**FASM**

Consola

Windows

**Nombre:** Vargas Cruz Jose Manuel **Carrera:** Ingeniería Informática

**Introducción al ensamblador:**

La mejor forma de desarrollar programas complejos es mediante el uso de los llamados

"ensambladores"; estos son programas que traducen un texto (programa fuente) a un

programa ejecutable.

De esta manera, se pueden aprovechar las características de los editores de texto ASCII. Se denomina así al formato más simple donde no se introducen códigos propios de los editores que mejoran el aspecto de los textos como negrillas, subrayados y otros.

"Lenguaje de máquina": se denomina conjunto de las instrucciones de un procesador.

El programa ensamblador acepta un lenguaje que incluye todo el lenguaje de máquina

("instrucciones") con ligeras y necesarias variaciones y añade una serie de "directivas",sentencias que tienen sentido en el ámbito del ensamblador y que pierden su identidad en el programa ejecutable.

Al lenguaje total, compuesto de instrucciones y directivas, se llaman "lenguaje ensamblador"

("assembly language") o simplemente "ensamblador" ("assembler").

Usualmente el archivo que contiene un programa ensamblador tiene la extensión ASM; por

ejemplo PRUEBA.ASM. Las líneas fuente que contiene deben tener el siguiente formato:

nombre acción expresión ;comentario

El "nombre" es una definición del usuario que crea un símbolo que puede ser utilizado luego

en el programa, pueden representar código, datos y constantes; solo debe existir si la "acción"

lo requiere y bien puede no existir.

La "acción" es una directiva o una instrucción. Si no existe, la línea debe ser solo un

comentario.

La "expresión" son el o los "parámetros" de la acción; por ejemplo operandos de

instrucciones.

El "comentario" es opcional y debe ir precedido de punto y coma.

Ejemplos:

P0: MOV AX,BX

Crea el nombre P0; seguido de : es una etiqueta representa código en el punto

donde esta la instrucción MOV AX,BX. Antes o después de esta instrucción se

puede bifurcar a ella como por ejemplo con JMP P0.

A DB 100

Crea el nombre A; es una variable y representa un dato. Se reserva espacio de un

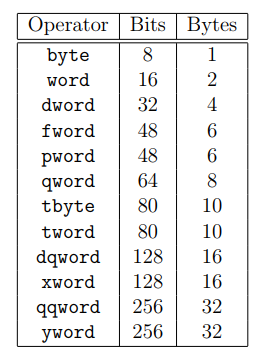
byte con la directiva DB (Define Byte), se le da el valor inicial 100 (decimal) y el

nombre A puede ser usado como operando memoria de instrucciones; por ejemplo se puede mover a AL con MOV AL,A o incrementar con INC A.

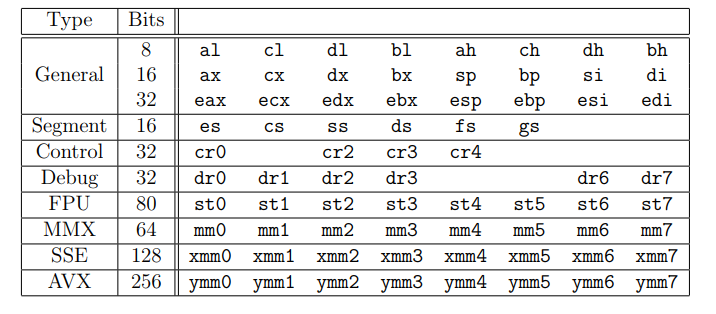
El proceso de creación de programas es el siguiente:



**Tamaño de Operadores:**

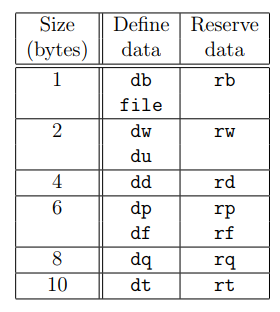
****

**Registros:**

****

**Definiciones de datos:**

Para definir datos o reservar un espacio para ellos, use una de las directivas enumeradas en la tabla:



La directiva de definición de datos debe ir seguida de una o más expresiones numéricas, separadas por comas. Estas expresiones definen los valores para las celdas de datos de tamaño según la directiva que se utilice. Por ejemplo, db 1,2,3 definirá los tres bytes

de los valores 1, 2 y 3 respectivamente.

La directiva de reserva de datos debe ir seguida de una sola expresión numérica,

y este valor define cuántas celdas del tamaño especificado deben reservarse. Todas las definiciones de directivas de datos también aceptan el valor ?, lo que significa que esta celda no debe ser inicializada a cualquier valor y el efecto es el mismo que al utilizar la reserva de datos directiva.

**Constantes y etiquetas**

En las expresiones numéricas también puede usar constantes o etiquetas en lugar de números.

Para definir la constante o etiqueta, debe usar las directivas específicas. Cada etiqueta puede ser definida sólo una vez y es accesible desde cualquier lugar de fuente (incluso antes de que fuera definido). La constante se puede redefinir muchas veces, pero en este caso sólo es accesible después de que se definió, y siempre es igual al valor de la última definición antes del lugar dónde se usa. Cuando una constante se define sólo una vez en la fuente, es accesible desde cualquier lugar.

La definición de constante consiste en el nombre de la constante seguida del carácter =

y expresión numérica, que después del cálculo se convertirá en el valor de constante.

Este valor siempre se calcula en el momento en que se define la constante. Por ejemplo tu

puedes definir la constante de recuento usando la directiva count = 17, y luego usarla en el

instrucciones de ensamblaje, como mov cx, count {que se convertirá en mov cx, 17 durante el proceso de compilación.

