UNIVERSIDAD DE LA LAGUNA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Estructura de Computadores
Práctica de programación, curso 2010/11

Profesor: Juan Julian Merino Rubio

Ensamblador en Windows

En esta asignatura Estructura de Computadores del primer cuatrimestre nos iniciamos en la programación en ensamblador. El objetivo principal es aprender el lenguaje, y el entorno de la programación no debe distraernos de este objetivo por lo que vamos a tratar de mantenerlo lo más simple posible, aunque se trate del entorno Windows.

Vamos a aprender a manejarnos con programas de 32 bits, con el procesador en modo protegido, sin las limitaciones de los segmentos del modo MS-DOS. Vamos a comprobar que teniendo una cierta destreza con el ensamblador los programas en entorno Windows no se diferencian tanto de los de MS-DOS, al menos al nivel de dificultad que se va a exigir.

Naturalmente no se pretende desarrollar un enorme programa, complejo, con muchas ventanas, con muchas opciones de interacción con él, no. Para eso están los lenguajes de alto nivel. Nos vamos a limitar a desarrollar unos programas sencillos que van a permitirnos entender las dificultades y limitaciones que los lenguajes de alto nivel tienen "por detrás", bajo la superficie.

Las mejores referencias que he encontrado para programar en el entorno de Windows son los libros de Kauler [1] y de Petzold [2] de la bibliografía, además de la siguiente página: `http://en.wikibooks.org/wiki/Windows_Programming/`
y el sitio y foro del paquete MASM32 que vamos a emplear. Unas lecciones muy completas tiene el famoso curso de **Iczelion**, que junto a grandes cantidades de información lo tienen en

`http://win32assembly.online.fr/`

y lo mejor de todo es que ese curso está traducido al español en

`http://mipagina.cantv.net/numetorl869/tut_es.htm`

El paquete MASM32

En el ftp de la asignatura (`ftp://ftp.etsii.ull.es/asignas/EC/`) en la carpeta `MASM32/` he puesto el paquete comprimido `m32v10r.zip` que contiene, autoinstalable, la versión 10 —última por ahora— del paquete MASM32. También pueden bajárselo directamente de

`http://www.masm32.com/`,

que es recomendable visitar de todas maneras, sobre todo su foro:

`http://www.masm32.com/board/index.php`.

El MASM32 SDK (Software Development Kit), una vez instalado, contiene todo lo necesario para desarrollar los programas en ensamblador, incluyendo un editor, el **Quick Editor** (`qeditor.exe`) que tiene todo lo necesario para desarrollar los programas, aunque cada uno puede usar el que mejor conozca, que esto es cuestión de gustos. Hay bastantes ejemplos resueltos (carpeta `examples`) y algunas lecciones (carpeta `tutorial`).

Los primeros pasos

Básicamente hay dos tipos de programas de Windows, el de tipo consola y el estándar de interfaz gráfico.

Un programa de tipo consola es muy parecido a los programas en entorno MSDOS en pantalla de texto y nuestros primeros programas serán de este tipo, como veremos a continuación.

Para ensamblar el código fuente, por ejemplo el primero `EC10p1.asm`, escriben en la línea de comamdo:

```
\masm32\bin\ml /c /coff EC10p1.asm
```

donde

/c le dice al MASM que ensamble sólo, sin enlazar.

/coff le dice al MASM que cree el archivo *.obj* en formato COFF [*Common Object File Format*].

Una vez que ensamble con éxito obtendremos el archivo objeto, `EC10p1.obj`, que hay que enlazar escribiendo:

```
\masm32\bin\link /SUBSYSTEM:CONSOLE /OPT:NOREF EC10p1.obj
```

y cuando la aplicación no sea de tipo consola sino GUI estándar cambiaremos la opción a

`/SUBSYSTEM:WINDOWS`.

Si están usando el **Quick Editor** pueden obtener el ejecutable desde el menú desplegable **Project**, donde se distinguen un tipo u otro de aplicaciones con dos opciones distintas:

Build All y **Console Build All**.

