Segunda Práctica en ensamblador, Arquitectura de computadores 2012 03

Por:Carlos Daniel Sánchez Ramírez
Santiago Pinzón Correa

En el trabajo, se debía de hacer un sistema que calculara el Seno Hiperbólico Inverso (`arcsinh(x)`) de un número entre 1 y -1, con una precisión indeterminada (es decir, que es el usuario quien decide cuan preciso debe ser.

En vez de usar en lenguaje C o C++, **preferimos usar D** por varias razones:

1. Es un lenguaje lo suficientemente maduro y usado como para tomarlo en serio.
2. Su sintaxis es casi idéntica a la de los dos lenguajes mencionados.
3. La sintaxis de ensamblador también es muy parecida a la que usa el compilador de VisualC++.
4. Es multiplataforma*, funcionando en Windows, Linux, MacOS, etc.
5. Permite separar código según la plataforma (así se puede hacer código especifico para Windows, y código especifico para Linux, o código especifico para 32bits y código especifico para 64bits, etc).

* Los compiladores basados en GCC(como MinGW) manejan una sintaxis de ensamblador muy diferente a la aprendida en el curso, por esta razón se descartó hacerlo en C.

En un principio se intentó hacer un sistema modular, creando las funciones *factorial* y *potencias* por separado. Sin embargo, dada la naturaleza mezclada de meter código ensamblador dentro del lenguaje D, el pasar parámetros fue una cosa muy enredada que finalmente no nos funcionó, por lo que al final, decidimos hacerlo todo en una función que devolviera un array de números *double* con la secuencia de la sumatoria.

Salvo el while de la sumatoria, todos los cálculos están en ensamblador. Eso es debido a que dentro del ensamblador embebido de D no es posible modificar Arrays como sí se puede en el VisualC++. Al parecer, los Arrays de D son bastante "inteligentes", por lo que son estructuras más complejas que un simple arreglo en la memoria. Por esta razón debía de estar fuera del código de ensamblador.

A la función se le pasa como parámetros el número x a calcular, y un N que es la precisión. Luego se pasa a inicializar las variables, en su mayoría de tipo *double*. Algunas de ellas serán solo auxiliares, y otras simplemente estarán allí de apoyo (como one y zero).

1. Se comienza el while (que va desde un `n=0` hasta `n=N`) "reseteando" las variables a cero, para que no vayan a interferir en los cálculos.
2. Se hace el primer pedazo, llamado `a1` que es la potencia de -1 a la n. Los cálculos de potencias

se hacen mirando primero si n es cero, en este caso, es 1; de lo contrario, comienza un ciclo n veces multiplicando por -1.

3. se sigue con `a2`, que es un `factorial( 2*n )`; primero se multiplica `2*n`, y luego se comienza el ciclo del factorial.
4. Seguimos con `b1`, que es una potencia de 4 a la n. El calculo es completamente igual al de la anterior potencia.
5. Luego hacemos `b2` con los cálculos anidados de `potencias(factorial(n), 2)`; se comienza con el factorial, tal cual como se hizo con el anterior factorial. Luego ese resultado se debe elevar a la 2, pero, como elevar a la dos es solo multiplicar la base por ella misma una vez, esta potencia se hace trivial.
6. Se pasa a calcular `r`. Por practicidad, se hizo el calculo sin la ayuda del coprocesador matemático, solo con operaciones de registros, ya que lo operación no es para nada compleja.
7. Finalmente se hace `c1`, que es el número x elevado a r. Dado que r nunca es cero, nos evitamos las comprobaciones y hacemos las multiplicaciones pertinentes de forma directa.
8. Para terminar se van colocando los valores de `a1, a2, b1, b2, r` y `c1` en el orden correcto para irlas operando dentro del stack del coprocesador matemático.
9. Se guarda el resultado en una variable auxiliar, la cual se suma al acumulado.
10. Fuera del bloque de ensamblador (pero aun dentro del while) se aumenta el tamaño del array dinámico de respuesta, y se coloca el valor (no la referencia) actual del acumulado en la posición `n` correspondiente.
11. Una vez el while finaliza sus ciclos, se retorna el resultado.

En el programa como tal, es decir, el `main()`, se tiene la declaración de algunas variables, y en particular, de algunos *Strings* dependiendo de la plataforma, como Windows no muestra tildes, hicimos una declaración para este sistema y otra para los demás (donde sí hay tildes).

Se piden los dos datos que serán pasados a la función `arcsinh`, que son el `x`, y el `N`. En caso de que alguno de estos dos valores no esté en los intervalos correctos, se piden nuevamente hasta que se den un `x` y un `N` correctos.

Luego se llama a la función, y finalmente se imprime el resultado recorriendo el array resultante en un *for*.