INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

FloatSort

Manual Técnico

Víctor R. Martínez P. **Fernando Aguilar R.**

01/06/2012

En este documento se describirán los principales métodos y algoritmos utilizados en el programa FloatSort. (Trabajo Final OPC 2012)

Tabla de contenido

Procedimientos:

1. MAIN

Método inicial. Realiza los siguientes pasos:

1. Inicializa el FPU
2. Lee los datos del archivo de entrada (LeeDatos)
3. Ordena el arreglo utilizando uno de los dos métodos programados (Sort)
4. Escribe el resultado en un archivo de salida (EscribeDatos)
5. Finaliza el programa

Al iniciar el FPU, el sistema establece la palabra de control en los valores iniciales. Asimismo, crea una pila vacía para guardar los flotantes.

2. KILL

Recibe: Nada
Devuelve: Nada
Utiliza:

- WriteString (Irvine16.inc)
- DumpRegs (Irvine16.inc)

Este método se llama en caso de que ocurra un error inesperado. Se imprimen los valores de los registros y un mensaje de error antes de terminar el programa.

3. LeeDatos

Recibe: Nada
Devuelve: Nada
Utiliza:

- AbreArchivoLect
- ReadFloat
- LeeByte
- CierraArchivo
- KILL
- WriteString (Irvine16.inc)
- WriteInt (Irvine16.inc)
- CrLf (Irvine16.inc)

Lee los datos del archivo de entrada (**float.in**) localizado en el mismo directorio del archivo ejecutable. Actualiza el arreglo **X** de tipo **REAL10** que se encuentra en memoria con los valores leídos.

En la primera línea del archivo se debe encontrar el número de elementos que se van a leer. A continuación se debe tener los elementos (uno por línea) escritos de forma numérica (números y punto decimal solamente).

En caso de error, imprime el mensaje y llama a KILL.

4. LeeInt

Recibe:

bx – Handle del archivo

Devuelve:

ax – Entero positivo leído de la línea del archivo

Utiliza:

- LeeByte

Lee una línea del archivo y guarda el resultado en un entero positivo.

5. LeeLinea

Recibe:

bx – Handle del archivo

ds:dx – Apuntador del buffer para guardar los datos

Devuelve:

Nada

Utiliza:

- LeeBytes
- KILL
- WriteString (Irvine16.inc)

Lee una línea de texto del archivo y va llenando la memoria a partir de dx con el resultado hasta encontrar el carácter CR. Recibe el archivo a leer en bx, y el inicio del espacio de memoria en dx. Devuelve el espacio actualizado a partir de dx.

En caso de que haya error en la lectura, imprime un mensaje de error y manda llamar a KILL.

6. LeeByte

Recibe:

bx – Handle del archivo

Devuelve:

al – Carácter leído

Utiliza:

Lee un byte del archivo y devuelve el resultado en al.

7. LeeBytes

Recibe:

bx – Handle del archivo

cx – Número de bytes a leer

ds:dx – Apuntador del búfer a llenar

Devuelve:

CF – 1 si hubo error, 0 en otro caso

ax – Si no hubo error, el número de bytes leídos. En caso de error, ax contiene el código del error ocurrido

Utiliza:

Lee un bloque de bytes del archivo y escribe lo leído en el espacio de memoria definido por ds:dx.

8. EscribeDatos

Recibe: Nada

Devuelve: Nada

Utiliza:

- CreaArchivoSalida
- PrintX
- CierraArchivo
- KILL
- WriteString (Irvine16.inc)

Escribe los datos del arreglo en memoria **X** de tipo **REAL10** en un archivo de salida (**float.out**).
Realiza los siguientes pasos:

1. Crea (trunca) y abre el archivo de salida (CreaArchivoSalida)
2. Imprime el arreglo (printX)
3. Cierra el archivo (cierraArchivo)

En caso de error, imprime un mensaje de error y manda llamar KILL.

9. EscribeByte

Recibe:

bx – Handle del archivo

al – Byte a escribir

Devuelve:

CF – 1 si hubo error, 0 en otro caso

Utiliza:

Escribe un byte en el archivo.

10. EscribeString

Recibe:

bx – Handle del archivo

ds:dx – Apuntador a la cadena a imprimir

Devuelve:

CF – 1 si hubo error, 0 en otro caso

Utiliza:

- Str_length (Irvine16.inc)

Escribe una cadena de caracteres al archivo.

11. EscribeChar

Recibe:

bx – Handle del archivo

al – Carácter a escribir

Devuelve:

CF – 1 si hubo error, 0 en otro caso

Utiliza:

Escribe un carácter al archivo.

12. EscribeInt

Recibe:

bx – Handle del archivo

dx – Entero a escribir

Devuelve:

CF – 1 si hubo error, 0 en otro caso

Utiliza:

Escribe un número entero al archivo.