El primer programa

Como primer ejemplo de programa vamos a escribir en la pantalla y a leer desde el teclado, es decir interaccionar con el sistema. El Windows proporciona funciones para manejar todos los dispositivos del sistema y estas se llaman pasándoles los parámetros por la pila. Cuando las funciones son relativas a algún dispositivo, ya sea físico o virtual, o en general manejen alguno de los muchos *objetos* que el Windows utiliza (en el sentido de la programación orientada a objetos, OOP, en la que se basa Windows), se impone la utilización de los "manipuladores" [**handles** en inglés]. Por ejemplo, para emplear a la consola como dispositivo de salida se emplea al manipulador `consoleOutputHandle`, y para emplearla como dispositivo de entrada el `consoleInputHandle`. Ambos manipuladores se le solicitan primero al Windows mediante la

función `GetStdHandle` y luego hay que pasárselos como parámetros a las funciones que emplean dichos dispositivos, como `WriteConsole`, por ejemplo.

Precisamente es `WriteConsole` la principal rutina para escribir en la consola, y tiene el siguiente prototipo:

```
WriteConsole  PROTO,                    ; escribe en la consola una cadena
    hConsoleOutput:HANDLE,              ; manipulador de la salida
    lpBuffer:PTR BYTE,                  ; puntero a la cadena
    nNumberOfCharsToWrite:DWORD,        ; tamaño de la cadena
    lpNumberOfCharsWritten:PTR DWORD,   ; puntero a nº de bytes escritos
    lpReserved:DWORD                    ; (no se usa)
```

y se usa de la siguiente forma: llamando primero a la función `GetStdHandle` para obtener un manipulador, y pasándole la cadena de caracteres a escribir y la longitud de la misma, además de un puntero a una variable donde se guardará el número efectivo de caracteres escritos, algo así

```
.data
msg1    BYTE    "Estructura de Computadores - Curso 2010/11",13,10
smsg1   DWORD   $ - msg1
consoleOutputHandle     HANDLE  0   ; handle de salida de la consola
bytesWritten    DWORD   ?           ; número de bytes escritos
.code
    ; Se obtiene primero el manipulador de salida por la consola:
    INVOKE  GetStdHandle, STD_OUTPUT_HANDLE
    mov     consoleOutputHandle,eax
    INVOKE  WriteConsole, consoleOutputHandle, ADDR msg1, smsg1,
        ADDR bytesWritten, 0
```

Los caracteres 13 y 10 con los que termina la cadena de caracteres `msg1` hacen el salto de línea en la consola y así el texto, si está formado por varias líneas, queda más legible. También podemos escribir una macro que haga el salto de línea:

```
.data
cr          EQU     0Dh     ; vuelta de carro
lf          EQU     0Ah     ; salto de línea
crlf        BYTE    cr,lf   ; carriage return/line feed

newline MACRO
    INVOKE WriteConsole,
        consoleOutputHandle, ADDR crlf, 2,
        ADDR bytesWritten, 0
ENDM
```

Más adelante veremos otras funciones que permiten darle formatos a las salidas por la consola.

Para leer el teclado, o lo que es lo mismo, para introducir datos al programa mediante el teclado, vamos a distinguir dos casos: la lectura de teclas aisladas y la introducción de cadenas de caracteres que se aceptan al pulsar `Enter`.

La lectura de teclas aisladas, útiles para que la interacción con el programa sea más amigable, parando el programa hasta que se pulse una tecla, por ejemplo, o respondiendo a menús sencillos con la tecla adecuada (`pulsar "S" para seguir, "ESC" para salir del programa`), no es sencillo de hacer en Windows, pero en el paquete MASM32 nos proporcionan la rutina `ret_key` que lo hace muy bien. La rutina está encerrada en la macro `getkey` y devuelve en EAX el código de la tecla.

