Prácticas 4, 5 y 6 de *Estructura de los Computadores*

Eduardo Espuch

Universitat d'Alacant
Universidad de Alicante

Resumen

Documento que reúne los entregables de las prácticas 4, 5 y 6 de la asignatura *Estructura de los Computadores* ademas de los ejercicios opcionales de la practica 6. El documento tendrá adjunto cuatro carpetas, tres de las para los ejercicios entregables y una para los opcionales.

1. Ejercicios práctica 4

Partiendo de que tendremos que usar el algoritmo de Booth para realizar multiplicaciones, veamos paso por paso que hacer, considerando que usaremos un registro ($s0, por ejemplo) para almacenar el resultado final:

1. El primer paso sera sumar la constante 3, una instrucción de desplazamiento inmediato bastaría pero por ahorrar memoria usaremos un addi mas adelante para desplazar el resultado y ademas sumarle la constante.
2. Hacemos una multiplicación del entero por 2, en este caso, $2_{10} = 2^1$ con lo cual un desplazamiento de 1 bit a la izquierda bastara. Al salir pasaremos el valor al registro sumándole la constante 3 anteriormente dicha.
3. Queremos elevar al cuadrado el entero. Sabiendo que $a^2 = a \cdot a$, si usamos la instrucción mult y sabemos que el valor que nos interesa se guarda en [*hi,lo*], en concreto en *lo*, basta con desplazar este valor al parámetro de salida.
4. Teniendo el cuadrado, lo multiplicamos por 5_{10}=0000 0000 0000 0000 0000 0000 0000 0101_2 que, con el algoritmo de Booth, quedaría tal que $5_{10} = 0 \ldots 0(28bits) + 1_3 - 1_2 + 1_1 - 1_0 = 2^3 - 2^2 + 2^1 - 2^0 = 8 - 4 + 2 - 1$. Aunque en este caso en particular seria mucho más cómodo hacer $5 = 2^2 + 1$ y por lo tanto seria una instrucción de desplazamiento de 2 bits y una suma, se realizara por Booth, siendo entonces 3 desplazamientos, 2 restas y una suma.

Se pueden observar en ej1p4.asm en la carpeta ejobp4.

2. Ejercicios práctica 5

El primer ejercicio, teniendo el pseudocodigo al alcance, consiste en usar las correctas instrucciones o conjunto de instrucciones que realicen la acción correspondiente, usando un bucle do-while en el que si la suma de dos valores introducidos es distinta de 0, te vuelve a pedir dos valores y si es 0, entonces finaliza. Se pide el bucle do-while porque se quiere que al menos se ejecute el código una vez y luego compruebe si se cumple.

El segundo ejercicio te pide que, al introducir dos valores, escriba por pantalla los valores entre uno y otro. Para ello seguiremos un orden tal que:

1. Introducir dos valores, el primero debe de ser menor que el segundo.
2. Comprobar que el primero es menor o igual que el segundo, ya que los extremos del intervalo están incluidos, si no se cumple, debe de finalizarse. Para esto usamos una combinación del resultado de la instrucción sle junto con la instrucción bqez.
3. Imprimir el primer valor y a continuación sumarle 1.
4. Volver al paso 2.

Se pueden observar en ej1p5.asm y ej2p5.asm en la carpeta ejobp5.

3. Ejercicios práctica 6

Ejercicios opcionales

Apartado 4

1. Al ejecutar el codigo observamos que tanto el text segment como el data segment están siendo usados y ademas que, con las opciones de **Hexadecimal adresses** y **Hexadecimal values** se nos permite modificar la forma en la que los valores de interés se nos presentan.
2. Están ocupadas tres palabras, si sabemos que una palabra son 4 bytes (32 bits), podemos afirmar que 12 bytes están siendo ocupados.
3. Esta usando 3 instrucciones para acceder a las direcciones de memoria de 3 palabras almacenadas en ella, 2 instrucciones que usan las direcciones anteriores para acceder a la palabra guardada y 1 instrucción para sobrescribir una palabra de la memoria, 6 instrucciones de acceso a la memoria.
4. Tiene la dirección correspondiente a la etiqueta **B**, es decir, \$t1=0x10010004_{16}.
5. Se almacena en la dirección correspondiente a la etiqueta **C**, es decir, \$t1=0x10010008_{16}.
6. Se produce un error ya que el termino independiente de la instrucción es el desplazamiento de bytes desde la dirección dada. Como te pide trabajar con una palabra no puedes acceder a media palabra o a un byte en concreto de ella, debes de tener acceso a la palabra desde el inicio. Es fácil comprobarlo si en este campo ponemos múltiplos de 4 (4 bytes es una palabra en 32 bits) ya que desplaza la dirección lo justo para que sea también una palabra. (consideramos que la dirección corresponde a palabras tambien ya que es lo que se da en el ejemplo).
7. $0x8d310000_{16}$ ↦lw[f] \$s1[rd], 0[k](\$t1[rs]), f es la función, rd el registro destino, rs el registro fuente (contiene una dirección) y k es el desplazamiento en C_2 de bytes respecto a la dirección de rs.

