

DIARIO DE PRÁCTICAS

**Luis Blanco de la Cruz
Roberto Gozalo Andrés**

Primera Práctica 19/09/18

Ejercicio 1:

a)

En el simulador Mars, tras ejecutar nuestro programa vemos que es Little Endian porque en la dirección más baja del registro se guarda el bit más significativo, en este caso el 0, si fuera Big Endian se guardaría un 1. Ambas instrucciones son sintéticas, porque el compilador las cambia por otras básicas.

b)

Después de muchos errores, conseguimos compilar y ejecutar el programa correctamente.

c)

En nuestro programa utilizamos estas instrucciones:

-li: La cambia por addi.

-la: La cambia por lu+ori.

-b: La cambia por bgz.

-lw: Cuando se utiliza una etiqueta se transforma en lui.

d)

Todas las instrucciones sintéticas las marca como errores. Si, ahora el programa es mucho más difícil de leer que antes.

Ejercicio 2:

a)

No se traducen las instrucciones de la misma manera. la instrucción li se traduce como addiu en Mars y como ori en QtSPIM.

b)

bne no nos la marca como instrucción sintética pero probamos con beq y en Mars la traduce como addi+beq y en QtSPIM como ori+beq. Ambas son correctas, la traducción depende de cómo esté escrito el compilador de cada programa.

Ejercicio 3:

Como pone en el guión, al marcar la casilla de Bkpt se para el programa, si se marca en la casilla de un bucle el programa se para en cada iteración.

Segunda Práctica 26/09/18

Ejercicio 1:

Como se indica en el guión vamos a crear un programa para realizar el siguiente cálculo sobre matrices cuadradas:

$$B[i][j] = A[i][j] - A[j][i]$$

Dicho programa tiene como argumentos las direcciones de ambas matrices así como el número de filas de ellas (ya que son matrices cuadradas no necesitamos conocer el número de columnas porque es igual al de las filas).

Para comenzar con nuestro programa decidimos crear una función que hemos llamado calcula y que realizará el cálculo de la dirección

$A[i][j]$ aplicando la fórmula obtenida en el guión.

$$A[i][j] = A + 4 * m * i + 4 * j$$

Durante la creación de dicha porción de código nos damos cuenta que nuestra función se puede optimizar sacando factor común quedando de la siguiente manera

$$A[i][j] = A + 4 * (m * i + j)$$

Al principio nuestro programa no funciona correctamente, se nos había olvidado la gestión de la pila. Para solucionar esto primero debemos decrementar el valor de la pila, ya que esta apunta a la dirección más alta y antes de salir de la función devolver el apuntador a la dirección más alta.

Para recorrer las funciones necesitamos dos bucles for anidados, al primero lo llamaremos loopI y al otro loopJ. Tras pasar el profesor por nuestro puesto y solucionarnos algunas dudas que nos habían surgido conseguimos que se recorran las funciones correctamente. Lamentablemente la sesión de laboratorio llega a su fin y debemos de pausar nuestro trabajo.

Ya en nuestras casas y mediante videoconferencia retomamos el trabajo donde lo habíamos dejado en clase.

Nos damos cuenta de que nuestro programa no realiza el cálculo correctamente, ya que nos aparecen los valores correctos multiplicados por 4.

Tras revisar nuestro código nos damos cuenta de que no estábamos almacenando \$t2 y \$t3. Almacenamos mediante la instrucción lw dichos registros en \$t6 y \$t7 respectivamente. Ahora si nuestro programa realiza la operación correctamente para matrices de 4*4.

28/09/18

Nos ponemos ahora con la función que imprimirá la matriz, a la que llamamos

funcionB en previsión de que la que pida la matriz por pantalla sea la A.

Creamos dos cadenas ascii para imprimir un espacio y una tabulación para que quede mejor la matriz al imprimir.

Creamos dos bucles que iterarán hasta el tamaño de matriz que tengamos introducido en \$a2, así podemos cambiar el tamaño más fácilmente cambiando solo un registro. Cogemos el código de la función calcula para calcular la posición a imprimir y lo introducimos en el bucle. Añadimos bifurcaciones para que imprima tabulaciones o saltos de línea cuando toque.

Al probarlo nos imprimía la matriz correctamente tabulada pero solo con 0s. Después de revisar el código, nos dimos cuenta que estábamos imprimiendo B, que estaba vacía ya que solo ejecutábamos la función de imprimir y no la de calcular. Al añadirla al bucle, ya funcionaba correctamente e imprimía los valores adecuados.

29/09/18

Nos disponemos a hacer la función de pedir los datos por pantalla. Después de echar un ojo a la referencia de mips conseguimos que almacene el dato introducido en la matriz.

