Estructura de computadores

Doble Grado en Ingeniería Informática y Admón. de Empresas - 2º Curso

Práctica 1:
Introducción al lenguaje ensamblador

Gonzalo Vega Sanz NIU: 100386334
e-mail: 100386334@alumnos.uc3m.es

Ernesto Vieira Manzanera NIU: 100386252
e-mail: 100386252@alumnos.uc3m.es

Contenido

1. Primer ejercicio

A. Enunciado

"El objetivo de este ejercicio es desarrollar un programa en ensamblador que busque el número de ocurrencias de una palabra en una sopa de letras representada mediante una matriz cuadrada de caracteres. Para ello se diseñará e implementará el código de una función denominada *SearchWords* que acepta tres **argumentos**:

- Argumento 1: dirección de inicio de la matriz de caracteres, que se almacena por filas.
- Argumento 2: dimensión de la matriz (N). Se considera que la matriz es cuadrada.
- Argumento 3: dirección de la cadena de caracteres que se desea buscar.

La función **devolverá** un solo valor, el número de veces que aparece la cadena en la matriz. La búsqueda no tendrá en cuenta si son mayúsculas o minúsculas. El segmento de texto del programa a desarrollar debe incluir, la función main, la función *SearchWords* y todas aquellas funciones auxiliares que se consideren necesarias para el desarrollo del programa. La función main solo debe encargarse de llamar a la función *SearchWords* con los argumentos correspondientes (*WordSearch*, *Word* y *N* definidos en el segmento de datos) e imprimir exclusivamente el resultado que devuelve."

B. Pseudocódigo

```
1.  #Segmento .data
2.
3.  #Segmento .text
4.  main:
5.      #paso de parámetros
6.      #a0 = WordSearch (sopa de letras)
7.      #a1 = N (dimension de la matriz)
8.      #a2 = Word (palabra a buscar)
9.
10.     #Guardado de argumentos y puntero de pila actual ($ra)
11.
12.     jal SearchWord (Llamada a la función)
13.
14.     #Recuperamos valores de la pila
15.     #Imprimir resultado ($v0)
16.     jr $ra #finalizar programa
```

```
SearchWord:
    for i : 0 -> N
    #que mire la primera letra de Word y vaya comprobando con la sopa de letras.
    #Comprobamos mayusculas y minúsculas sumando o restando 32
    #if WordSearch(i) == Word(0) -> igual
    #else:
        #i += 1
        b for (inicio del bucle)
    igual:
        c = 0 #contador de caracteres para cada caso en que podamos tener la palabra
        Derecha:
            #c += 1
            #si N%(i+c) == 0 -> cambio de fila (no podemos analizar en esta direccion)
            #if Word(c) == caracterNulo: #Hemos encontrado la palabra
                #$v0 += 1
                b for
            #if WordSearch(i+c) == Word(c):
                b Derecha
            #else:
                #c = 0
                b Izquierda
        Izquierda:
            #c += 1
            #si N%((i+c)-1) == 0 -> cambio de fila (no podemos analizar en esta direccion)
            #if Word(c) == caracterNulo: #Hemos encontrado la palabra
                #$v0 += 1
                b for
            #if WordSearch(i+c) == Word(c):
                b Izquierda
            #else:
                #c = 0
                b Abajo
        Abajo:
            #c += N
            #si  i+c > N*N == 0 -> estamos fuera de la matriz (no podemos analizar en esta direccion)
            #if WordSearch(c/N) == caracterNulo: #Hemos encontrado la palabra
                #$v0 += 1
                b for
            #if WordSearch(i+c) == Word(c/N):
                b Abajo
            #else:
                #c = 0
                b Arriba
        Arriba:
            #c -= N
            #si  i+c < 0 -> estamos fuera de la matriz (no podemos analizar en esta direccion)
            #if Word(c/N) == caracterNulo: #Hemos encontrado la palabra
                #$v0 += 1
                b for
            #if WordSearch(i+c) == Word(c/N):
                b Arriba
            #else:
                #c = 0
                b for
    return:
        jr $ra
```