   1000 11[f]01 001[rs]1 0001[rd] 0000 0000 0000 0000[k]$_2$

8. Con el segundo código que se no da, únicamente se modifica el segmento de datos, y observamos que accedemos a palabras enteras, medias palabras, bytes o agrupamos bytes en palabras. Tendremos ocupados los bytes desde 0x10010000 hasta 0x100103f.

9. Con **Hexadecimal values** y **ASCII** modificamos la visualización de los datos y vemos como se muestran los valores que hemos introducido, en particular la cadena de caracteres que se agrupan 4 caracteres en cada palabra y dentro de la palabra se lee de derecha a izquierda (debido a que se el primer carácter se guarda en el byte de menor peso).

10. El carácter 'l' esta en la palabra 0x10010028, ocupando el byte de menor peso, por lo tanto esta al inicio de la palabra y la dirección del byte corresponde con el de la palabra.

Apartado 5

1. Veamos que hace el código por orden:

   1 Almacena en memoria dos palabras, una contiene 6 y otra 8, y se etiquetan con A y B.

   2 Se reservan 4 bytes (una palabra) y se colocan 0, la dirección de esta palabra se etiqueta con C y va a continuación de B.

   3 Se guarda 1 en el byte a continuación de C y se etiqueta esta dirección con X.

   4 Se guardan en los bytes a continuación de la dirección X tres cadenas de caracteres acabadas en
   0. Los primeros 3 caracteres de la primera cadena y el byte de la dirección X comparten palabra, un dato importante a conocer.

   5 Las 12 primeras instrucciones que usan las funciones 4 y 11 del syscall se encargan de imprimir por pantalla las cadenas de caracteres almacenadas en memoria (la función 4 del syscall lee cadenas de caracteres hasta el
   0).

   6 Se guarda en$t0 la dirección de A y considerando que las direcciones de las palabras difiere en 4 bytes podemos acceder a la dirección B y C sin tener que guardar estas direcciones en algún registro.

   7 Se guardan en dos registros distintos el contenido de A y B y se suman en un tercero el cual se guardara en la dirección C.

   8 Se mueve el resultado al registro $a0 y se imprime por pantalla.

   9 Se finaliza el programa.

2. Como ya se ha comentado, la funcion 4 del syscall consiste en imprimir por pantalla una cadena de caracteres que se encuentra en un dirección dada por $a0 hasta encontrar un
   0.

3. La codificación maquina para syscall es $0x0000000c_{16}$=0000 0000...0000 1100_2.

4. El la dirección de A desplaza 8 bytes, es decir, la dirección de C.

Podemos observar los códigos usados en la carpeta ejopp6.

Ejercicios entregables

El ejercicio nos pide que introduzcamos dos valores cuando una frase nos lo indique y que estos valores se guarden en dos posiciones de memoria reservadas, las cuales con la función SWAP intercambiaran sus valores y que finalmente imprima el de menor valor y después el de mayor valor. Veamos que hacer para cada condición:

1. Funciones de nacionalización de la memoria:

   Usamos dos instrucciones .space 4 que reservan dos palabras de la memoria con todo 0 e indicamos dicha posición con dos etiquetas para reservar las posiciones de los valores que introduciremos. Usaremos también .asciiz para guardar las frases que nos interesen.

2. Funciones para almacenar los valores en memoria:

   Imprimimos con la función 4 del syscall las frases que hemos guardado para pedir el entero y guardamos en al menos un registro la dirección que habíamos reservado, en nuestro caso usamos un registro general $s3 ya que lo querremos usar mas adelante y que si usamos el desplazamiento de 4 bytes accedemos a la dirección siguiente.

   Cuando reciba en entero por consola, usar sw para almacenarlo en la dirección que le corresponde, A=$s3 y B=$s3+4.

3. Funciones para SWAP:

   Usando jal y jr podemos movernos a la dirección que hemos etiquetado como SWAP, donde se realizara el cambio de valores entre las posiciones de memoria A y B. Este intercambio es tan sencillo como leer su contenido y guardarlo en 2 registros distintos con lw y posteriormente guardar el registro correspondiente a A en la dirección B y el registro correspondiente a B en la dirección A con sw.

4. Imprimir de menor a mayor:

   Si volvemos a usar lw para guardar en dos registros los valores almacenados en las direcciones de A y A+4, y usamos sle junto con bqez para comprobar cual si A es menor que B (si son iguales no importara el orden) y bastara con hacer la impresión de los valores en el orden correcto según la respuesta dada por bqez.

El código con aclaraciones se podrá encontrar en la carpeta ejobp6.