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

**Ver documentos digitales.*

8.1. main.c

8.2. transpose.c

8.3. transpose.S

8.4. Makefile

8.5. pruebaScript.sh

9. Enunciado

**Ver documentos digitales.*