Madrid, 8 de noviembre de 2020

Marco Antonio Merola Alibrandi NI: 100413665 correo: 100413665@alumnos.uc3m.es G81
Adrián Pérez Ruiz NI: 100429044 correo: 100429044@alumnos.uc3m.es G81

Práctica I
Introducción al lenguaje ensamblador

ÍNDICE

Ejercicio 1.1

Comportamiento: Inicializa todos los valores de la matriz A a cero. Y devuelve -1 si las filas o columnas son negativos o cero. En caso contrario ejecuta la función y devuelve 0.

Decisiones de diseño:
-Almacenar los parámetros en los registros a0,a1 y a2
-Obtener el número de elementos de la matriz multiplicando filas por columnas
-Crear un contador en un registro temporal inicializándolo a cero e incrementarlo de unidad en unidad en un bucle hasta que este sea igual o mayor al número de elementos de la matriz.
-El uso de pila no es necesario ya que la función solo recibe tres parámetros.

Pseudocódigo:

```
def Init(matriz,N,M):

if N<=0 or M<=0:
        return -1

for i in range(N):
        for j in range(M):
                matriz[i][j]=0
return 0
```

Batería de pruebas:

Datos a introducir	Descripción de la prueba	Resultados esperados	Resultado obtenido
M = 1 N = 1	Prueba con filas positivas, columnas positivas	Función devuelve: 0 Rellena matriz con 0s en el segmento de datos.	Resultado esperado
M = -1 N = 1	Prueba con filas negativas, columnas positivas	Función devuelve: -1	Resultado esperado
M = 1 N = -1	Prueba con filas positivas, columnas negativas	Función devuelve: -1	Resultado esperado
M = 0 N = 1	Prueba con filas igual a 0, columnas positivas	Función devuelve: -1	Resultado esperado
M = 1 N = 0	Prueba con filas positivas, columnas igual a 0	Función devuelve: -1	Resultado esperado

Ejercicio 1.2

Comportamiento: La función recibe la dirección de inicio de una matriz. Recibe una posición (i,j) de inicio de recorrido y recibe una posición (k,l) para finalizar el recorrido. La función debe sumar todos los elementos comprendidos entre dichas posiciones, devuelve un cero y el resultado del sumatorio. En caso de que la posición (k,l) sea anterior a la (i,j), la función devuelve -1. Si M o N son cero o números negativos la función devolverá -1.

Decisiones de diseño:
-Almacenar los parámetros de la dirección de inicio de la matriz, M,N,i en los registros por convenio $a0,$a1,$a2 y $a3
-Los restantes parámetros serán almacenados en pila
-Determinamos la posición de memoria (i,j) y (k,l) la guardamos en registro temporal $t0 y $t1.
-Iniciamos un bucle, donde se irán sumando valores de los elementos de la matriz, hasta que la posición (i,j) iguale a la (k,l) sumando de 4 en 4.

Pseudocódigo:

```
def Add (matriz, M, N,i, j, k, l):
        if M<=0 or N<=0 or i<0 or j<0 or k<0 or l<0:
                return -1
        if k<i:
                return -1
        if i==k and l<j:
                return -1
        suma=0
        for c in range(M):
                for d  in range(N):
                        if c==i and d==j:
                                suma+=matriz[c][d]
                        if c==k and d==l:
                                return 0,suma
```

Batería de pruebas:

-Nota: La comprobación de código M o N son negativos o cero se realizó en el ej1.1

Datos a introducir	Descripción de la prueba	Resultados esperados	Resultado obtenido
M = 3 N = 3, i=0,j=1,k=1,l=1 Matrix = 2 3 4 5 6 7	Prueba que i,j sean anteriores a k,l	Función devuelve: 0 y devuelve el sumatorio de los elementos situados entre la posición (i,j) y (k,l) de la matriz igual a 20	Resultado esperado
M = 3 N = 3, i=1,j=1,k=0,l=0	Prueba que i sea mayor que k, lo que significa que (k,l) es anterior a (i,j)	Función devuelve: -1	Resultado esperado
M = 3 N = 3 i=1,j=2,k=1,l=0	Prueba que i y k sean iguales, para verificar si (i,j) es anterior a (k,l) comprobando las columnas j y l	Función devuelve: -1	Resultado esperado
M = 3 N = 3 i=4,j=0,k=1, l=2	Prueba que i sea mayor que M	Función devuelve: -1	Resultado esperado