Para introducir cadenas de caracteres desde el teclado tenemos a la rutina `ReadConsole`, con un prototipo parecido al de `WriteConsole`:

```
ReadConsole PROTO,
        hConsoleInput:HANDLE,           ; manipulador de entrada
        lpBuffer:PTR BYTE,              ; puntero a la cadena destino
        nNumberOfCharsToRead:DWORD,     ; nº máximo de caracteres a leer
        lpNumberOfCharsRead:PTR DWORD,  ; puntero al nº real de bytes leídos
        lpReserved:DWORD                ; (no se usa)
```

y se usa de la siguiente manera: igual que antes, hay que conseguir un manipulador, que lo proporciona el Sistema Operativo a través de la función `GetStdHandle` –estos manipuladores se pueden usar a lo largo de todo el programa, no hay que pedir uno cada vez–, luego hay que preveer un sitio (un *buffer* o cadena destino) donde se van a depositar los caracteres que se introduzcan y a `ReadConsole` se le pasa su dirección y su tamaño; por último también hay que pasarle la dirección de una variable donde quedará el número efectivo de caracteres introducido; en el *buffer* se debe dejar sitio para los dos caracteres que forman el "final-de-línea", y si se quiere que el *buffer* contenga una cadena terminada en cero se reemplaza el byte que contiene 13 (cr) con un byte nulo.

```
.data
anchobuffer     EQU     80    ; buffer para escribir cadenas
msg4            BYTE    "Escribe una frase: "
smsg4           DWORD   $ - msg4
msg5            BYTE    anchobuffer DUP(?)
bytesRead       DWORD   ?     ; número de bytes leídos
.code
    INVOKE  GetStdHandle, STD_OUTPUT_HANDLE
    mov     consoleOutputHandle,eax
    INVOKE WriteConsole,
        consoleOutputHandle, ADDR msg4, smsg4,
        ADDR bytesWritten, 0
    INVOKE  ReadConsole,
        consoleInputHandle, ADDR msg5, anchobuffer-2,
        ADDR bytesRead, 0
```

En la pantalla (ventana consola) aparece escrito:

```
Escribe una frase: _
```

y el cursor se queda esperando a continuación; podemos teclear hasta 78 caracteres; la longitud de la cadena introducida se devuelve en la variable `bytesRead`.

Las librerías de funciones

Las funciones `GetStdHandle`, `WriteConsole` y `ReadConsole` pertenecen a la librería de enlaces dinámicos **kernel32.dll** del Windows, y están a disposición de todas las aplicaciones. Para llegar hasta ellas disponemos de la librería estática **kernel32.lib** que contiene los enlaces adecuados. Al montar el programa le debemos decir al **linker** que use la librería **kernel32.lib** para resolver las referencias a las funciones. Además el ensamblador necesita los prototipos de las funciones por lo que el paquete MASM32 proporciona el archivo *include* **kernel32.inc**.

Windows proporciona un abanico de recursos para los programas **win32** (programas de 32 bits en modo protegido). En el centro de todos los recursos está la API de Windows (*Aplication Programming Interface*. La API de Windows es una colección enorme de funciones, muy útiles, que residen en el propio Windows, y que están listas para ser usadas por cualquier programa de Windows. Estas funciones se almacenan en varias librerías de enlaces dinámicos (DLLs) tales como la ya nombrada **kernel32.dll**, la **user32.dll** y la **gdi32.dll**. La librería **kernel32.dll** contiene a las funciones del API que tratan de la gestión de la memoria y de los procesos, es responsable de cosas como las funciones de acceso a los archivos, la temporización y sincronización de los procesos, los mensajes entre procesos, y todo eso. La librería **user32.dll** contiene a las funciones de control de los aspectos del interfaz con el usuario, como las ventanas de mensajes, botones, etc. Por último **gdi32.dll** contiene a las funciones responsables de las operaciones gráficas, permitiendo dibujar líneas, círculos, rectángulos, sobre la pantalla, así como mostrar y manipular bloques de mapas de bits (*bitsmaps*). Además de las "tres principales" hay otras DLLs que pueden usarse en los programas, supuesto que se tenga suficiente información sobre las funciones de la API que se deseen usar. Por ejemplo, como la mayor parte de Windows 44se ha desarrollado en C se han implementado las funciones de la librería estándar del C (**stdlib**), es decir las funciones definidas en los archivos de cabecera (.h) más corrientes, como `stdio.h`, `string.h`, `stdlib.h`, etc, y se han puesto en una librería de enlaces dinámicos, la **msvcrt.dll**. Gracias a esta última librería tenemos a nuestra disposición, entre otra muchas, a la rutina `ret_key` para leer un único teclazo.