Hay diferentes formas de definir etiquetas. Lo más simple es seguir el nombre de la etiqueta. por los dos puntos, esta directiva puede incluso ser seguida por la otra instrucción en el mismo línea. Define la etiqueta cuyo valor es igual al desplazamiento del punto donde está definida. Este método se usa generalmente para etiquetar los lugares en código. La otra forma es seguir el nombre de la etiqueta (sin dos puntos) por alguna directiva de datos. Define la etiqueta con valor igual al desplazamiento del comienzo de los datos definidos, y se recuerda como una etiqueta para los datos con tamaño de celda como se especifica para esa directiva de datos en la tabla.

La etiqueta se puede tratar como una constante de valor igual al offset del código o datos etiquetados.

Por ejemplo, cuando define datos usando la directiva etiquetada **char db 224**, para poner el

desplazamiento de estos datos en el registro **bx**, debe usar la instrucción **mov bx,char** y poner el valor del byte direccionado por la etiqueta **char** al registro **dl**, debe usar **mov dl, [char]** (o mov dl, ptr char). Pero cuando intentas ensamblar **mov ax, [char],** causará

un error, porque el fasm compara los tamaños de los operandos, que deberían ser iguales. Tú puedes forzar el ensamblaje de esa instrucción usando la anulación de tamaño: **mov ax, word [char]**, pero recuerde que esta instrucción leerá los dos bytes comenzando en la dirección **char**, mientras se definió como un byte.

Un formato para un programa .EXE con las llamadas directivas puede ser:

format PE CONSOLE 4.0

entry start

include 'win32a.inc' ; Incluímos definiciones de estructuras y constantes

section '.text' code readable executable

start:

;líneas de código

section '.data' data readable writeable

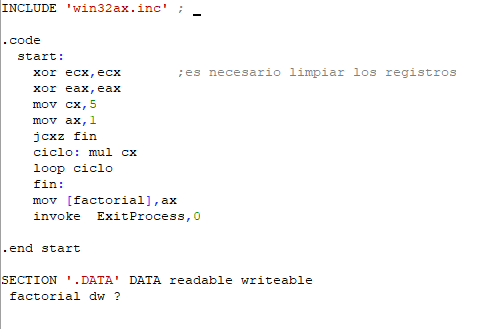
msg DB "Hola mundo", 0

section '.idata' import data readable writeable

library kernel32,'KERNEL32.DLL'

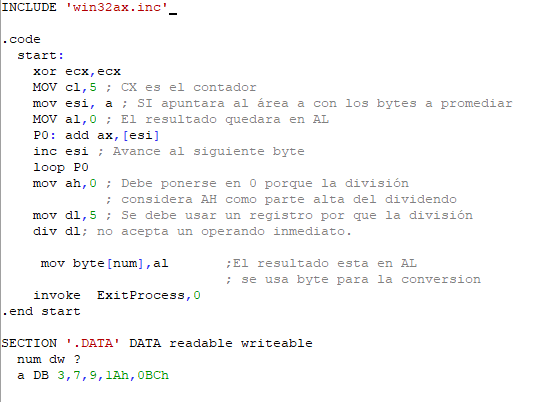
include 'api\kernel32.inc'

Por ejemplo, un cálculo del factorial de 5 en un archivo llamado FACTOR.ASM puede ser:

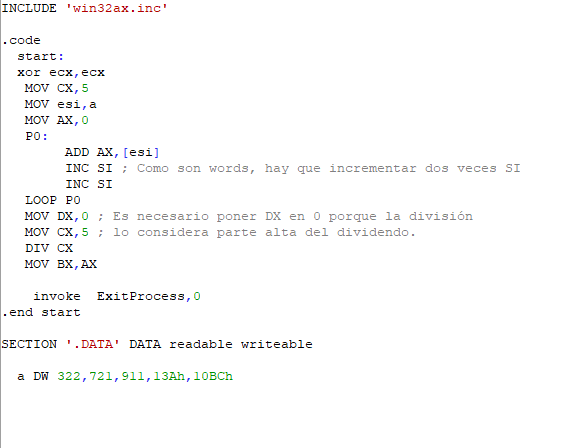


Ejercicios resueltos:

1.- Promediar 5 bytes. Dejar el resultado en AL

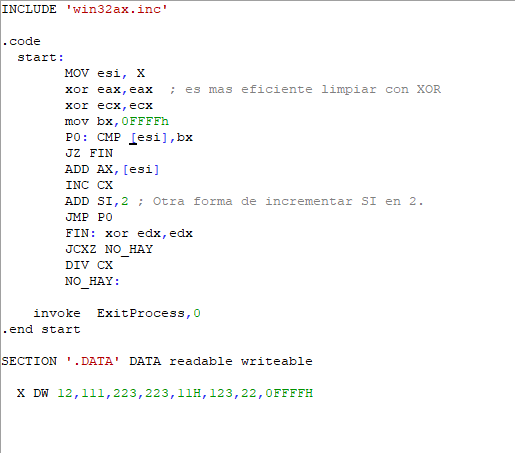


2.- Promediar 5 words. Dejar el resultado en BX



3.- En X se encuentra una lista de words. No se conoce la cantidad exacta de words pero se

sabe que el ultima es anterior a la marca FFFF. Promediarlas y dejar el resultado en DX.



**Instrucciones lógicas**

**'not'** invierte los bits en el operando especificado para formar un complemento a uno del operando.

No tiene ningún efecto sobre las banderas. Las reglas para el operando son las mismas que para la instrucción inc.

