas){
        int contadorColumna = 0;
        char *auxi = malloc((*cantColumnas)*sizeof(char
           ));
        char *eptr;
        int contadorAuxi = 0;
        int contadorFila = 0;
        int tamanioFile = strlen(fila);
        while(contadorFila <= tamanioFile){
```

```
            if(fila[contadorFila] == ' ' ||
               contadorFila == tamanioFile){
                    auxi[contadorAuxi] = '\0';
                    long long numero = strtoll(auxi
                       , &eptr, 10);
                    if (*eptr){
                         fprintf(stderr,"Error de
                            conversion en fila: %d ,
                            columna: %d\n",
                            contadorFilas,*
                            cantColumnas);
                         return ERROR;
                    }
                    if(contadorAuxi > 0){
                             vectorFila[
                                contadorColumna] =
                                numero;
                             contadorColumna++;
                             contadorAuxi = 0;
                    }
                    if(contadorColumna > (*
                       cantColumnas)){
                             free(auxi);
                             fprintf(stderr, "Error:
                                exceso de columnas
                                en fila %d\n",
                                contadorFilas);
                             return ERROR;
                    }
            }else{
                    auxi[contadorAuxi] = fila[
                       contadorFila];
                    contadorAuxi++;
            }
            contadorFila++;
    }
    if(contadorColumna < (*cantColumnas)){
            free(auxi);
            fprintf(stderr, "Error: faltan columnas
               en fila %d\n",contadorFilas);
            return ERROR;
```

```
        }
        free(auxi);
        return SALIDA_EXITOSA;
}

long long *parsearMatriz(FILE *inputFile, int *
   cantFilas, int *cantColumnas){
        fscanf(inputFile,"%d %d\n", cantFilas,
           cantColumnas);
        long long *matrix = malloc(((*cantFilas)*(*
           cantColumnas))*sizeof(long long));
        int contadorMatrix = 0;
        int contadorFilas = 0;
        char *fila = NULL;
        int contadorCadenaFila = 0;
        char caracter = fgetc(inputFile);
        while (!feof(inputFile)){
                while((!feof(inputFile)) && (caracter!=
                   '\n')){
                        if(fila == NULL){
                                fila = malloc((
                                   contadorCadenaFila
                                   +1)*sizeof(char));
                                fila[contadorCadenaFila
                                   ] = caracter;
                        }else{
                                fila = realloc(fila, (
                                   contadorCadenaFila
                                   +1)* sizeof(char));
                                fila[contadorCadenaFila
                                   ]=caracter;
                        }
                        contadorCadenaFila++;
                        caracter = fgetc(inputFile);
                }
                if(fila != NULL){
                        contadorFilas++;
                        if(contadorFilas > (*cantFilas)
                           ){
                                fprintf(stderr, "Error:
                                   exceso de filas\n")
```

```
                                        ;
                                return NULL;
                        }

                        fila = realloc(fila, (
                            contadorCadenaFila+1)*
                            sizeof(char));
                        fila[contadorCadenaFila]='\0';

                        long long *vectorFila = malloc
                            ((*cantColumnas)*sizeof(long
                             long));
                        if(llenarFila(fila,vectorFila,
                            cantColumnas,contadorFilas)
                            == ERROR){
                                return NULL;
                        }
                        int i= 0;
                        for(i = 0; i < *cantColumnas;i
                            ++){
                                matrix[contadorMatrix]
                                    = vectorFila[i];
                                contadorMatrix++;
                        }
                        free(vectorFila);
                        free(fila);
                }
                fila = NULL;
                contadorCadenaFila = 0;
                if(!feof(inputFile)){
                        caracter = fgetc(inputFile);
                }
        }
        if(contadorFilas < (*cantFilas)){
                fprintf(stderr, "Error: faltan filas\n"
                    );
                return NULL;
        }
        return matrix;
}
```

```
int main(int argc, char *argv[])
{
    int option = 0;
        char *file;
    const char *short_opt = ":o:hV";
    struct option long_opt[] = {
            {"version", no_argument,        NULL, 'V'},
            {"help",    no_argument,        NULL, 'h'},
            {"output",   optional_argument, NULL, 'o'},
            {NULL, 0,                       NULL, 0}
    };
    FILE *inputFile = NULL;
    FILE *outputFile = NULL;
    bool ingresoGuion = false;
    while ((option = getopt_long(argc, argv, short_opt,
        long_opt, NULL)) != -1) {
        switch (option) {
                case 'V':
                        printf("TP #0 de la materia
                            Organizacion de Computadoras
                            \n");
                        printf("Alumno: \n");
                        printf("        Quino Lopez
                            Julian \n");
                        printf("        Lund Eirik
                            Harald \n");
                        return SALIDA_EXITOSA;
                case 'h':
                        printf("Usage: \n");
                        printf("        %s -h \n", argv
                            [0]);
                        printf("        %s -V \n", argv
                            [0]);
                        printf("        %s [options]
                            file\n", argv[0]);
                        printf("Options: \n");
                        printf("        -V, --version
                            Print version and quit. \n")
                            ;
                        printf("        -h, --help
                            Print this information. \n")
```