M = 3 N = 3 i=0,j=5,k=0,l=2	Prueba que j sea mayor que N	Función devuelve: -1	Resultado esperado
M = 3 N = 3 i=0,j=1,k=5,l=2	Prueba que k sea mayor que M	Función devuelve: -1	Resultado esperado
M=3 N=3, i=0,j=0,k=0,l=8	Prueba que l sea mayor que N	Función devuelve: -1	Resultado esperado
M=3,N=3, i=-1,j=0,k=1,l=2	Prueba que i sea negativo	Función devuelve: -1	Resultado esperado
M=3, N=3, i=0,j=-2,k=1,l=2	Prueba que j sea negativo	Función devuelve: -1	Resultado esperado
M=3,N=3, i=0,j=0,k=-1,l=2	Prueba que k sea negativo	Función devuelve: -1	Resultado esperado
M=3,N=3, i=0,j=0,k=1,l=-5	Prueba que l sea negativo	Función devuelve: -1	Resultado esperado

Ejercicio 1.3

Comportamiento: Comparar los elementos de la matriz A con los de la B situados entre (i,j) y (k,l). Y devuelve el número de elementos que coinciden en la misma posición. La función recibe 8 argumentos.

Decisiones de diseño:
-Almacenar los parámetros de la dirección de inicio de la matriz_A, matriz_B, M,N en los registros por convenio $a0,$a1,$a2 y $a3
-Los restantes parámetros serán almacenados en pila
-Determinamos la posición de memoria (i,j) de ambas matrices y (k,l) de la matriz A y la guardamos en registros temporales
-Apilamos los parámetros $aN anteriores para no perderlos al llamar a la subrutina cmp.
-Quitamos los dos parámetros innecesarios $a2 y $a3 para llamada a subrutina cmp
-Iniciamos un bucle, hasta que la posición (i,j) iguales a la (k,l) sumando de 4 en 4. En este bucle se harán las llamadas a la subrutina cmp, comparando los elementos en la posiciones i,j de ambas matrices y sumando a un contador las coincidencias.
-Se recupera y se desapila los parámetros $aN anteriores a la llamada a cmp.

Pseudocódigo:

```
from apoyo.o import cmp
def Compare (matriz_A, matriz_B , M, N,i, j, k, l):
        if M<=0 or N<=0 or i<0 or j<0 or k<0 or l<0:
                return -1
        if k<i:
                return -1
        if i==k and l<j:
                return -1
        coincidencias=0
        for c in range(M):
                for d  in range(N):
                        if c==i and d==j:
                                r = cmp (matriz_A[c][d],matriz_B[c][d])
                                if r==1:
                                        coincidencias += 1
                        if c==k and d==l:
```

return 0, coincidencias

Batería de pruebas:

-Nota:
La comprobación de código M o N son negativos o cero se realizó en el ej1.1
La comprobación de código de si los índices pasados como argumentos (i,j,k y l) se encuentran fuera de rango se realizó en el ej1.2
La comprobación de código de si el elemento (k,l) es anterior en la matriz al elemento (i,j) se realizó en el ej1.2

Datos a introducir	Descripción de la prueba	Resultados esperados	Resultado obtenido
matriz_A=1 (1x1) matriz_B=2 (1x1) i=0,j=0,k=0,l=0 M=1,N=1	Comprobar si cmp devuelve 0 si son distintos	Tras llamar a cmp, se almacene en v0 un 0	Resultado esperado
matriz_A=1 (1x1) matriz_B=1 (1x1) i=0,j=0,k=0,l=0 M=1,N=1	Comprobar si cmp devuelve 1 si son distintos	Tras llamar a cmp, se almacene en v0 un 1	Resultado esperado
matriz_A = 2 3 4 5 6 7 matriz_B = 2 3 4 8 8 8 i=0,j=0,k=1,l=2 M = 2 N = 3,	Detecta el número de elementos coincidentes entre la matriz A y B.	Función devuelve: 0 y devuelve el número de coincidencias de elementos, igual a 3 en este caso.	Resultado esperado