C. Funciones implementadas

La función pedida en este ejercicio es la función *SearchWord* que recibe como parámetros la dirección de inicio de la matriz que representa la sopa de letras, *WordSearch*; la dimensión de dicha matriz, *N*; y la dirección de inicio de la palabra que deberá buscar, *Word*.
La función irá letra por letra de la matriz; en el momento en que encuentra una letra igual a la primera letra de *Word* analizará la matriz en las 4 dimensiones para encontrar correspondencias con la palabra. La estructura básica para buscar en una dirección es:

```
dirección:
                    i = indice del primer elemento Word en la matriz
                    c = carácter de Word a analizar, con respecto a i
                    #si  (condición de pertenencia)->  (Si es falsa la condición, estamos intentando analizar
                    un elemento no válido, no podemos seguir analizando en esta dirección)
                    #Aumentamos el contador de carácter, el que correspondería al siguiente carácter de la
          palabra en la dirección elegida.
                    #if WordSearch(c+i) == caracterNulo: #Hemos encontrado la palabra
                              #$v0 += 1
                              #Seguimos con el for, siguiente elemento de la sopa
                    #if WordSearch(i+c) == Word(c): #el elemento es igual que su correspondiente en Word,
       seguimos analizando en esta dirección
                              b direccion
                    #else:
                              #c = 0
                              #Analizamos otra dirección
```

Para los casos en que encontramos el carácterNulo o cambiamos la dirección a analizar, hemos implementado las etiquetas siguiente, que realizan respectivas funcionalidades:

- *igualdef*: Cuando el registro que cargue el valor word sea igual a *$zero* irá a esta etiqueta mediante ***beqz $registro igualdef*** cuando se haya encontrado una palabra . En este caso sumaremos 1 en el resultado, y después iremos a la etiqueta *cambio*.
- *cambio*: Con esta etiqueta haremos el cambio de una a otra sección de la función mediante un switch. Así, si venimos de analizar la dirección derecha iremos a izquierda, de izquierda a abajo, y de abajo a arriba reseteando en cada caso el contador que sirve para recorrer Word.

D. Pruebas

	Datos a introducir		Descripción de la prueba	Resultado esperado	Resultado obtenido
1	Z z Z O o O C c C	Coz	Indiferencia entre mayúsculas y minúsculas.	3	3
2	R o M a O R o m M o R O a M O r	Roma	Una misma letra es el inicio de dos o más palabras buscadas.	4	4
3	L o l O o O L O l	lol	Palíndromos: la palabra se lee igual de izquierda a derecha que de derecha a izquierda. El resultado se duplica.	8	8
4	L o l O o O L O l	Cal	La palabra no se encuentra en la sopa de letras	0	0
5	R o m A t t P k w	Roma	La palabra se encuentra partida por una linea	0	0

2. Segundo ejercicio

A. Enunciado

"El objetivo de este ejercicio es desarrollar un programa en ensamblador que procese una matriz de números en coma flotante de simple precisión (float). Para ello se desarrollará una función denominada *ExtractExponents* que acepta los siguientes argumentos en el orden indicado:

- Argumento 1: Dirección de inicio de la matriz de números en coma flotante (A).
- Argumento 2: Número de filas de la matriz (N).
- Argumento 3: Número de columnas de la matriz (M).
- Argumento 4: Dirección de inicio de una matriz de números enteros (B) de dimensión NxM.
- Argumento 5: Número entero X.

La función se encargará de procesar todos los elementos de la matriz A y realizará la siguiente funcionalidad:

Si *exponente(A[i,j] < X)* ⇒ B[i,j] = exponente(A[i,j])

Si no, B[i,j] = 99999;

La función **devolverá** el número de exponentes que cumplen la condición de ser menores que X. La función ha de considerar el valor real del exponente almacenado en el número.

Tenga en cuenta que los exponentes se representan en el estándar IEEE 754 en exceso."

