PRÁCTICA 1

ESTRUCTURA DE COMPUTADORES

Enrique Benvenutto Navarro NIA: 100405806 Grupo: 80
100405806@alumnos.uc3m.es

Jorge Ríos Marfil NIA: 100405942 Grupo: 80
100405942@alumnos.uc3m.es

Índice:

Inicializar:

La función inicializar debe inicializar, es decir, poner a cero, todos los valores de una matriz A, que se pasa como parámetro. Otros parámetros que se pasan son M y N, siendo M el número de filas y N el de columnas. La función debe devolver -1 si tanto M como N son negativos o cero. En caso de que no lo sean, debe devolver 0.

A la hora de implementar esta función, nuestro equipo decidió, para preservar el valor de los registros *$a* por donde se pasan los parámetros, copiar todos los valores a registros temporales *$t* para después poder trabajar con ellos más cómodamente. Una vez movidos los valores, lo primero que hacemos es comprobar que m y n cumplan los requisitos que se nos piden (Es decir, no ser ni menores ni iguales a cero). Para ello usamos dos instrucciones blez, que saltan a la etiqueta finm1, si se cumple que el valor de alguno de los dos parámetros no es válido. Esta etiqueta lo que hace es pasar el parámetro a *$v0*, y restaurar la ejecución del método que llamó a la función inicializar.
Si se cumple que estos dos parámetros tienen valores aceptables, la función sigue con la ejecución de la siguiente instrucción. Las cuatro que siguen calculan un "contador". Hallan el tamaño de la matriz A multiplicando M por N, le restan 1 para poder ir de (MxN)-1 a 0 en vez de tener que ir de (MxN) a 1 (se verá la utilidad de hacer esto más adelante). Una vez calculado (MxN)-1, lo multiplicaremos por el número de direcciones de memoria que ocupa un registro (4). Esto facilitará recorrer la matriz en el bucle a continuación.
Tras hacer estos cálculos, entramos en el bucle que inicializará la matriz. Irá cargando el valor cero en cada uno de los registros que componen la matriz, e irá avanzando en ella restando 4 al "contador" que hemos calculado en los pasos descritos anteriormente. Una vez que dentro del bucle se cumpla que el contador y la dirección inicial de la matriz coinciden, se saltará a la etiqueta fin0, donde se pasará el valor 0 a *$v0*, y se restaurará la ejecución del método que llamó a la función inicializar.

Datos a introducir	**Descripción de la prueba**	**Resultado esperado**	**Resultado obtenido**
M <= 0 N > 0 Matriz no nula	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
M > 0 N <= 0 Matriz no nula	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
M <= 0 N <= 0 Matriz no nula	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
M > 0 N > 0 Matriz no nula	Probamos valores válidos	v_0=0 Se ejecuta todo el programa e inicializa la matriz	v_0=0 Se ejecuta todo el programa e inicializa la matriz

Sumar:

La función debe realizar la suma de dos matrices de enteros de la misma dimensión y dejarlos en una tercera matriz, por lo que hay que pasarle como argumentos las tres direcciones y las dimensiones de las matrices, en caso de que las dimensiones no sean correctas (menores o iguales que cero) la función nos devuelve -1, y si son correctas 0 y además guarda el resultado, si no, no ejecuta la suma.

En la decisión de diseño se ha decidido copiar los valores que se encuentran en los *$a* y en la primera posición de pila *$sp*, a registros *$t* para así poder trabajar sobre ellos, después comprobamos que los valores de *M* y *N* sean válidos (mayores que 0), si lo son, seguimos con la ejecución, si no, la finalizamos y devolvemos -1, después calculamos la dimensión de la matriz, multiplicando el número de filas por el de columnas, lo cual nos da un rango de (1 a n), para trabajar desde la posición de inicio le restamos uno al número de elementos (0 a n-1), después lo multiplicamos por 4 (tamaño de una palabra, 4 bytes) para ver el tamaño real de la matriz en memoria. Para recorrer la matriz hemos decidido recorrerla de fin a inicio, de esta forma nos ahorramos un registro ya que no tenemos que guardar la posición de fin de la matriz y la actual, sino que solo tenemos que comprobar que la posición por la que vamos de la matriz sea mayor o igual que 0. Calculamos las posiciones en las que vamos a trabajar de cada matriz, para ello sumamos a la posición inicial de cada matriz, la posición en la que queremos trabajar (que es nuestra posición del bucle), leemos los valores de la matriz B y C correspondiente y los guardamos en un registro temporal, los sumamos y lo guardamos en la posición correspondiente de A, después restamos 4 a la posición en la que nos encontremos y volvemos al inicio del bucle, así hasta que la posición que queramos sumar sea menor que 0, en ese caso devolvemos 0 y terminamos la ejecución.

Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado obtenido
M <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No realiza suma	v_0=-1 No realiza la suma
N <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No realiza suma	v_0=-1 No realiza la suma
Todos los valores válidos	Probamos que la función realice su función	v_0=0 Realiza la suma perfectamente	v_0=0 Realiza la suma perfectamente

ExtraerFila:

La función debe copiar la fila indicada de la matriz dada en el vector dado, para ello se pasa como argumento la dirección de un vector, la de una matriz, las dimensiones de la matriz y la fila a copiar. En caso de que M, N o J no sean válidos, la función devuelve -1 y no sigue con le ejecución, sino que devuelve 0 y copia los valores al vector. A la hora de plasmar la función en código ensamblador, hay que tener en cuenta varias cosas. Para empezar, al pasar los parámetros, no caben todos en registros $a, por lo que, siguiendo el convenio, asumimos que el resto, en nuestro caso el parámetro j, se pasa por pila. Lo primero que hacemos es pasar todos los argumentos a registros temporales $t para trabajar con ellos y poder modificar sus valores.

Una vez tenemos todo lo necesario, pasamos a comprobar que M y N no son ni negativos, ni cero, y que j es mayor que 0 y menor que M. Si lo fuese, el programa saltaría a la etiqueta finm1, devolviendo -1 y finalizando la ejecución de la función. Tras hacer las comprobaciones, tenemos que averiguar la dirección de la memoria donde empieza la fila J. Para ello hallamos la dirección de inicio de la fila multiplicando el tamaño de fila por J (la fila a copiar), lo multiplicamos por 4 (el tamaño de una palabra), y se lo sumamos a la dirección de inicio de la matriz.

Para recorrer la fila la vamos a recorrer de fin a inicio, para eso creamos un contador que va a tener el tamaño de una fila. Para ello usamos N, el tamaño de fila, que va de 1 a n y le restamos uno para que vaya desde 0. Lo siguiente es pasar a un bucle que nos recorrerá tanto la matriz como el vector, mientras que el contador sea mayor o igual a cero, y si no lo es salta a la etiqueta fin0, donde devolvemos 0 y terminamos la ejecución. Dentro del bucle se suma a la dirección de inicio de la fila y la del vector el contador. Almacenamos el valor de esa posición de la matriz en un registro temporal y lo guardamos en la posición correspondiente del vector. Tras esto, el contador se decrece en 4, y se vuelve al principio del bucle.

Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado obtenido
M <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
N <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
0 > j Resto de valores validos	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
j > N Resto de valores validos	Probamos valores no válidos	v_0=-1 No se ejecuta el resto del programa	v_0=-1 No se ejecuta el resto del programa
Todos los valores válidos	Probamos valores válidos	v_0=0 Se ejecuta todo el programa y extrae la fila	v_0=0 Se ejecuta todo el programa y extrae la fila

MasCeros:

La función nos pide, haciendo uso de la librería proporcionada, desarrollar una función que nos determine cuál de las dos matrices dadas tiene mayor número de ceros y, dependiendo de eso, nos devuelve 0 (en caso de que A tenga mas que B), 1 (en caso de que B tenga mas que A) o 2 (En caso de empate), y en caso de que los datos introducidos no sean válidos, -1.

Lo primero que hacemos es guardar los *$s* en pila ya que estamos obligados a mantener su valor original, después movemos los valores de los *$a* y *$ra* a los *$s* para no perder los valores al llamar a las subrutinas y comprobamos que las dimensiones de las matrices sean válidas (M y N > 0). En caso de que no lo sean terminamos la ejecución y devolvemos -1. Si son válidas, guardamos en *$a* los argumentos necesarios para llamar a la función calcular, la llamamos y movemos el resultado a un *$s* para no perderlo al volver a llamar a la función calcular, actualizamos los argumentos y la volvemos a llamar para obtener el número de ceros de la segunda matriz. Restauramos los valores de los $a que teníamos almacenados en los *$s*, después los valores de los *$s*, y por último comparamos los resultados y dependiendo de la comparativa devolvemos 0, 1, 2. Hemos decidido usar los *$s* (los cuales está obligada la función a conservar su valores) en vez de guardar y restaurar los valores de los *$t* antes de llamar a la función para así solo necesitar acceder una vez a pila en vez de dos una por cada llamada a la función calcular.

Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado obtenido
M <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No compara	v_0=-1 No compara
N <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No compara	v_0=-1 No compara
A y B sin ningún 0	Comprobamos que en caso de empate sin ceros se ejecute bien	v_0=2 Ejecuta sin fallos	v_0=2 Ejecuta sin fallos
A y B validos (los tres casos: empate, gana A o B)	Comprobamos el correcto funcionamiento de la función	v_0=0,1,2 Ejecuta sin fallos	v_0=0,1,2 Ejecuta sin fallos

ExtraerValores:

En este ejercicio se nos pide desarrollar una función que pasándole como parámetros: dirección de una matriz de floats; número de filas, M; número de columnas, N; y la dirección de inicio de un vector V de 6 posiciones. Nos devuelva en las posiciones del vector el número de elementos de cada tipo que tiene la matriz.
En nuestro caso, lo primero que hacemos es comprobar que M y N sean válidos si no, devuelve -1 y termina la ejecución, y si lo son, creamos las máscaras en registros temporales. Éstas máscaras son cadenas de 1s y 0s específicamente pensados para aislar los valores que necesitamos (en nuestro caso, el signo, el exponente y la mantisa) del resto del número, mediante la operación lógica AND, y así poder trabajar con sus componentes individualmente.
Recorremos la matriz de inicio a fin, para ello necesitamos el tamaño de la matriz, que obtenemos multiplicando el número de filas por el de columnas (1 a n), después le restamos 1 (0 a n-1), lo multiplicamos por 4 (el número de direcciones de memoria que ocupa un registro), y después se lo sumamos a la dirección de inicio de la matriz.
Mientras que nuestra posición sea mayor o igual que la inicial de A, se ejecuta el bucle. En él, leemos el valor de la posición que corresponda de la matriz, y lo almacenamos en un registro de coma flotante *$f*, para luego pasarlo a un registro temporal *$t*. Esto lo hacemos para poder aplicar la máscara. Una vez tenemos el valor aplicamos las tres máscaras (aplicar la operación lógica AND y rotar si es necesario hasta dejar a la derecha el valor) y guardamos cada valor obtenido en un registro $t para después comparar el signo, exponente y mantisa.
Después en los condicionales, se compara lo dicho anteriormente, para clasificarlo en los 5 tipos de valores que puede almacenar un valor en IEEE754, dependiendo del tipo de valor leído de A. En cada uno, se lee el valor de la posición del vector V que corresponda al tipo de valor en IEEE, se aumentará en 1, y se volverá a almacenar en la misma posición de V.
Hemos decidido crear una etiqueta que será llamada en todos los condicionales, para avanzar en la matriz, y volver a correr el bucle. Además tenemos las etiquetas fin0 y finm1, que devuelven 0 o -1, de acuerdo a lo solicitado en el enunciado y terminan la ejecución.

Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado obtenido
M <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No almacena nada en V, no se ejecuta el resto del programa.	v_0=-1 No almacena nada en V, no se ejecuta el resto del programa.
N <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No almacena nada en V, no se ejecuta el resto del programa.	v_0=-1 No almacena nada en V, no se ejecuta el resto del programa.
Todos los valores válidos	Probamos que la función realice su función	v_0=0 Deja en V los valores correspondientes indicados	v_0=0 Deja en V los valores correspondientes indicados

Sumar:

La función debe de realizar la suma de dos matrices de floats de la misma dimensión y dejarlos en una tercera matriz, por lo que hay que pasarle como argumentos las tres direcciones y las dimensiones de las matrices, en caso de que las dimensiones no sean correctas (menores o iguales que cero) la función nos devuelve -1, si son correctas 0 y además guarda el resultado, si no, no ejecuta la suma.

La estrategia seguida es la misma que el apartado sumar del primer ejercicio con la diferencia de que aquí tenemos que usar las instrucciones y los registros correspondientes para trabajar con el procesador de coma flotante, por lo que cambian el tipo de instrucciones que debemos de utilizar a la hora de realizar la suma de floats, las instrucciones que tenemos que usar para leer los valores que hay almacenados en memoria en coma flotante y los registros que se deben de utilizar para almacenar floats, ya que no son los mismo que usamos para almacenar enteros o direcciones de memoria, sino que son los $f los cuales además siguen una estructura propia de, cuales se deben usar para almacenar argumentos, temporales, temporales a salvar, etc. Los cambios realizados a la hora de realizar el ejercicio son los siguientes:

- *lw ---> ls*
- *add ---> add.s*
- *sw ---> s.s*
- *$t --->$f*

Datos a introducir	Descripción de la prueba	Resultado esperado	Resultado obtenido
M <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No realize suma	v_0=-1 No realiza la suma
N <= 0 Resto de valores validos	Probamos valores no válidos	v_0=-1 No realize suma	v_0=-1 No realiza la suma
Todos los valores válidos(también con infinitos, números no normalizados, y NaN)	Probamos que la función realice su función	v_0=0 Realize la suma perfectamente	v_0=0 Realiza la suma perfectamente**