Ahora creamos otro bucle y volvemos a copiar el código para calcular la posición en la que introducir el dato. Después de un par de errores por repetir registros en varias funciones, vemos que funciona correctamente.

Decidimos poner un poco más bonito la parte de introducción de datos y añadimos otras cadenas ascii. Ahora al pedirte los datos te mostrará un mensaje y la posición en la que introduces los datos; así será más fácil saber donde pones cada número.

Tercera Práctica 03/10/18

Ya que en el segundo ejercicio vamos a emplear la función del primero decidimos comentar nuestro trabajo a partir del segundo ejercicio, que incluye todo lo realizado en el primero.

Primero vamos a solicitar al usuario que introduzca un número en decimal, que

como se indica en el gui3n de la pr3ctica almacenaremos en \$a1 y despu3 guardamos el valor de la m3scara que utilizaremos en la funci3n con el valor 0xF0000000. Tambi3n guardamos un 0 con la instrucci3n lw y una X con la instrucci3n lb ya que los n3meros en hexadecimal vienen siempre precedidos por 0x.

Despu3s realizamos un bucle que se repite tantas veces como d3gitos tiene el n3mero en hexadecimal (8 en concreto) dentro del bucle se realiza la operaci3n and con la m3scara para poder obtener los 4 bits m3s significativos y despu3s son desplazados 28 posiciones a la derecha. Comprobamos despu3s, si ese n3mero es mayor

o igual a 10, de ser as3 se le suma 55 de esta manera, mediante el c3digo Ascii somos capaces de representar las letras de la A a la F. Sino, se le suma 48, para poder representar los d3gitos del 0 al 9.

Guardamos ahora mediante la instrucci3n lb el n3mero o la letra obtenido e incrementamos la direcci3n de guardado. Despu3s desplazamos hacia la izquierda 4 posiciones para poder coger los siguientes bits en la siguiente iteraci3n. Cuando se termina el bucle se a3ade el fin de carrera al n3mero en formato hexadecimal.

Tras salir de la funci3n se imprime el n3mero mediante la instrucci3n li \$v0, 4

Como se indica en el gui3n vamos ahora a probar con los diferentes datos que nos indican, tras realizar las pruebas y comprobar los resultados con la ayuda de un conversor de decimal a hexadecimal, podemos comprobar que nuestro programa funciona correctamente.

06/10/18

Vamos a modificar nuestro programa para que ahora el n3mero mostrado en pantalla sea el resultado de multiplicar el n3mero introducido por el usuario por 4 y convertido a formato hexadecimal. Decidimos utilizar la funci3n del apartado 1 y simplemente meter una multiplicaci3n por 4 dentro de la propia funci3n.

Para ello y tras corregir varios fallos, finalmente a3adimos en nuestra funci3n la instrucci3n sll \$t1,\$t1,2 de esta manera desplazando a la izquierda dos posiciones multiplicamos por 4 el n3mero introducido por el usuario.

Siguiendo con el gui3n vamos ahora a probar los n3meros del apartado 2b observamos que con los 33ltimos n3meros, tras realizar la multiplicaci3n por 4,

se obtiene un número que no puede ser representado en hexadecimal con 8 dígitos. En principio, el resto de números en principio correctos .

Ejercicio 3

Continuando con el guión de la práctica vamos ahora a activar las opciones correspondientes en Mars.

Tras activar las diferentes opciones que se especifican en el guión vemos como Mars nos muestra la ruta de datos. En ella aparecen varios colores. Rosa op code, verde rs, azul rt, azul claro desplazamiento (aunque también el desplazamiento va por otro cable al ALU Control), amarillo PC y el 4 que se suma al PC, en naranja rd, en gris y rojo las señales de control dependiendo de si están en desactivadas o activadas respectivamente, en un marrón anaranjado el cable que va por el sign extended al shift left y al add, también va a la ALU si es una instrucción donde la ALU necesite el desplazamiento. Por arriba tenemos varios cables que se utilizan para determinar si es una instrucción jump o de bifurcación y el cable marrón oscuro es el que lleva el PC+4. Por último el cable azul oscuro de abajo que sale de la ALU y si no es una instrucción lw o sw pasa por el multiplexor y llega al write data del banco de registros, sino es un cable negro que sale de la memoria de datos y llega al mismo sitio pero no escribe nada en el banco de registros

Los caminos críticos son:

R: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux
lw: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux +
Memoria de datos
sw: Memoria de instrucciones + Banco de registros + Mux + ALU + Mux
Bifurcación: Memoria de instrucciones + Banco de registros + Mux + ALU +
Mux

Después de ver todos los caminos críticos, no se nos ha ocurrido ninguna representación más optima.