Las instrucciones **and**, **or** y **xor** realizan las operaciones lógicas estándar. Ellos actualizan

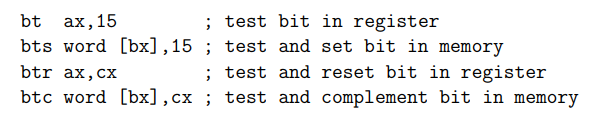
las banderas SF, ZF y PF. Las reglas para los operandos son las mismas que para la instrucción **add**.

Las instrucciones bt, bts, btr y btc operan en un solo bit que puede estar en la memoria o en un registro general.

La ubicación del bit se especifica como un desplazamiento del orden inferior final del operando. El valor del desplazamiento se toma del segundo operando, puede ser un byte inmediato o un registro general. Estas instrucciones primero asignan el valor del bit seleccionado a CF. La instrucción **bt** no hace nada más, **bts** establece el seleccionado

bit a 1, **btr** restablece el bit seleccionado a 0, **btc** cambia el bit a su complemento. los

El primer operando puede ser word o double word.

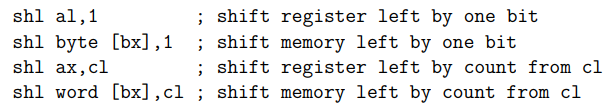


Las instrucciones **bsf** y **bsr** escanean una palabra o palabra doble para el primer bit establecido y almacenan el índice de este bit en el operando de destino, que debe ser un registro general. El bit cadena siendo escaneado está especificada por el operando de origen, puede ser un registro general o memoria. La bandera ZF se establece si toda la cadena es cero (no se encuentran bits establecidos); de lo contrario está despejado. Si no se encuentra ningún bit establecido, el valor del registro de destino no está definido.

**bsf** de orden inferior a orden superior (comenzando desde el índice de bits cero). **bsr** escanea desde lo alto orden a orden inferior (a partir del índice de bits 15 de una word o el índice 31 de una double word)

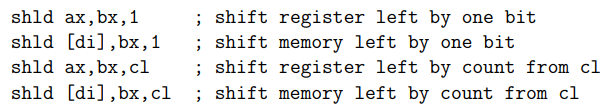


**shl** desplaza el operando de destino a la izquierda el número de bits especificado en el segundo operando. El operando de destino puede ser un registro general de byte, word o doble word o memoria. El segundo operando puede ser un valor inmediato o el registro cl. Los procesadores desplazan ceros desde el lado derecho (orden inferior) del operando a medida que los bits salen de el lado izquierdo. El último bit que salió se almacena en CF. **sal** es sinónimo de **shl.**

****

**shr** y **sar** desplazan el operando de destino a la derecha el número de bits especificado en

el segundo operando. Las reglas para los operandos son las mismas que para la instrucción **shl**. **Shr** desplaza ceros desde el lado izquierdo del operando a medida que los bits salen del lado derecho. El último bit que salió se almacena en **CF**. **sar** conserva el signo del operando cambiando a ceros en el lado izquierdo si el valor es positivo o cambiando a unos si el valor es negativo. **shld** desplaza los bits del operando de destino a la izquierda el número de bits especificado en el tercer operando, mientras se desplazan los bits de orden superior del operando de origen al operando de destino a la derecha. El operando fuente permanece sin modificar. El destino El operando puede ser un registro general o memoria de word o double word, el operando fuente debe ser un registro general, el tercer operando puede ser un valor inmediato o el registro cl.



**hrd** desplaza bits del operando de destino hacia la derecha, mientras desplaza bits de orden inferior desde el operando de origen al operando de destino a la izquierda. El operando fuente permanece sin modificar. Las reglas para los operandos son las mismas que para la instrucción **shld**. **rol** y **rcl** rotan el operando de destino de byte, word o double word que deja el número de bits especificados en el segundo operando. Para cada rotación especificada, el bit de orden alto que sale de la izquierda del operando vuelve a la derecha para convertirse en el nuevo poco de orden bajo. Además, **rcl** pone en CF cada bit de orden superior que sale por la izquierda lado del operando antes de que vuelva al operando como el bit de orden inferior en el siguiente ciclo de rotación. Las reglas para los operandos son las mismas que para la instrucción shl. ror y rcr rotan el operando de destino de byte, palabra o palabra doble a la derecha del número de bits especificados en el segundo operando. Para cada rotación especificada, el bit de orden bajo que sale por la derecha del operando regresa a la izquierda para convertirse en el nuevo bit de orden superior. rcr adicionalmente pone en CF cada bit de orden bajo que sale por la derecha lado del operando antes de que vuelva al operando como el bit de orden superior en el siguiente

ciclo de rotación. Las reglas para los operandos son las mismas que para la instrucción shl.

prueba realiza la misma acción que la instrucción y, pero no altera la operando de destino, solo actualiza las banderas. Las reglas para los operandos son las mismas que para la instrucción **and**.

**Funciones para Consola:**

* **GetStdHandle** devuelve un identificador para la entrada estándar, la salida estándar o el dispositivo de error estándar.