Ejercicio 1.4

Comportamiento: Esta función almacena en un vector todos los elementos de la matriz situados entre los índices (i,j) y (k,l)

Decisiones de diseño:
-Almacenar los parámetros de la dirección de inicio de la matriz_A, M,N, y el vector B en los registros por convenio $a0,$a1,$a2 y $a3
-Los restantes parámetros serán almacenados en pila
-Empleando la siguiente fórmula deducida por nosotros, comprobamos que la dimensión del vector coincida con el número de elementos que hay entre (i,j) y (k,l):
[(k-i+1) x N] - [(j+N-l-1)]
-Determinamos la posición de memoria (i,j) de la matriz A y la guardamos en un registro temporal.
-Inicializamos un contador a 0.
-Iniciamos un bucle, hasta que el contador alcance el valor igual al total de elementos entre las posiciones i,j y k,l. Iremos sumando de 4 en 4 y tomando como posición de inicio la posición de memoria (i,j) de la matriz A. Este bucle se encargará de almacenar cada elemento de la matriz A en un vector B.

Pseudocódigo:

```
def Extract (matriz_A, M, N, B, P, i, j, k, l):
        if M<=0 or N<=0 or P<=0 or i<0 or j<0 or k<0 or l<0:
                return -1
        if k<i:
                return -1
        if i==k and l<j:
                return -1
        elementos = [(k-i+1) x N] - [(j+N-l-1)]
        if P != elementos:
                return -1

        for c in range(M):
                for d  in range(N):
                        if c==i and d==j:
                                B.append(matriz_A[c][d])
                        if c==k and d==l:
                                return 0
```

Batería de pruebas:

-Nota:
La comprobación de código M o N son negativos o cero se realizó en el ej1.1
La comprobación de código de si los índices pasados como argumentos (i,j,k y l) se encuentran fuera de rango se realizó en el ej1.2
La comprobación de código de si el elemento (k,l) es anterior en la matriz al elemento (i,j) se realizó en el ej1.2

Datos a introducir	Descripción de la prueba	Resultados esperados	Resultado obtenido
P=-1 M=1 N=2 i=0 j=0 k=1 l=1	Comprobar si P es negativo	Devuelve -1	Resultado esperado
P=0 M=1 N=2 i=0 j=0 k=1 l=1	Comprobar si P es igual a cero	Devuelve -1	Resultado esperado
matriz_A = 2 3 4 5 6 7 1 2 3 vector =[0,0] P=2, M = 3 N = 3 i=0,j=0,k=2,l=2	Detecta que la dimensión del vector coincida con el número de elementos que hay entre (i,j) y (k,l)	Devuelve -1	Resultado esperado
matriz_A = 2 3 4 5 6 7 1 2 3 vector =[0,0,0,0,0,0,0,0,0] P=9, M = 3 N = 3 i=0,j=0,k=2,l=2	Detecta que la dimensión del vector coincida con el número de elementos que hay entre (i,j) y (k,l)	Función devuelve: 0 y almacena en memoria un vector con los elementos de la matriz que hay entre la posición (i,j) (k,l)	Resultado esperado

Ejercicio 2

Comportamiento: La función almacena la dirección de inicio de cuatro matrices de igual dimensión. Se deberá recorrer las matrices: La matriz A es de números en coma flotantes de simple precisión. En la Matriz S se deberá almacenar el bit de signo de cada elemento de la matriz A. En la Matriz E se almacenarán los exponentes de los elementos de la Matriz A, y finalmente en la Matriz Ma la mantisa.