B. Pseudocódigo

```
1.  #Segmento .data
2.
3.  #Segmento .text
4.  main:
5.      #paso de parámetros
6.      #a0 = A
7.      #a1 = B
8.      #a2 = N
9.      #a3 = M
10.
11.     #Pasamos el quinto parámetro por la pila
12.     #Guardamos el quinto parametro en la pila y $ra (PUSH)
13.
14.     jal ExtractExponents (Llamada a la función)
15.
16.     #Recuperamos valores de la pila
17.     #Imprimimos el valor pedido (return de ExtractExponents)
18.         move $a0, $v0
19.         li $a0, 1
20.         syscall (imprimir Int)
21.     #Imprimimos la matriz B resultado
22.         bucle for n = 0 -> N*M
23.             #imprimimos el elemento i
24.             #Si fin de linea (contador de salto de linea = M) -> Imprimir salto de linea/ contador de
25.             # salto de línea = 0
26.             #Si no -> Imprimir espacio-coma (", "")
27.
28.         fin del for
```

```
29. ExtractExponents(Matriz, X):
30.     bucle for i = 0 -> N*M (Tantas iteraciones como elementos haya en la matriz)
31.         #Guardamos el numero en IEEE 754 en registro Reg (1 bit | 8 bits | 23 bits)
32.         #Caso especial para el 0
33.
34.         1. Eliminamos los bits de la mantisa
35.             srl $Reg, $Reg, 23 (1 bit | 8 bits)
36.         2. Nos quedamos con los 8 bits del exponente haciendo mascara con 0x0FF (000011111111)
37.             and $Reg, $Reg, 0x0FF
38.         3. Restamos el sesgo y obtenemos el valor real
39.             addi $Reg, $Reg, -127
40.
41.         #Analizamos el exponente:
42.         1. Caso Exponente < X
43.             #Exponente en la posición correspondiente de B
44.             #$v0 += 1
45.         2. Caso Exponente >= X
46.             #9999 en la posición correspondiente de B
47.         3. Caso Exponente no normalizado (viene de línea 23
48.             #-126 en la posición correspondiente de B
49.             #$v0 += 1
50.         #Actualizamos contador del bucle i++
51.         b for
52.     exit
```

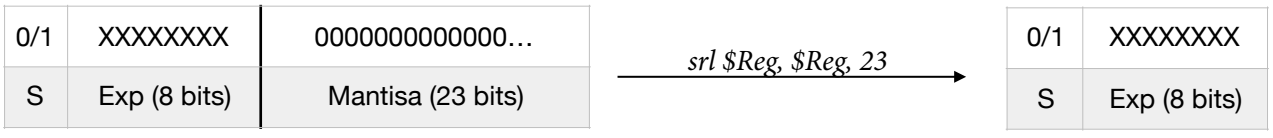
C. Funciones implementadas

La función pedida en este ejercicio es la función *extractExponents*, que recibe una matriz de N x M *floats* representados en IEEE 754 para extraer el exponente de cada uno y compararlo con un número dado X, realizando una de estas operaciones:

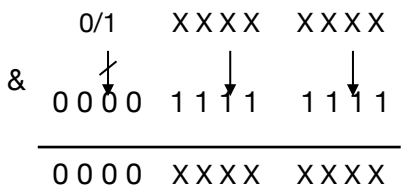
- si **Exponente < X**: copia el valor de Exponente a la misma posición del elemento pero en una nueva matriz B, y aumenta en 1 el contador de la variable de retorno.
- si **Exponente >= X**: copia el valor 9999 a la misma posición del elemento pero en una nueva matriz B
- si el **número analizado no está normalizad**o, pondremos el valor -126 en la posición correspondiente de la matriz B, y aumenta en 1 el contador de la variable de retorno.

