Organización del Computador

Facultad de Ingeniería, Universidad de Buenos Aires

Trabajo práctico 1: Programación MIPS

Padrón	Alumno	Dirección de correo
101109	Camila Dvorkin	camidvorkin@gmail.com
99139	Fernando Sinisi	fersinisi@gmail.com
97649	Rodrigo Souto	rnsoutob@gmail.com

Segundo Cuatrimestre de 2019

Práctica Martes

Índice

1. Introducción

Se propone escribir un programa cuya función principal se realice en C y desde allí se llame a subrutinas programadas en código MIPS32 que se encarguen de multiplicar matrices cuadradas de números reales, representados en punto flotante de doble precisión. Se utiliza el programa GXemul para simular el entorno de desarrollo, una máquina MIPS que corre una versión de NetBSD. Esto mismo ya fue realizado en el Trabajo Práctico 0 pero la novedad radica en implementar las función *matrix_multiply* en MIPS32.

2. Documentación relevante al diseño e implementación del programa

2.1. Comandos

Se cuenta con diferentes comandos que facilitan el uso del programa:
`-V, -version` Imprime la version y cierra el programa.
`-h, -help` Despliega el menú de ayuda y cierra el programa.
`<in_file >out_file,` Resuelve el problema planteado para los valores extraidos de *in_file* y devuelve los resultados en el archivo correspondiente.

2.2. Diseño

El programa principal se encuentra desarrollado en la función main en lenguaje C. Las matrices a multiplicar ingresarán como texto por entrada estándar (stdin). Por cada par de matrices que se presenten por cada l´ínea de entrada, el programa las multiplicará y presentará el resultado por su salida estándar (stdout). Ante un error, el progama deberá informar la situación inmediatamente por stderr y detener su ejecución.

Las acción de *multiply_matrix* se separa e implementa siguiendo una lógica similar. La misma se encuentra en el archivo *multiply.S* .

Se puede encontrar el código del programa en el repositorio:
https://github.com/camidvorkin/OrganizacionDelComputador

2.3. Detalles de implementación

2.3.1. multiply.S

Para diseñar la función *multiply_matrix* en mips32 se fue evaluando y separando la función usada en el TP0 (en lenguaje C) en partes para poder trabajar mas cómodos. Luego de lograr esto y concluir que no usariamos las areas LTA y FRA del stack frame comenzamos a definir que uso le daremos a los registros temporales. Al comienzo del codigo de multiply_matrix creamos el stack frame de 32 bytes (8 registros de 4 bytes) y obtenemos los registro a0 y a1 donde están los punteros a las matrices a multiplicar. Siguiente a esto se llama a la función create que nos devuelve en el registro v0 el puntero a la matriz que usaremos como resultado. Esta función está implementada en el archivo create.S y encargandose de calcular el tamaño de memoria a pedir para la creacion de la matriz resultado y utilizando myMalloc y myFree provistos por la catedra. Si la función devolviera null se corta el flujo y se llama a la etiqueta matrix_multiply_return que se encarga

de recuperar los registros, destruir el stack frame y volver a la funcion (o sistema operativo) de donde se la llamó. En caso contrario, para seguir el diseño del codigo en .C, se diseñaron los 3 ciclos necesarios para realizar la multiplicación de las matrices, utilizando los registros temporales para el manejo de los mismos y los registros de punto flotantes para el manejo de los valores de la matrices. Finalmente cuando se termina el flujo de los bucles se termina llamando a la misma etiqueta mencionada antes matrix_multiply_return para finalizar la ejecucion de la función.

3. Comandos para compilar el programa

En esta sección se detallan los pasos para compilar el programa en NetBSD a partir del entorno proporcionado por GXemul.

