**ARQUETECTURA Y ORGANIZACIÓN DE COMPUTADORES – Diario de prácticas**

Daniel de Vicente Garrote

Daniel Crespo Martínez

***PRÁCTICA 1***

13/09/16: *1.a)* Para realizar el primer apartado del ejercicio 1, necesitamos crear un código en ensamblador MIPS de 32 bits que sume a un vector de enteros los valores de otro vector que, en este caso, sólo contiene un número. Tal y como indicamos en los comentarios del código, asignamos los siguientes valores a las variable: #$s0 para f, $s1 para G, $s2 para H, $s3 y $s4 para direcciones de memoria, $s6 y $s7 para controlar el bucle, $t0 la dirección de G. Recorremos el vector para ir sumando el número fijo del vector F a cada posición de G mediante un bucle.

14/09/16: *1.b) y 1.c)* Buscamos las instrucciones sintéticas y las sustituimos por sus equivalentes(para el apartado d):

* la: Se utiliza para poder utilizar la matriz en un registro y poder recorrerla con facilidad (sumamos 4 a la dirección para pasar al siguiente elemento).
  + La instrucción la $s3,F se traduce como

lui $s3, 0x1001

ori $s3,$s3, 0x0000

* li: Se utiliza para imprimir enteros y en la parte final del programa, para terminarlo.
  + li se traduce como

lui $v0, 0x0000

ori $v0,$v0,0x0001

*1.d)* Si se desactiva la opción de permitir pseudoinstrucciones la línea de la $s3,F tira un error de sintaxis debido a que es una pseudoinstrucción (ocurriría lo mismo con la $s4,G).

*2.b)* La versión sin pseudoinstrucciones necesita unos pocos arreglos para funcionar. Tras realizar el programa, utilizamos el simulador mars, que se ve de forma similar al qtspim, y tiene funciones casi idénticas. Cargamos el programa con y sin pseudoinstrucciones, si desactivamos la opción de permitir pseudoinstrucciones en mars también tira la excepción de antes (no permite pseudoinstrucción).

*2.c)* Las traducciones de las pseudoinstrucciones entre Qtspim y Mars son ligeramente diferentes, pues en Qtspim falta alguna instrucción por traducir que solo aparece en mars (ori $19,$1,0 no está presente en Qtspim),ademas beq presenta una ligera diferencia con los números al final de la instrucción. La traducción más correcta debería ser la de mars, ya que dicha pseudoinstrucción se divide en dos instrucciones, pero QtSpim ignora una instrucción del primer la.

20/9/2016

***Práctica 2***

1.a)Cargamos el número 1 en forma de byte y leemos el primer elemento para comprobar si es Little-Endian o Big-Endian. Como el resultado es un 1, es little-endian.

Comprobamos mediante la ventana de registros que aparece el valor 1 en su dirección correspondiente.

1.b)Tanto QTSpim como Mars siguen el criterio little-endian.

1.c)Los valores escritos en hexadecimal los almacena en la memoria en complemento a 2, concretamente se almacena el signo en el primer bit cuando esté escrito en binario.

1.d)En precisión simple, los números en punto flotante se guardan de la siguiente manera: 1 bit para el signo, 8 para el exponente y 23 para la mantisa.

En precisión doble, corresponden: 1 bit para el signo, 11 para el exponente y 52 para la mantisa.

1.e)Lo comprobamos a través de la simulación en Mars.

2.a)Para comenzar, calculamos todos los valores de los elementos de la matriz y los declaramos por filas en los vectores correspondientes. Cargamos los valores en sus registros.

Creamos un bucle que sume el primer elemento de cada vector y lo desplazamos.

2.b)Por columnas se realiza igual, cambiando el valor de los vectores en filas por columnas.

2.c)La diferencia a la hora de almacenar en la memoria es el orden en el que se guardan los valores. En el primer caso

se almacenan en el mismo orden de las filas; en el segundo, en el orden de las columnas.

2.d)Cuando se guardan los valores como byte,estos solo ocupan 8 bits de espacio,lo que permite almacenar dichos valores en menos direcciones.

21/9/2016

3.a)En la versión de bytes se utiliza se utilizan menos números para almecenar los datos, por lo tanto sólo se utilizan 4 direcciones de memoria.