HANDLE GetStdHandle(

DWORD nStdHandle // input, output, or error device

);

Parametros

nStdHandle:

Especifica el dispositivo para el que devolver el identificador. Este parámetro puede tener uno de los siguientes valores:

Value Meaning

STD\_INPUT\_HANDLE Standard input handle

STD\_OUTPUT\_HANDLE Standard output handle

STD\_ERROR\_HANDLE Standard error handle

* **WriteFile** escribe datos en un archivo y está diseñado para operaciones sincrónicas y asincrónicas. La función comienza a escribir datos en el archivo en la posición indicada por el puntero del archivo. Una vez completada la operación de escritura, el puntero del archivo se ajusta por el número de bytes realmente escritos, excepto cuando el archivo se abre con FILE\_FLAG\_OVERLAPPED. Si el identificador de archivo se creó para entrada y salida (E / S) superpuestas, la aplicación debe ajustar la posición del puntero del archivo una vez finalizada la operación de escritura.

BOOL WriteFile(

HANDLE hFile, // handle to file to write to

LPCVOID lpBuffer, // pointer to data to write to file

DWORD nNumberOfBytesToWrite, // number of bytes to write

LPDWORD lpNumberOfBytesWritten, // pointer to number of bytes written

LPOVERLAPPED lpOverlapped // pointer to structure needed for overlapped I/O

);

* **ReadFile** lee datos de un archivo, comenzando en la posición indicada por el puntero del archivo. Una vez completada la operación de lectura, el puntero de archivo se ajusta por el número de bytes realmente leídos, a menos que el identificador de archivo se cree con el atributo superpuesto. Si el identificador de archivo se crea para entrada y salida (E / S) superpuestas, la aplicación debe ajustar la posición del puntero del archivo después de la operación de lectura.

BOOL ReadFile(

HANDLE hFile, // handle of file to read

LPVOID lpBuffer, // address of buffer that receives data

DWORD nNumberOfBytesToRead, // number of bytes to read

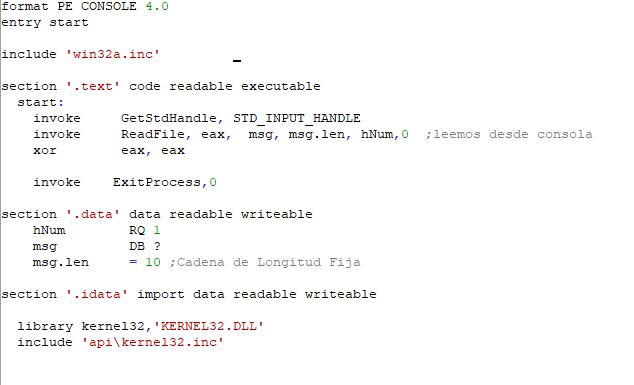
LPDWORD lpNumberOfBytesRead, // address of number of bytes read

LPOVERLAPPED lpOverlapped // address of structure for data

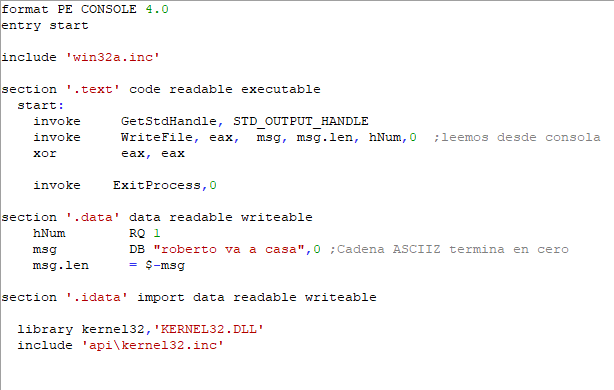
);

**Ejercicios resueltos**

1.- Leer una cadena de longitud fija

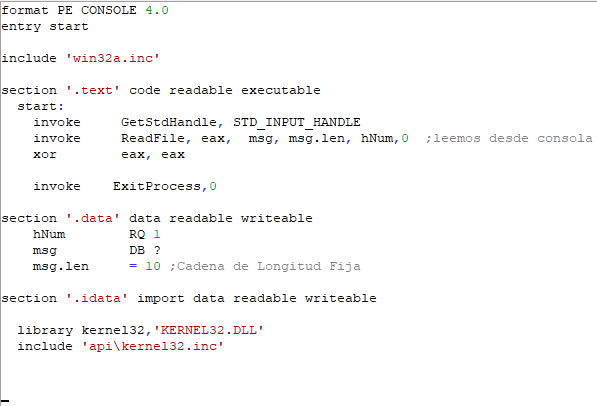


2.- Desplegar una cadena Asciiz



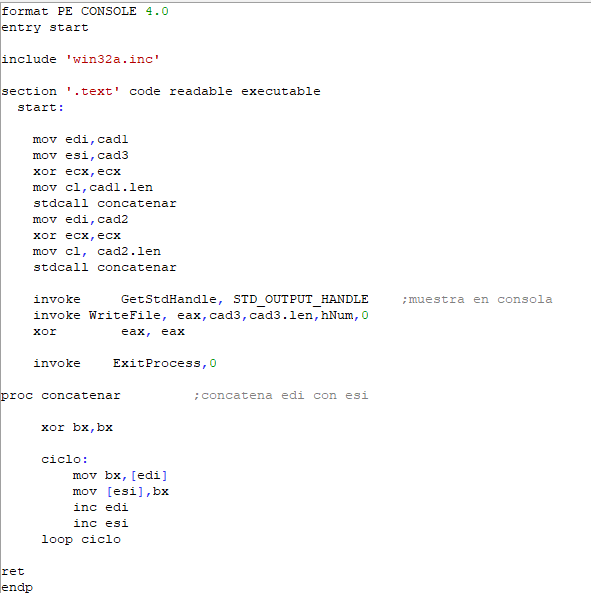
Ejercicios Usando FASM (Consola)

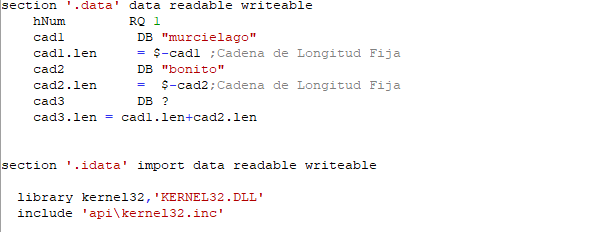
1.- Leer cadenas estructuradas y leer cadenas de longitud fija.