- Desde el directorio donde se instaló GXemul se corre el siguiente comando para bootear la imagen del disco patrón:
  `hostOS ./gxemul -e 3max -d netbsd-pmax.img`
- Desde otra consola de linux se crea en el host OS con el usuario root un alias para la interfaz loopback (lo:0) con la IP 172.20.0.1 con el siguiente comando:
  `hostOS ifconfig lo:0 172.20.0.1`
- Luego se ejecutan los siguientes comandos para la conexión contra la interfaz creada:
  `hostOS export TERM=xterm`
  `hostOS ssh -p 2222 root@127.0.0.1`
- Se transfieren los archivos a compilar a NetBSD con los siguientes comandos:
  `scp -P2222 -r TP1 root@127.0.0.1:/root`
- Luego se ejecutan los siguientes comandos para realizar la compilación y extraer el código MIPS generado por el compilador en el sistema operativo que corre sobre GXemul:
  `root@: ls`
  `root@: pwd`
  `root@: mkdir TP1`
  `root@: cd TP1`
  `root@: make mips`
  – -s para detener el compilador luego de generar el código assembly. – -mrnames para cambiar los números de los registros por sus nombres de convención.
- Luego se transfiere el archivo .s para el sistema operativo sobre el cual corre GXemul:
  `hostOS scp -P2222 root@127.0.0.1:/root/TP0NetBSD/TP0/*.s /home/user`
- Finalmente, se compila el código del programa usando el comando `$ make`

4. Corrida de Pruebas

Las pruebas realizadas se basaron en los ejemplos del enunciado. Se probaron los comandos básicos como -h y -V para probar que muestren el resultado esperado. Luego, se realizaron pruebas con diferentes problemas que podrían resultar tras leer los archivos:

- Se probó que la cantidad de argumentos pasados sea correcto. Al ejecutar:

```
./TP1 ejemplo-simple.txt
```

Al no contar con un archivo de salida, imprime en la salida de error: *Error: Cantidad de parametros erronea*

- Se probó que el archivo de entrada ingresado exista. Al ejecutar:

```
./TP1 ejemplo1.txt salida.txt
```

Como el archivo ejemplo1.txt no existe, se imprime en la salida de error: *Error: El archivo que se desea abrir no existe*

Se probaron diferentes errores que podían surgir al multiplicar los valores que se ingresaron:

- Se probó que el primer valor no sea un entero, ya que se trata de la dimensión. Algunos ejemplos de matrices que contienen fallas que fueron testeadas son:

```
m 1 2 3 4 5 6 7 8
2.2 1 2 3 4 5 6 7 8
* 1 2 3 4 5 6 7 8
```

En consecuencia, se imprime en la salida de error que no se cuenta con un largo adecuado para generar una matriz.

- Se probó que la cantidad de números en una linea coincida con la cantidad de valores que debe tener la matriz. Algunos ejemplos de este estilo:

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

En este caso, se imprime un Error en relación a la cantidad de elementos ingresados.

- Se probó que se que los valores ingresados sean únicamente números reales, representados en punto flotante de doble precisión. Se probaron los ejemplos implementados por la cátedra en el enunciado del trabajo y algunos otros más:

```
2 1 2 3 4 5 6 7 8
2 11 12 13 14 15 16 17 18

3 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
2 1 2 3 4 5 1e-4 7 8
```

Se contó con un Makefile con el objetivo de agilizar la tarea de compilar código. Los comandos más importantes fueron:

- Ejemplo corrida con casos exitosas: `$ make test-successful`
- Ejemplo corrida con casos fallidos: `$ make test-failure`
- Ejemplo corrida con casos exitosas en netBsd: `$ make test-successful-mips`
- Ejemplo corrida con casos fallidos: `$ make test-failure-mips`
- Corrida de Valgrind para evitar la perdida de memoria: `$ make valgrind`

  Corridas de test agregadas en el Anexo 1

5. Código en C

Agregado en el Anexo 2

6. Código en MIPS32

Agregado en el Anexo 3

7. Makefile

Agregado en el Anexo 4

8. Conclusiones

En conclusión, se cumplió el objetivo del Trabajo Práctico ya que se desarrolló el programa detallado por el enunciado. Los comandos del programa se ejecutan como detalla el comando -h y tras ser sometidos a distintas pruebas se concluye que funcionan según lo esperado. Así mismo fue posible la implementación del código en MIPS32 y se pudo utilizar el programa GXemul para simular un entorno de desarrollo de una máquina MIPS corriendo el sistema operativo NetBSD para ejecutarlo.