Decisiones de diseño:
-Almacenar los parámetros de la dirección de inicio de la matriz A, S,E, y Ma en los registros por convenio $a0,$a1,$a2 y $a3.
-Los restantes parámetros serán almacenados en pila.
-Multiplicar MxN para crear un bucle de recorrido para las matrices
-Para quedarnos con el bit de signo, y el exponente, se aplicarán máscaras con el uso de “andi” y se realizarán desplazamientos lógicos a la derecha
-Para la mantisa, también se aplicará una máscara, pero tendremos en cuenta si el número está normalizado o no, si lo está se le añadirá un uno en el bit 23.

Pseudocódigo:

Nota: Hay partes de este pseudocódigo que no hemos implementado en python pues este, no sería de ayuda a la hora de desarrollar el código en mips (por ejemplo, la aplicación de máscaras, la cual en python no tendría el mismo sentido)

```
def extractValues (matriz_A, matriz_S,matriz_E,matriz_Ma,M,N):

        if M<=0 or N<=0:
                return -1

        for c in range(M):
                for d  in range(N):
                        #obtenemos bit de signo
                        matriz_S.append(bit_de_signo)
                        #obtenemos exponente
                        matriz_E.append(exponente)
                        #If normalizado:
                                #se añade un uno en el bit 23 de la mantisa
                        #Else:
                                #se añade un cero en el bit 23 de la mantisa
                        matriz_Ma.append(mantisa)
                        return 0
```

Batería de pruebas:

-Nota:
La comprobación de código M o N son negativos o cero se realizó en el ej1.1

Datos a introducir	Descripción de la prueba	Resultados esperados	Resultado obtenido
matriz_A = -1.1 M=N=1	Comprobar que el bit de signo es 1	Comprobar que en la memoria (matriz_S) se almacena 1. Y la función devuelve un 0	Resultado esperado
matriz_A = 2.2 M=N=1	Comprobar que el bit de signo es 0	Comprobar que en la memoria (matriz_S) se almacena 0. Y la función devuelve un 0	Resultado esperado
matriz_A = 0.1 M=N=1	Comprobar que se almacene el exponente correctamente para un número normalizado.	Comprobar que en la memoria (matriz_E) se almacena 0111 1011. Y la función devuelve un 0	Resultado esperado
matriz_A = 999999999999999999999999999999999999999.9 M=N=1	Comprobar que se almacene el exponente y la mantisa correctamente para un número no normalizado, infinito.	Comprobar que en la memoria (matriz_E) se almacena 1111 1111. Y que en la memoria (matriz_Ma) se almacena un 0. Y la función devuelve un 0	Resultado esperado
matriz_A = 0 M=N=1	Comprobar que se almacene el exponente y la mantisa correctamente para un número no normalizado, igual a 0.	Comprobar que en la memoria (matriz_E) se almacena 0000 0000. Y que en la memoria (matriz_Ma) se almacena un 0. Y la función devuelve un 0	Resultado esperado

Conclusiones y problemas:

Al realizar la práctica, tuvimos inconvenientes con el paso de parámetros en pila, ya que al inicio no teníamos muy claro el uso de la misma.

Además, en el problema de la función Extract, tuvimos dificultad para hallar una fórmula que determinase cuántos elementos hay entre dos posiciones (i,j) (k,l) de una matriz, así pues, tras investigar en internet y no encontrar mucha información al respecto, decidimos crear nuestra propia fórmula: [(k-i+1) x N] - [(j+N-l-1)].

Por último, tuvimos algunas dudas en el segundo ejercicio a la hora de tratar de extraer el bit de signo, exponente y mantisa. Pero en una clase magistral, tras una explicación sobre el uso de máscaras en MIPS, estas quedaron resueltas.

Como conclusión, nos hemos dado cuenta de la importancia que adquiere en este tipo de proyectos la estrategia de “divide y vencerás”. Pues desarrollar cada una de las funciones por separado y fusionarlas finalmente en un mismo documento, facilita la tarea.