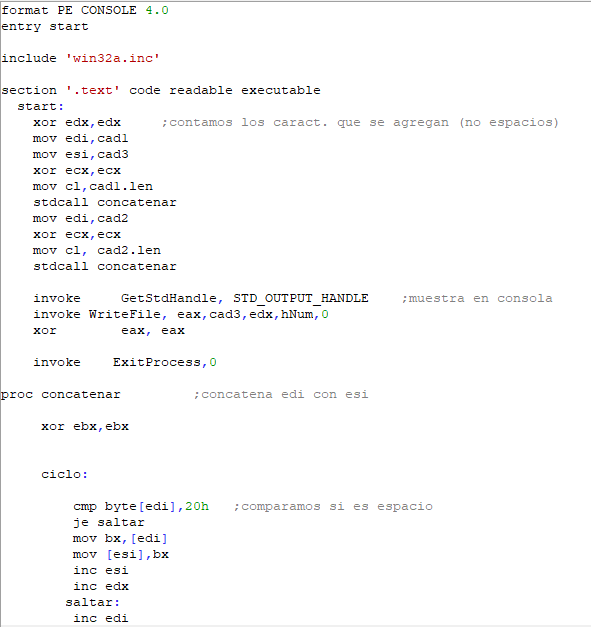
2.- Concatenar cadenas

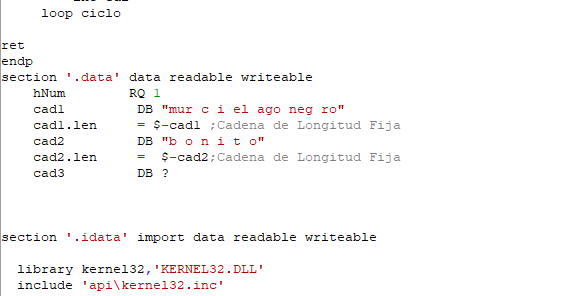
a) longitud fija con longitud fija



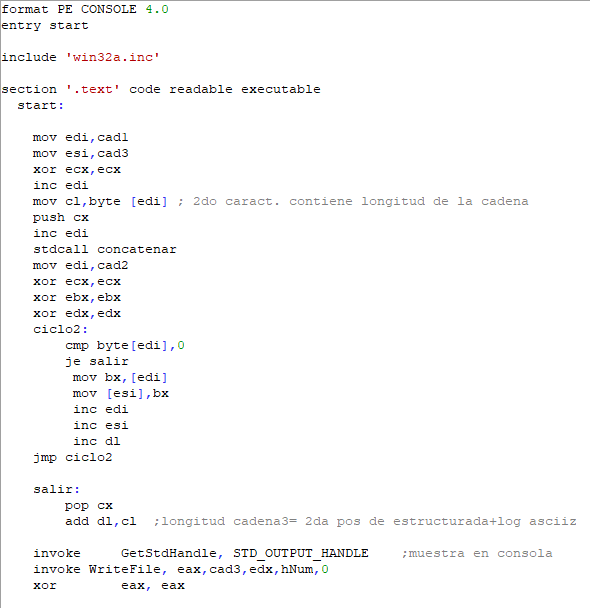


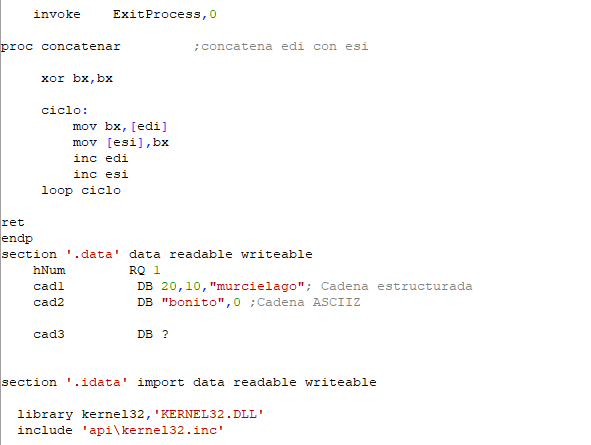
b) longitud fija con longitud fija con supresión de espacios en blanco