La extracción del exponente se realiza de la siguiente forma:

1. Borrado de los bits de la mantisa



2. Obtención del exponente con una máscara 0x0FF




```
li $t7,0x0FF
and $Reg, $Reg, $t7
```

3. Resto del sesgo para obtener el valor real del exponente

$$\varepsilon = Exp + 127 \quad \Rightarrow \quad Exp = \varepsilon - 127 \qquad\qquad addi\ \$Reg, \$Reg, -127$$

Un caso especial que debemos contemplar es el *0.0*, que supone un número no normalizado en el sistema IEEE 754 y que por tanto requiere un tratamiento especial. Para eso añadimos la etiqueta *caso0*, que contiene el siguiente código:

```
caso0:
    add $t4, $a1, $t2        #Actualizamos la dirección de memoria
    li $t5, -126             #Cargamos en $t5 el inmediato -126
    sw $t5, ($t4)            #Guardamos el valor -126 en la posición de B
    addi $v1, $v1, 1         #Actualizamos el contador de ocurrencias (return)
    j reCount                #Continuamos el bucle
```

D. Pruebas

Las siguientes pruebas fueron realizadas para comprobar el correcto funcionamiento del código, intentando abarcar el mayor rango de posibilidades posible para encontrar errores en el funcionamiento

	Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado Obtenido
1	A = 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 N = 3 M = 5 X = 10	Análisis del caso 0.0. Matrices de dimensión diferente a la del ejemplo. Correcta impresión de la matriz.	5 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126	5 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126 -126
2	A = 0.0004 0.0002 0.1 0.000000006 0.00000007 0.000003 N = 3 M = 2 X = 10	Análisis de casos de números pequeños. Números de representación inexacta en IEEE 754.	6 -12, -13 -4, -28, -24, -19	6 -12, -13 -4, -28, -24, -19
3	A = 3784674627347823647.0 5967345983495830947.0 N = 2 M = 1 X = 0	Análisis de casos de números muy grandes.	0 99999 99999	0 99999 99999
4	A = 3784674627347823647.0 5967345983495830947.0 N = 2 M = 1 X = 127	Análisis de casos de números muy grandes. X muy grande también.	2 61 62	2 61 61

5	A = 37846746273.0, 5967345983.0 N = 2 M = 1 X = 127	Análisis de casos de números bastante grandes. X muy grande también.	0 35 32	0 35 32
6	A = 0.000000000000000007, 0.000000000000000008 N = 2 M = 1 X = -127	Análisis de casos de números muy pequeños. X muy pequeño también.	0 99999 99999	0 99999 99999
7	A = 0.000000000000000007, 0.000000000000000008 N = 2 M = 1 X = -56	Análisis de casos de números muy pequeños. X bastante pequeño también.	2 -57 -57	2 -57 -57

*Las pruebas con fondo verde se realizaron y obtuvieron el resultado esperado.
*Las pruebas con fondo rojo se realizaron y obtuvieron resultado diferente al esperado.

3. Conclusiones y problemas

Una vez completados los dos ejercicios y comprobar su correcto funcionamiento mediante las pruebas descritas anteriormente, hemos podido encontrar un par de circunstancias que pudieran ser causas de error en el código:

- Ejercicio 1: la existencia de palabras palíndromas supone el conteo por duplicado de cada caso. Una solución pudiera ser el análisis previo de la palabra a analizar, para qué, en el caso de ser un palíndromo, el resultado final se dividiese por 2.
- Ejercicio 2: en contadas situaciones, el introducir determinados numero en la matriz hemos obtenido resultados para el exponente diferentes de los que esperábamos. Esta incoherencia la podemos atribuir al error cometido al representar números en el sistema IEEE 754.

Tras realizar ambos ejercicios hemos sido conscientes de la diferencia entre la programación en alto y bajo nivel. Con la mayor complejidad que supone programar en lenguaje programador, nos hemos encontrados con numerosos problemas sobre en cuanto al tratamiento de los distintos tipos de datos y como se interpretan. No obstante, nuestro conocimiento acerca de este tipo de programación, el funcionamiento de las partes más internas del ordenador y cómo tratar la información de esta forma nos abre un gran numero de posibilidades.