***Al realizar las suma de algo con infinito devuelve infinito, al realizar la suma de un número normalizado con uno que no lo suma y lo deja como un número normalizado por lo que los decimales pequeños son truncados, y al sumar un NaN con un número devuelve NaN*

Pseudocódigos:

1.1 Inicializar:

```
mueve valores a registros temporales
si M || N <= 0  devuelve -1, fin
si no: continúa
calcula tamaño matriz
resta 1 tamaño matriz
multiplica por 4 tamaño matriz
suma tamaño matriz y dirección inicial matriz, almacena en tamaño matriz
mientras (tamaño > dirección inicial matriz) hacer:
        carga cero en posición de memoria que referencia tamaño matriz
        resta 4 a tamaño matriz
        salta inicio bucle
devuelve 0
fin
```

1.2 Sumar:

```
guarda valores
lee pila
guarda valor pila
si m y n > 0 continua si no, devuelve -1 y fin
calcula tamaño matriz
mientras tamaño > 0 (inicio bucle)
        lee valores matriz
        suma valores
        guarda valores
        resta 4 tamaño
        salta inicio bucle
devuelve 0
fin
```

1.3 ExtraerFila:

```
mueve valores a registros temporales
carga en registro temporal parámetro pasado por pila
Si M || N <= 0  devuelve -1, fin
Si no: continúa
sii 0 < j < M  continúa
Si no, devuelve -1, fin
calcula N*j almacena resultado en dirección j
multiplica dirección j por 4
suma dirección j y dirección inicial matriz, almacena en contador
resta 1 a N y almacena en contador
multiplica contador por 4
mientras (contador >= 0) hacer:
        suma contador y dirección j almacena en posición A
        suma contador y dirección A almacena en posición B
        almacena valor de posición A en posición B
        resta 4 a contador
        salta inicio bucle
devuelve 0
fin
```

1.4 MasCeros:

guarda valores de $s en pila y $ra después $a en $s
si m y n > 0 continua si no, devuelve -1 y fin
guarda argumentos para calcular
llama calcular
guarda resultado en $s
guarda argumentos para calcular
llama calcular
restaura $a, $ra despues $s y pila
si a > b → devuelve 0
si a < b -> devuelve 1
Si no devuelve 2 y fin

2.1 ExtraerValores:

```
M y N <= 0 ? si: devuelve -1, fin; no: continúa
carga máscaras en registros temporales
((calcula tamaño matriz) resta 1) multiplica por 4
suma tamaño matriz y dirección inicial matriz, almacena en tamaño matriz
mientras (tamaño matriz >= dirección inicial matriz) hacer:
        cargar IEEE en registro temporal
        hacer AND con registro temporal y máscara signo, almacenar en signo
        rotar signo, 1 posición izquierda
        hacer AND con registro temporal y máscara exponente, almacenar en exponente
        rotar exponente, 9 posiciones izquierda
        hacer AND con registro temporal y máscara mantisa, almacenar en mantisa
        si (exponente = 255)
                si (mantisa = 0)
                        si (signo = 0) suma 1 a la posición 1 de V
                        si no, suma 1 a la posición 2 de V
                si no, suma 1 a la posición 3 de V
        si (exponente = 0)
                si (mantisa = 0) suma 1 a la posición 0 de V
                si no, suma 1 a la posición 4 de V
        si no, suma 1 a la posición 5 de V
        resta 4 a tamaño matriz
        salta inicio bucle
devuelve 0 y fin
```

2.2 Sumar: El pseudocódigo es igual que 1.2 Sumar

Conclusiones y problemas encontrados:

La práctica nos ha servido para entender cómo funciona la programación en bajo nivel, la cual resulta interesante aprender ya que de esta forma entendemos cómo funciona el procesador, y como solucionar de forma más eficiente problemas en lenguajes de alto nivel.
A la hora de realizar la práctica no nos hemos encontrado casi dificultades ya que nuestra estrategia para resolver los ejercicios era: primero los hacíamos en pseudocódigo después los pasábamos a un lenguaje de alto nivel y después ya a ensamblador. Donde más dificultades hemos encontrado ha sido a la hora de realizar las máscaras y aislar las partes de coma flotante ya que hasta antes de la práctica no entendíamos cómo funcionaban.
A parte de todas las pruebas mostradas anteriormente también hemos probado que aunque partiendo de unos valores en *$t* diferentes de 0 nuestros métodos no se viesen afectados por ello.