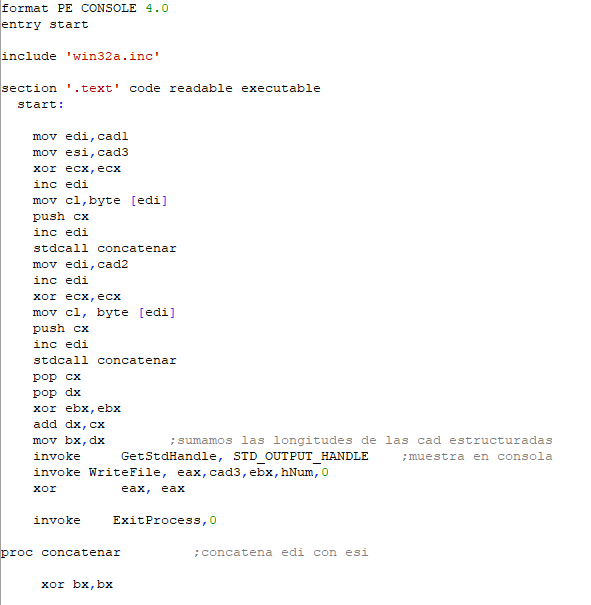


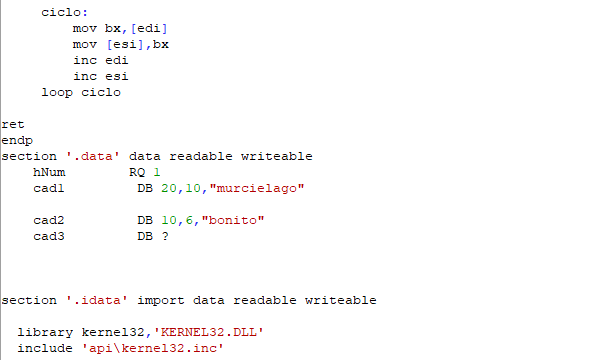
c) estructurada con ASCIIZ





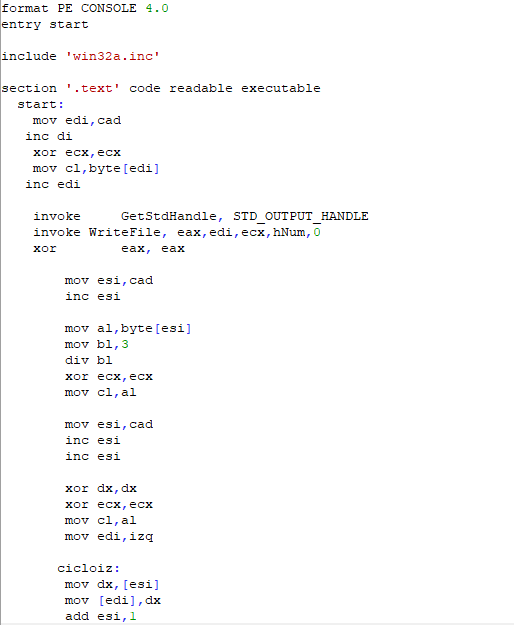
d) estructurada con estructurada

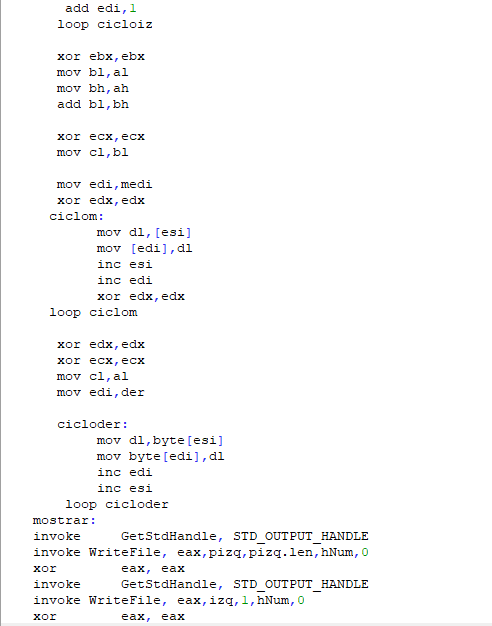


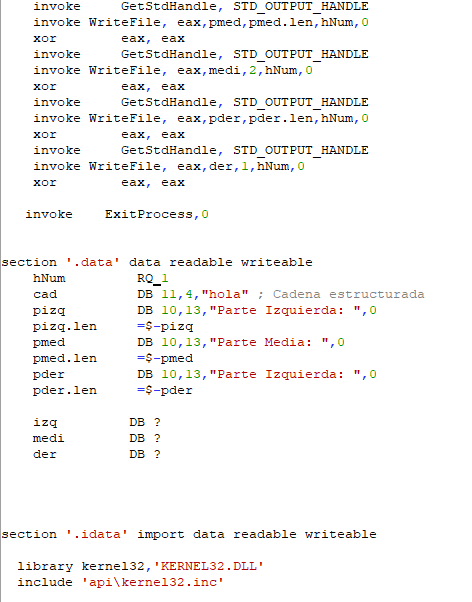


3.- Obtener sub-cadenas; parte izquierda, derecha y central de:

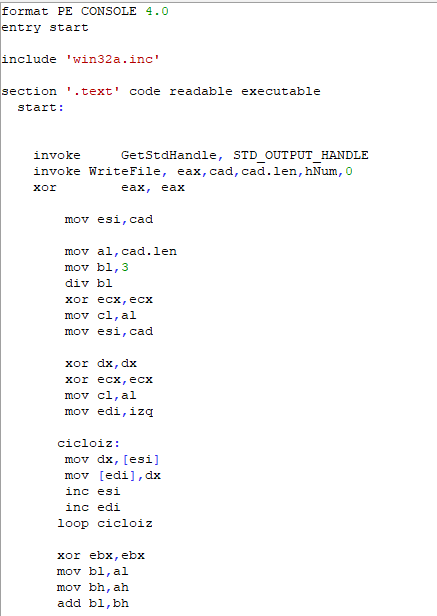
a) cadena estructurada

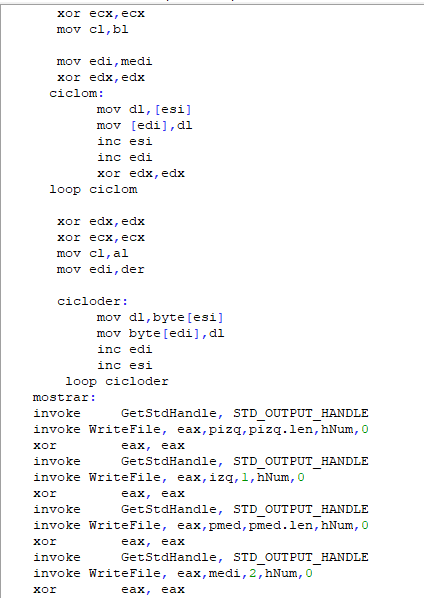


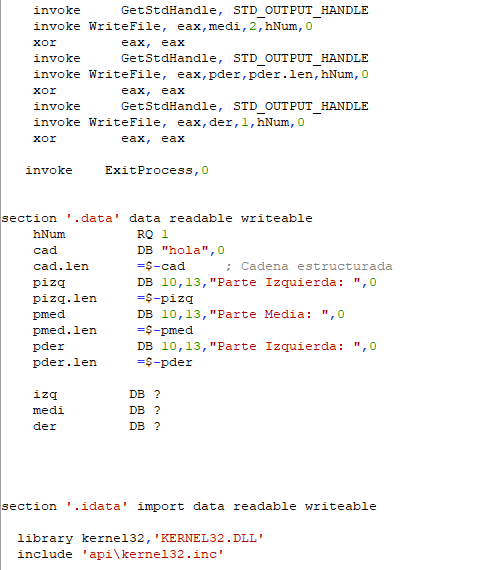




b) cadena ASCIIZ

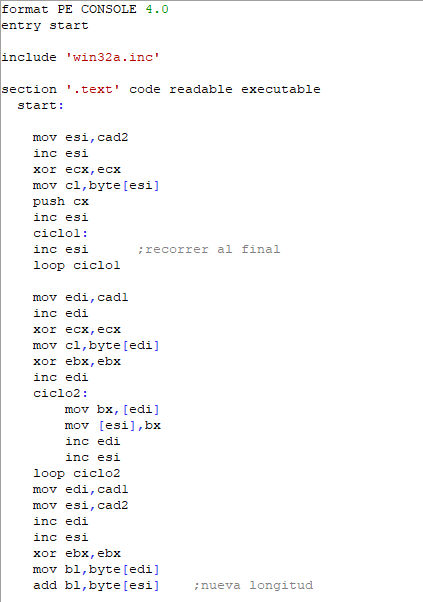


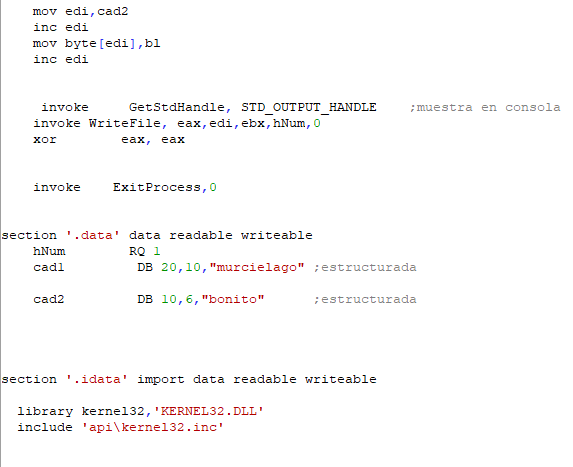




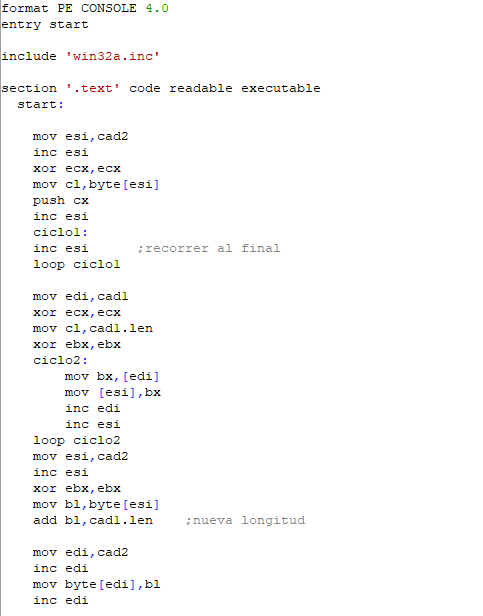
4.- Asignar cadenas:

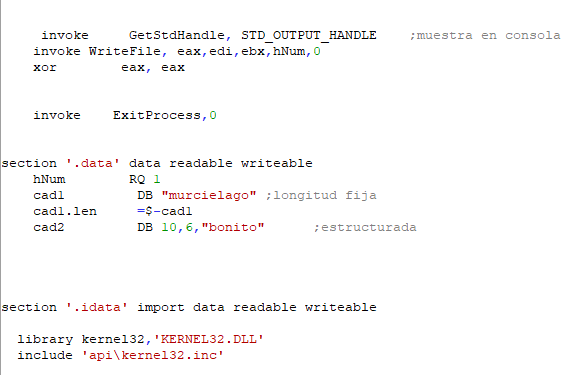
a) estructurada a estructurada



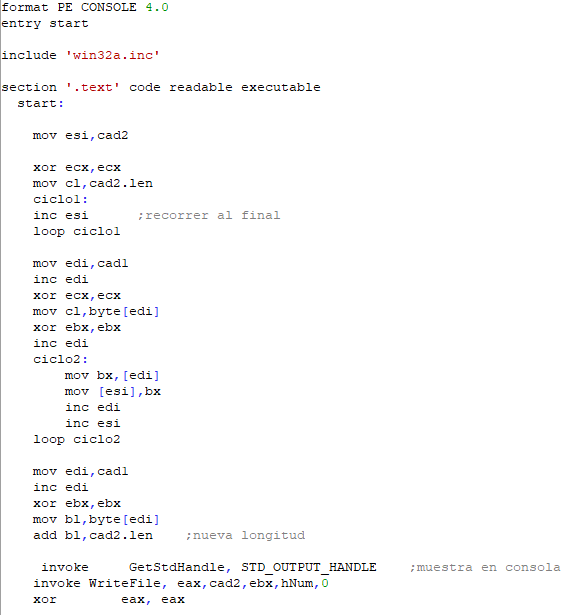


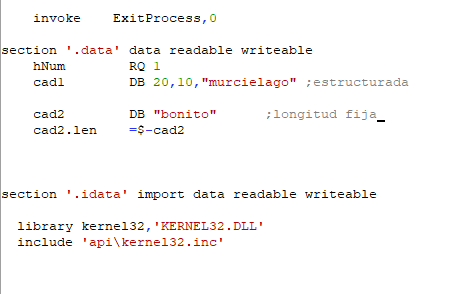
b) longitud fija a estructurada



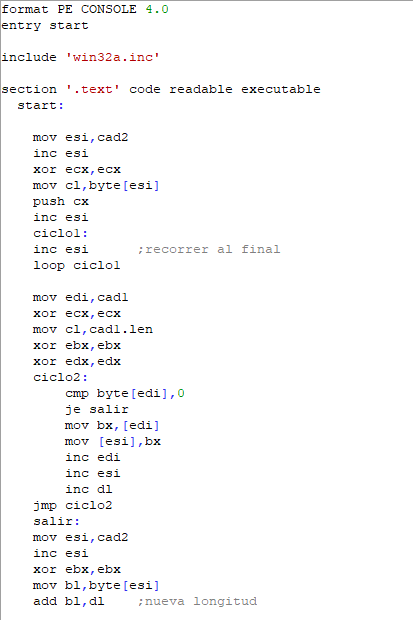


c) estructurada a longitud fija



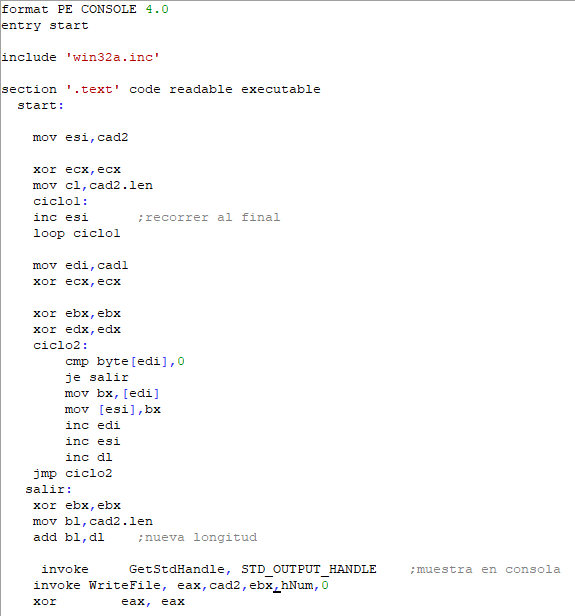


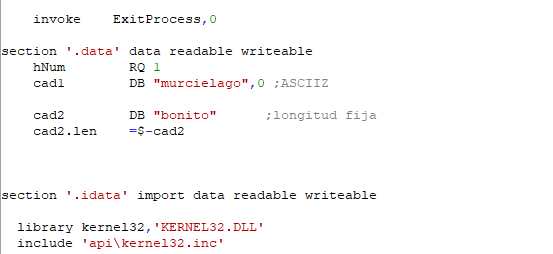
d) ASCIIZ a estructurada



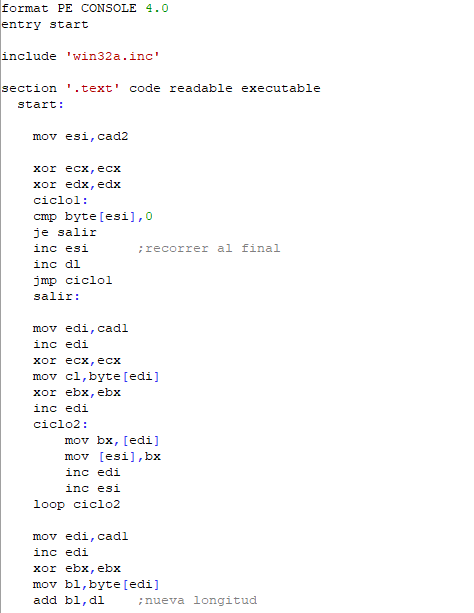


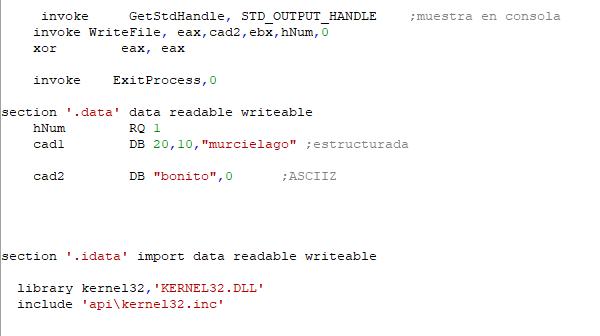
e) ASCIIZ a longitud fija





f) estructurada a ASCIIZ





Nota: archivos .ASM y .EXE adjuntados en un .ZIP

\*De preferencia **compilar con Flat Assembler**, u similar y **ejecutar el .EXE desde consola** (CMD)