Ahora cargamos el primer programa y examinamos con la opción MIPS X-RAY,la cual muestra un mapa del procesador,que incluye el PC,las instrucciones de memoria,los registros y la memoria de datos,entro otros componentes.

La instrucción primera (la $t0,Prueba) manda a la instrucción de memoria dicha instrucción,lo cual hace que envíe una señal a dichos registros para poder escribir en el registro $t0 la dirección de memoria de Prueba,luego desde el banco de memoria se envía dicha dirección al registro,utilizando los registros de control RegDst,WriteReg y opALU1.

La instrucción lb $t0,0($t0) carga el valor de la dirección el el mismo registro $t0,para ello carga desde el banoc de memoria y lo añade a dicho registro,activando los controles AULSrc,MemToReg,WriteReg y ReadMem.

La instrucción lógica bne $t0,$zero,Little comprueba el valor del byte $t0 con el valor cero y si son iguales no bifurca,pero si los son bifurca el programa a la etiqueta Little. Activa los controles Branch y opALU0(ya que se cumple la condición y por lo tanta real¡iza dicha bifurcación.

La instrucción la $a0,LE realiza algo parecido a la anterior la,activando los controles RegDst,WriteReg y opALU1.

La instrucción li $v0 4 carga la instrucción de imprimir cadena,la cual activa los controles RegDst,WriteReg y opALU1,además activa el control de escritura de los registros,igual que las otras instrucciones de la y lb,y envia señal al registro $v0(el resto no reciben la señal y no generan salida).

La función syscall simplemente ejecuta la instrucción codificada como un número en $v0,en este caso imprime la cadena en pantalla.

Las dos últimas instrucciones son prácticamente las mismas que antes(un li y un syscall),así termina el programa.

El color de los cables representan si se envian señales o no,y el tipo de estas señales:

-Negro:No envía señal

-Rojo:Envía señal efectiva

-Morado:Señal de opcode

-Verde:Señal de rs(dirección destino)

-Azul:Señal de rt(señal origen)

-Azul claro:Señal de rd(otra señal de origen)

-Negro(shamt):Para diferenciar diferentes instrucciones de misma opcode

-Naranja:Señal de function(valor inmediato)

3.b)El camino crítico que recorre dicho programa pasa por PC->instrucciones de memoria->Shift left 2->MUX->Vuelta a PC,es el recorrido de mas latencia según la herramienta.

3.c)La representación que ofrece dicha herramienta no es mala,pues muestra como y por donde pasan las señales a través de los diferentes circuitos con el objetivo de ejecutar la instrucción correspondiente,sin embargo los tiempos que tardan en llegar la señal no son del todo precisos,además si mostramos un componente con mas detalles este debe realizar el recorrido de forma independiente a la ventana principal,lo que hace mas díficil interpretar el resultado.

***Práctica 3***

30/9/2016

Para el primer programa lo que hacemos es crear una máscara 0x0000000F que con la operación AND coja los primeros 4 bits del número,si este es un número(menor o igual que 9),le suma 48;y si es letra(mayor que 9) le suma 55,así tenemos un carácter en hexadecimal,desplazamos la máscara 4 bits para coger el siguiente elemento,así hasta pasar por los 32 que tiene,y almacenamos el resultado en $a1

Para los del ejercicio 2 solo tenemos que reemplazar el valor por defecto en $a0 por un read int en decimal que moveremos a ese registro y realizamos la misma operación que en el ejercicio anterior,pero esta vez imprimimos cada número como char para obtener el equivalente ASCII

(Error:Aunque toma los valores ASCII,algunos números imprimen símbolos que no son ni letras ni números,probablemente debido a sumas incorrectas)

Realizamos el mismo programa pero devuelve el resultado multiplicado por 16,que se puede multiplicar la entrada por dicho valor,al ser una simple traducción

***PRÁCTICA 4***

4/10/2016

1)Ahora necesitamos diseñar una función que coja una cadena de caracteres que representen un número en hexadecimal,lo convierta en número,lo pase a binario y lo almacene en $v0.

Para la función debemos coger los 8 primeros bits de la cadena,los cargamos en un registro,los restamos para ajustarlo a un número o letra y desplazamos dicho registro para poder poner el siguiente valor en hexadecimal.