El grupo que ha desarrollado el paquete MASM32 ha escrito su propia colección de funciones y macros, poniéndolas en `masm32.lib` y `macros.asm`, respectivamente. Es muy instructivo su estudio. Cada librería .DLL tiene asociada una librería .LIB con los enlaces, y a su vez cada .LIB tiene su propio .INC con los prototipos. Además se necesitan montones de constantes y de estructuras que se suelen agrupar en el archivo `window.inc`. Es por tanto normal el tener que incluir todos estos archivos en nuestro código fuente, resultando algo así:

```
include \masm32\include\windows.inc
include \masm32\include\masm32.inc
;
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\msvcrt.inc
include \masm32\macros\macros.asm
;
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\msvcrt.lib
```

Conversión de números

Bien, con lo expuesto hasta ahora podemos escribir en la pantalla e introducir datos desde el teclado. Vamos un paso más allá. Para escribir en la pantalla un número contenido en una variable primero hay que convertir el número de binario a cadena de caracteres. En el código fuente del primer ejemplo, `EC10p1.asm`, se incluyen las rutinas `Bin4Asc` y `SBin4Asc`, con los siguientes prototipos:

```
Bin4Asc proto near, NUM4:DWORD, CADENA:PTR BYTE
SBin4Asc proto near, SNUM4:SDWORD, CADENA:PTR BYTE
```

y las descripciones de las rutinas, tal como aparece en el código fuente son:

```
;======================================== Bin4Asc ================
; Convierte un número entero, sin signo, de 32 bits a ASCIIZ
; El resultado es desde '0',0 hasta '4294967296',0
;-------------------------------------------------------------------
; Entrada:  parámetros por la pila (NUM4,CADENA)
;           NUM4 = número de 32 bits sin signo a convertir
;           CADENA = puntero al destino de la cadena ASCIIZ
; Salida:   EAX = nº de cifras decimales (1..10)
;===================================================================

;========================================== SBin4Asc ==============
; Convierte un número entero con signo de 32 bits a ASCIIZ
; El resultado es desde '-2147483648',0 hasta '2147483647',0
; pasando por '0',0
;-------------------------------------------------------------------
; Entrada:  SNUM4 = número a convertir
;           CADENA = puntero al destino de la cadena ASCIIZ
; Salida:   EAX = longitud de la cadena, cifras más el signo, (1..11)
;===================================================================
```

La conversión de cadena a número se hace con la rutina `AscDectoBin`, también incluida, y con el prototipo

```
AscDectoBin proto near, CADENA:PTR BYTE
```

y su descripción:

```
;============================== AscDectoBin ========================
; Convierte una cadena ASCII decimal a un entero con signo (SDWORD)
;-------------------------------------------------------------------
; Entrada:      CADENA = dirección de la cadena, con el formato
;                        [espacios][signo][dígitos]
; Salida:       EAX = resultado
; Modifica:     nada
;===================================================================
```

Además de las anteriores les he incluido en el código fuente las rutinas de conversion entre binario y cadena ASCIIZ hexadecimal y viceversa.