```
                ;
            printf("        -o, --output
                Path to output file. \n");
            printf("\n Examples: \n");
            printf("        %s - mymatrix \
                n", argv[0]);
            printf("        %s -o - 
                mymatrix \n", argv[0]);
            return SALIDA_EXITOSA;
        case 'o':
            if(strcmp(optarg, "-") != 0){
                    file = optarg;
                    outputFile = fopen(
                        file, "w" );
                    if (outputFile == NULL)
                        {
                            fprintf(stderr,
                                "Error 
                                archivo 
                                Salida: %s\n
                                ", strerror(
                                errno));
                            return ERROR;
                    }
            }
            else{
                    ingresoGuion = true;
                    printf ("salida por 
                        stdout\n");
            }
            break;
        default:
            fprintf(stderr, "Parametro 
                invalido, use el comando -h\
                n");
            return ERROR;
            break;
    }

}
```

```
    inputFile = validacionDeArchivoEntrada(argc,argv,
       ingresoGuion);
    if(inputFile==NULL){
        return ERROR;
    }
    int cantFilas = 0;
    int cantColumnas = 0;
    long long *matrix = parsearMatriz(inputFile,&
       cantFilas, &cantColumnas);
    if(matrix==NULL){
        return ERROR;
    }
    long long *matrixSalida = malloc((cantFilas*
       cantColumnas)*sizeof(long long));

    transponer(cantFilas, cantColumnas, matrix,
       matrixSalida);

        if ( NULL == outputFile ) outputFile = stdout;
        guardarMatriz(matrixSalida, cantFilas,
           cantColumnas, outputFile);

        if(outputFile != stdout){
                fclose(outputFile);
        }

        fclose(inputFile);
    free(matrix);
    free(matrixSalida);
    return SALIDA_EXITOSA;
}
```

6.2. Código Assembly

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

#Argumentos de la funcion
#define VALIDATE_ARG3                    44
#define VALIDATE_ARG2                    40
#define VALIDATE_ARG1                    36
```

```
#define VALIDATE_ARG0                    32

#tamanio Stack
#define VALIDATE_TS                              32

#SRA
#define VALIDATE_RA                      24
#define VALIDATE_FP                              20
#define VALIDATE_GP                              16

#LTA
#define VALIDATE_J                               8
#define VALIDATE_CONT_MATRIZ     4
#define VALIDATE_I                               0

        .text
        .align  2
        .globl  transponer
        .ent    transponer
transponer:
        sw              ra  ,VALIDATE_RA(sp)
        sw              $fp  ,VALIDATE_FP(sp)
        sw              gp  ,VALIDATE_GP(sp)

        sw              a0          ,VALIDATE_ARG0(sp)
        sw              a1          ,VALIDATE_ARG1(sp)
        sw              a2          ,VALIDATE_ARG2(sp)
        sw              a3          ,VALIDATE_ARG3(sp)

        move            $fp  ,  sp

        sw              zero ,VALIDATE_I($fp)
        sw              zero ,VALIDATE_CONT_MATRIZ($fp)
        sw              zero ,VALIDATE_J($fp)
forPrincipal:
        lw              t0 ,VALIDATE_I($fp)
        lw              t1 ,VALIDATE_ARG1($fp)
        bge             t0 ,t1 ,transponerMipsFin

        sw              zero ,VALIDATE_J($fp)
```

```
forSecundario:
        lw              t0,VALIDATE_J($fp)
        lw              t1,VALIDATE_ARG0($fp)
        bge             t0,t1,salirDeForPrincipal

        lw              t0,VALIDATE_ARG1($fp)
        lw              t1,VALIDATE_J($fp)
        sll             t1,t1,3
        mul             t1,t1,t0
        lw              t0,VALIDATE_I($fp)
        sll             t0,t0,3
        addu            t1,t0,t1
        lw              t0,VALIDATE_ARG2($fp)
        addu            t1,t1,t0
        lw              t3,0(t1)
        lw              t4,4(t1)

        lw              t0,VALIDATE_CONT_MATRIZ($fp)
        sll             t0,t0,3
        lw              t1,VALIDATE_ARG3($fp)
        addu            t0,t1,t0

        sw              t3,0(t0)
        sw              t4,4(t0)
        lw              t0,VALIDATE_CONT_MATRIZ($fp)
        addu            t0,t0,1
        sw              t0,VALIDATE_CONT_MATRIZ($fp)

        lw              t0,VALIDATE_J($fp)
        addu            t0,t0,1
        sw              t0,VALIDATE_J($fp)
        b               forSecundario

salirDeForPrincipal:
        lw              t0,VALIDATE_I($fp)
        addu            t0,t0,1
        sw              t0,VALIDATE_I($fp)
        b               forPrincipal

transponerMipsFin:
        li              v0,1
```

```
        lw              ra  ,VALIDATE_RA($fp)
        lw              gp  ,VALIDATE_GP($fp)
        lw              $fp  ,VALIDATE_FP($fp)
        addu                    sp,sp,VALIDATE_TS
        j               ra
.end transponer
```