La función debe ser así:

Función:

j Binario

Binario:

lb $s1,0($a0) #Carga los 8 primeros bits de la cadena

slti $s2,$s1,58 #Comprueba si es número o letra

beq $s2,1,Numero

beq $s2,0,Letra

Numero:

addi $s1,$s1,-48 #Si es número resta 48

or $v0,$s1,$v0 #Almacenamos los bits en $v0,incluyendo resultados anterioes

addi $s0,$s0,-1 #Resta el contador

beq $s0,0,Fin #Si llega a cero termina la funcion

rol $v0,$v0,4 #Rotamos los siguientes 8 bits para ir sumando

addi $a0,$a0,1 #Movemos a los siguientes 8 bits de la cadena

jr $ra #Retorna al bucle

Letra:

addi $s1,$s1,-55

or $v0,$s1,$v0

addi $s0,$s0,-1 #Resta el contador

beq $s0,0,Fin #Si llega a cero termina la funcion

rol $v0,$v0,4

addi $a0,$a0,1

jr $ra

2.a)Ahora creamos un programa entero que utilice dicha función para obtener un número en binario,pero que lo imprima en decimal por pantalla.Para ello utilizamos la función,pero debemos coger el resultado entero e imprimrlo por pantalla como entero.

2.b)Los resultados se imprimen correctamente,si el bit de la izquierda del todo es uno,imprime un número en negativo,si es pequeño(0x000000A0)sale 16,y si todos son F sale -1,por lo que imprime con binarios en complemento a 2

5/10/2016

2.c)La principal dificultad es que al pedir un número por teclado dicho número se almacena en $v0,y guardamos dicho número en un espacio reservado para poder cargar su dirección en $a0 (hay que poner $v0 a cero antes de operar,ya que es donde se realiza la transformación y el 8 de antes la altera).

2.d)El procedimiento es el mismo,pero al final del programa metemos una multiplicación por 4

3)Para la detección de caracteres incorrectos comprobamos el valor ASCII del carácter,si es distinto(según que valor tenga)de lo que puede escribirse se salta a ErrorCaracter,donde se muestra el error 1,si no se imprime el número y el valor 0(todo correcto).

(No está implementado la detección de longitud incorrecta)

4)Para esto solo hay que aplicar las sustituciones realizadas a lo largo de la práctica a los otros ejercicios de la práctica anteriores

5)Para esta última simplemente podemos multiplicar el número en binario por -1 y decirle al sistema con la orden li $v0 34 que imprima dicho número hexadecimal. Otra solución pasaría por utilizar la instrucción XORI con la máscara 0xFFFFFFFF para obtener el inverso

***PRÁCTICA 5***

11/10/2016

1)Para realizar la transformación de decimal a binario solo tenemos que cargar la cadena byte a byte,sumarla a un resultado y multiplicar por 10 por cada byte,así tenemos el número que la máquina interpreta como binarios

2.a)El procedimiento es mas largo,pues hay que variar el valor de $v0 constantemente,y luego hay que tener en cuenta el signo negativo del valor que vayamos a meter por teclado. En el caso de números negativos detectamos el valor ascii del signo(45) y metemos un -1 en un registro para que se multiplique al finalizar la conversión,ignorando la primera conversión para evitar errores(el valor del registro es 1 por defecto y solo cambia a -1 en ese caso).

2.b)Funciona para cualquier valor entero,ya sea positivo,negativo,una cifra,varias cifras,el cero…

2.c)Hay que hacer lo mismo,pero utilizamos funciones diferentes para cada número,asi como registros distintos para evitar mezclar registros y confundirnos,además aplicamos la suma justo antes de imprimir para tener los números correctos

3)En este caso introducimos los mensajes de error,imprimiendo el tipo de error que haya ocurrido

13/10/2016

4)Utilizamos el programa del apartado 2c,pero le añadimos un mecanismo que detecte los errores de carácter incorrecto o de valor excesivamente grande(Se reserva espacio de 32 bits para cada número)

Error:Cuando se hace el salto de linea,aparece un carácter extraño,desconozco el por que,pero no afecta a la funcionalidad del programa,solo es estético