Enunciado de la práctica

Se debe escribir un programa llamado `PUNTOS.ASM`, en lenguaje Ensamblador de la arquitectura IA32, en entorno Windows, que solicite del usuario un número natural, que deberá ser mayor que 100 pero menor de 10.000. Este será el número de puntos que se deberán generar, con coordenadas aleatorias, enteras, pero dentro del cuadrado del plano comprendido entre las ordenadas $X_{min} = -5.000$ y $X_{max} = +5.000$ y las abcisas $Y_{min} = -5.000$ e $Y_{max} = +5.000$.

Una vez generados los puntos se deberá contar el número de ellos en cada cuadrante e informar por pantalla del recuento.

A continuación se ordenarán todos los puntos según su distancia al centro y se formará una tabla de ocho entradas con la distribución de los puntos según la distancia, en tramos de un octavo de la máxima distancia, siempre redondeando a enteros.

Ejemplo

Supongamos que introducimos un número de 3600 puntos:

```
C:\MASM32\PRACTICA>puntos

Estructura de Computadores - Curso 2010/11

Práctica realizada por Alicia Fernández Rodríguez, alu8845

Introducir un número de puntos: 3600
-
```

A continuación el programa generará los 3.600 puntos y el recuento por cuadrantes se muestra por pantalla:

cuadrante	primero	segundo	tercero	cuarto	total
nº de puntos	878	1037	916	769	3600

Se ordenan los puntos y supongamos que el más alejado (el último de la lista), de coordenadas (-4172,+4930), está a una distancia de 6458. Dividiendo esta distancia en ocho tramos iguales darían: 0 a 807, de 808 a 1614, de 1615 a 2422, de 2423 a 3229, de 3230 a 4036, de 4037 a 4844, de 4845 a 5651 y de 5652 a 6458. Se cuentan los puntos que hay en cada uno de esos tramos de distancia y se muestra en forma de tabla:

tramos	nº de puntos
0 a 807	653
808 a 1614	455
1615 a 2422	88
2423 a 3229	782
3230 a 4036	886
4037 a 4844	103
4845 a 5651	470
5652 a 6458	163

y se termina el programa, normalmente dando la opción de repetir o salir.

El programa debe defenderse ante el profesor, funcionando. La calificación será de 5 puntos si no tiene ninguna ampliación. Si no funciona correctamente no se admite. Una vez admitido el profesor se queda una copia para posterior comprobación y contrastación para descartar copias; por eso la nota es siempre provisional hasta la contrastación. Se puede conseguir una calificación más alta si se hacen ampliaciones del programa. A continuación les propondré algunas posibles ampliaciones, pero cada alumno puede intentar mejorar el programa en otros sentidos. La calificación definitiva de la práctica, con un peso del 20 %, se suma a la nota del examen, que es el restante 80 %.

Ampliaciones opcionales

El programa se plantea, en principio, para realizarlo en modo CONSOLE, sin más complicaciones, por lo tanto una ampliación inmediata es realizar el programa en modo WINDOWS gráfico, en su propia ventana, con ventanas auxiliares para mostrar las tablas, etc.

Otra ampliación, hacer el recuento de puntos por octantes en lugar de por cuadrantes, y mostrar la lista de todos los puntos, con sus coordenadas y la distancia al centro. Como la lista no cabría entera en pantalla debe poderse hacer *scroll* arriba y abajo para poderla ver entera.

Otra ampliación más puede ser la representación gráfica de los puntos, usando las funciones del API para realizar gráficos, dibujando además las circunferencias que delimitan las distancias de la última parte del programa.

Aprender a manejar las funciones que permiten crear archivos, escribirlos y leerlos, es una buena ampliación, guardando en un archivo los puntos generados, y pudiéndolos recuperar para hacer sobre ellos el recuento.

Bibliografía

[1] *Barry Kauler*
Windows Assembly Languaje and Systems Programming,
R&D Books, 1997.

[2] *Charles Petzold*
Programación en Windows,
Anaya Multimedia, 1992.

[3] *Kip R. Irvine*
Lenguaje ensamblador para computadoras basadas en Intel,
Pearson Education, Inc., 2008. (Capítulo 11)