13. EscribeCrLf

Recibe:

bx – Handle del archivo

Devuelve:

CF – 1 si hubo error, 0 en otro caso

Utiliza:

- EscribeString

Escribe los caracteres CR (10) y LF (13) al archivo.

14. CreaArchivoSalida

Recibe:

ds:dx – Cadena con el nombre del archivo terminada en nulo.

Devuelve:

CF – 1 si hubo error. 0 en otro caso.

ax – Si no hubo error, handle del archivo en modo escritura. En caso de error, el código indicando cual fue la causa.

Utiliza:

Crea el archivo de salida con el nombre dado por la cadena ds:dx. Regresa el CF con 1 si hubo error, y en ax el código de error. En caso de no haberlo, ax es el handle del archivo en modo escritura.

15. AbreArchivoEsc

Recibe:

ds:dx – Apuntador al nombre del archivo

Devuelve:

CF – 1 si hubo error, 0 en otro caso.

ax – Si hubo error, el código del error. El handle del archivo en otro caso.

Utiliza:

- AbreArchivo

Abre un archivo de nombre ds:dx en modo escritura.

16. AbreArchivoLect

Recibe:

ds:dx – Apuntador al nombre del archivo

Devuelve:

CF – 1 si hubo error. 0 en otro caso

ax – Si hubo error, el código del error. Si no, el handle del archivo en modo lectura.

Utiliza:

- AbreArchivo

Abre el archivo de nombre ds: dx en modo lectura.

17. AbreArchivo

Recibe:

ds:dx – Apuntador al nombre del archivo

Devuelve:

CF – 1 si hubo error. 0 en otro caso

ax – Si hubo error, el código del error. Si no, el handle del archivo en modo lectura.

Utiliza:

Procedimiento auxiliar para abrir un archivo haciendo uso de las interrupciones de DOS.

18. CierraArchivo

Recibe:

bx – Handle del archivo

Devuelve:

CF – 1 si hubo error. 0 en otro caso.

ax – Si hubo error, contiene 6, el único error posible (bx no es un *handle* válido).

Utiliza:

Cierra el archivo abierto por AbreArchivo. Recibe el manejador del archivo en bx, si es válido, lo cierra. En otro caso establece la bandera de **carry**.

19. SelectionSort

Recibe:

ds:si – Apuntador al arreglo **X**

cx – Número de elementos en **X**

Devuelve: Nada

Utiliza:

- Compara

Ordenamiento por selección. Ordena el arreglo de flotantes en memoria **X** haciendo uso del siguiente algoritmo [1]:

1. Encuentra el valor mínimo dentro del arreglo.
2. Intercambiarlo con la primera posición.
3. Repetir para el resto de la lista.

El resultado del ordenamiento queda en el mismo espacio de memoria ds:si

20. InsertionSort

Recibe: Nada

Devuelve: Nada

Utiliza:

- Compara

Ordenamiento por inserción. Ordena el arreglo de flotantes en memoria **X** haciendo uso del siguiente algoritmo [2]:

1. Supongamos que existe una función *Insertar* diseñada para insertar en una secuencia ordenada al principio del arreglo. Comenzando con el final del arreglo, recorre cada elemento una posición a la derecha hasta encontrar una posición adecuada para el insertar el nuevo elemento. Esta función tiene el efecto colateral de sobrescribir el valor guardado inmediatamente después de la posición insertada.
2. Se realiza un ordenamiento por inserción, empezando por el elemento del arreglo más a la izquierda e invocando *Insertar* para insertar cada elemento encontrado en su posición adecuada. La secuencia ordenada a la que se le inserta el elemento es guardada al principio del arreglo en un conjunto de índices ya examinados. Cada inserción sobrescribe un solo valor, el valor a insertar.

21. Compara

Recibe:

ST(0) y ST(1) – Dos números en la pila de números flotantes

Devuelve:

CF, SF, ZF – Banderas de acuerdo al resultado de la comparación

Utiliza:

Compara el elemento flotante en la primera posición (ST[0]) contra el siguiente elemento (ST[1]). Asimismo copia la palabra de control a las banderas del CPU para realizar movimientos condicionales de acuerdo al resultado.

Este módulo genera los siguientes resultados:

Condiciones	Movimiento Condicional
ST(0) >= ST(1)	**jbe**
ST(0) <= ST(1)	**Jae**

22. printX

Recibe: Nada
Devuelve: Nada
Utiliza:

- EscribeInt
- EscribeCrLf
- WriteFloat
- CrLf (Irvine16.inc)

Imprime los contenidos del arreglo en memoria **X** de tipo **REAL10**

23. DEBUG

Recibe: Nada
Devuelve: Nada
Utiliza:

- ShowFPUStack
- WriteString (Irvine16.inc)
- DumpRegs (Irvine16.inc)

Imprime un mensaje de debug junto con el estado de los registros del CPU y el estado de la pila de números flotantes.

Procedimientos de Irvine Kip (2010)

24. ReadFloat

Recibe: Nada

Devuelve: Nada

Utiliza:

- fpuSet
- fpuReset
- GetChar
- power10
- WriteString (Irvine16.inc)

Lee un número flotante desde el teclado (o desde un archivo) y lo traduce a un punto flotante binario. Se coloca el resultado en la pila de números flotantes, en específico, en ST(0).

25. WriteFloat

Recibe:

bx – Handle del archivo

Devuelve: Nada

Utiliza:

- fpuSet
- fpuReset
- fChkNaN
- fcompare
- splitup
- wrdigits
- EscribeString
- EscribeChar

Imprime el primer valor de la pila de números flotantes al archivo de salida.

26. ShowFPUStack

Recibe: Nada
Devuelve: Nada
Utiliza:

- WriteFloat
- CrLf (Irvine16.inc)
- WriteDec (Irvine16.inc)

Imprime la pila FPU en formato decimal con exponenciales. Éste módulo fue escrito por James Brink de la universidad Pacific Lutheran y es utilizado con permiso en la librería de Irvine Kip.

Utiliza la técnica de FINCSTP para mover el elemento de hasta arriba de la pila sin sacarlo de esta. El mismo procedimiento limpia la palabra de control del FPU antes de terminar.

27. fpuSet

Recibe: Nada
Devuelve: Nada
Utiliza:

Salva una copia de la palabra de control y establece los bits RC (10,11) y PC (8,9) de la palabra de control del FPU junto con los bits de excepción (0 a 5).

RC: xx00 0000 0000b

00 – redondear al par más cercano
01 – redondear hacia abajo (para -inf)
10 – redondear hacia arriba (para +inf)
11 – redondear hacia cero (truncar)

PC: xx 0000 0000b

00 – precisión sencilla (24 bits)
01 – reservado
10 – precisión doble (53 bits)
11 – doble precisión extendida (64 bits)

28. fpuReset

Recibe: Nada
Devuelve: Nada
Utiliza:

Restablece la palabra de control del FPU a partir del valor guardado en fpuSet.

29. fChkNaN

Recibe: Nada
Devuelve: Nada
Utiliza:

Compara el resultado de la última instrucción FTST para ver si la bandera Z está puesta para verificar si en realidad el número es un NaN.

30. fChkInfinity

Recibe: Nada
Devuelve: Nada
Utiliza:

Compara el resultado de la última instrucción FTST para ver si el resultado en realidad en un +/- INFTY.

31. fcompare

Recibe: Nada
Devuelve: Nada
Utiliza:

Compara dos valores punto flotante. Transfiere los registros ZF y SF de la palabra de control del FPU hacia el CPU para hacer movimientos condicionales.

32. normalize

Recibe: Nada
Devuelve: Nada
Utiliza:

- fcompare

Recorre ST(0) al rango definido por 10^8 y 10^9 ajustando el exponente en el proceso.

33. splitup

Recibe: Nada
Devuelve: Nada
Utiliza:

- fcompare
- normalize

Recibe un número no negativo en ST(0). Suponiendo que el valor del número es V, se busca encontrar un exponente E y una mantissa M tal que

$$V = (M * 10^{-8}) * 10^{E}$$

sujeto a

$$10^{8} \leq M \leq 10^{9}$$

Utilizando la tabla de potencias pwr10 se hace una búsqueda binaria y reducción para reducir las posibles soluciones. La mantissa resultante quedará en ST(0) y el exponente en la variable *exponent*.

34. wrdigits

Recibe:
ecx – Número de dígitos a escribir
eax – Dígitos a escribir
Devuelve: Nada
Utiliza:

- EscribeChar
-

Procedimiento auxiliar. Escribe *ecx* dígitos del registro *eax* como dígitos decimales con ceros al principio.

35. power10

Recibe:

eax – Exponente con signo

Devuelve: Nada

Utiliza:

- WriteString (Irvine16.inc)

Agrega 10.0^{s*eax} con s el signo a la pila de números flotantes FPU.

Bibliografía

[1] Wikipedia contributors, «Selection sort,» Wikipedia, The Free Encyclopedia., 1 Junio 2012. [En línea]. Available: http://en.wikipedia.org/w/index.php?title=Selection_sort&oldid=495441217. [Último acceso: 1 Junio 2012].

[2] Wikipedia contributors, «Insertion sort,» Wikipedia, The Free Encyclopedia., 2012 Mayo 31. [En línea]. Available: http://en.wikipedia.org/w/index.php?title=Insertion_sort&oldid=495315642. [Último acceso: 1 Junio 2012].