```
MODEL small
.STACK 100h
```

Establece el modelo de memoria del programa. Un modelo **small** usa máximo **64Kb** de código y **64Kb** de datos. Los registros **DS** y **SS** apuntan al mismo segmento.

Se reservan 256 bytes para la pila. La **h** indica que 100 está en formato hexadecimal.

```
;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here
```

Se usa un **;** para incluir comentarios en el código.

Directivas **EQU** y **=** se colocan en esta parte, para declarar constantes y valores simbólicos.

```
INCLUDE procs.inc
```

Se Incluye una biblioteca de funciones. En el archivo **procs.inc** está la declaración de los procedimientos externos.

```
LOCALS
```

Establece que las etiquetas usadas en un procedimiento son locales a él. De esta manera, se puede usar el mismo nombre de etiqueta en distintos procedimientos.

```
.DATA
```

Inicio del segmento **DATA**. En esta sección se declaran los datos del programa.

```
mens      db  'Hola Mundo',0
```

Declaración de la cadena de caracteres **mens**, la cual está inicializada con "**Hola mundo**",**0**. Los valores encerrados entre comillas (ya sean sencillas o dobles) son caracteres ASCII. El cero, al no estar entre comillas, es un valor *crudo*, es decir, en memoria se almacena un 0, **no** su correspondiente ASCII (30h).

```
.CODE
;-----   Insert program, subrutine call, etc., here
```

Inicio del segmento **CODE**. En esta sección se colocan las instrucciones del programa.

```
Principal     PROC
```

La directiva **PROC** se usa para indicar el inicio de un procedimiento. Este procedimiento se llama **Principal**.

```
mov ax,@data          ; Inicializar DS a la direccion
mov ds,ax             ; del segmento de datos (.DATA)
```

@data se refiere al segmento **DATA**. Se inicializa **DS** para que apunte al área de memoria con los datos del programa.

```
call clrscr
```

La instrucción **call** se usa para invocar un procedimiento. En esta sentencia se está invocando al procedimiento externo **clrscr**, declarado en **procs.inc** y definido en la biblioteca **PCLIB06.lib**, el cual borra el contenido actual en pantalla.

```
mov dx, offset mens
```

La directiva **offset** se usa para obtener la dirección en memoria (desplazamiento) de un símbolo definido en el programa. Puede ser una variable como en este caso, o un procedimiento.

```
call puts
```

Invocación del procedimiento externo **puts**. Este despliega en pantalla una cadena de caracteres direccionada por **DS:DX**. Imprime carácter por carácter hasta llegar a un 0 *crudo*.

```
call getch
```

Invocación del procedimiento externo **getch**. Este detiene la ejecución hasta que ocurra el evento de recibir un carácter del buffer del teclado. El carácter se almacena en **AL** y no se despliega en pantalla.

Coloquialmente se llama *detener pantalla* cuando se coloca un **getch** al final del programa y no se hace uso del carácter recibido.

```
mov ah,04ch       ; fin de programa
mov al,0
int 21h
```

Invocación de la interrupción **21h** servicio **4Ch**. Es la terminación del programa, retornando el control a quien lo invocó (DOS en nuestro caso). Estas instrucciones se colocan una vez en el programa, al final del procedimiento principal. Todos los demás procedimientos deben terminar con la instrucción **ret**.

```
ENDP
```

La directiva **ENDP** se usa para indicar el fin de un procedimiento.

```
; incluir procedimientos
; ejemplo:
; funcionX PROC ; < -- Indica a TASM el inicio de un procedimiento
;             ;
;             ; < --- contenido del procedimiento
;          ret
;         ENDP; < -- Indica a TASM el fin del procedimiento

      END
```

La directiva **END** se usa para indicar el fin del programa.

Desarrollando programas en ensamblador con TASM

Generalidades:

a) Por defecto, en **TASM** los números están en formato decimal. Para hexadecimales, se coloca una h al final del número.

b) Todos los números deben iniciar con un dígito 0 – 9, incluyendo los hexadecimales. Si un número hexadecimal inicia con una letra, por ejemplo, FFh, se debe anteceder un 0, quedando 0FFh.

c) Para reservar espacio en memoria para datos y variables, coloque en la sección DATA:
   - una etiqueta (nombre) del espacio en memoria,
   - el tamaño del dato,
   - el valor inicial.

   Ejemplos:

```
suma db 0 ; dato de 1 byte inicializado en 0.
lado dw 8 ; dato de 2 bytes inicializado en 8.
area dw ? ; dato de 2 bytes no inicializado.
```

```
cadena db 32 dup(?) ; 32 posiciones de memoria de 1 byte
reservadas, no inicializadas.
promedios dw 10 dup(0) ; 10 posiciones de memoria de 2 bytes
reservadas, todas inicializadas en 0.
precios dw 451, 110, 56, 439, 1180, 900, 1270 ; 7 posiciones de
memoria de 2 bytes reservadas, cada una inicializada.
```

Para trabajar con TASM y TLINK es necesario usar el emulador DOSBox. Realice los siguientes pasos:

1. Cree una carpeta de trabajo, por ejemplo, C:\OCLE
2. Descargue de Moodle los archivos TASM, TLINK, PCLIB06, procs.inc y formato.asm y guárdelos en la carpeta de trabajo.
3. Virtualice en DOSBox la carpeta, por medio de los comandos:
   `Mount X C:\OCLE`
   `X:`
   Puede sustituir `X` por otra denominación de disco disponible en su sistema.
4. Ensamble la plantilla por medio de TASM, ingrese en DOSBox:
   `TASM formato.asm`
   Se generará el archivo:
   - *formato.obj*: Traducción del código fuente.
5. Encadene la plantilla por medio de TLINK, ingrese en DOSBox:
   `TLINK formato.obj,,,PCLIB06.lib`
   Se generarán los archivos:
   - *formato.exe*: Ejecutable del programa, contiene también los procedimientos externos.
   - *formato.map*: Describe las direcciones lógicas donde se encuentran los segmentos del programa y sus tamaños.
6. Ejecute el programa, ingrese en DOSBox:
   `formato.exe`

Biblioteca de funciones PCLIB06

Contiene las siguientes funciones. En el archivo procs.inc se puede ver también la descripción de los procedimientos y sus entradas y salidas.

1. clrscr: Borra el contenido actual de la pantalla.
2. putchar: Despliega en pantalla el carácter ASCII en AL en la posición actual del cursor.
3. puts: Despliega en pantalla una cadena de caracteres ASCII terminada en 0 (null). El registro DX contiene el apuntador a la cadena a imprimir.
4. getchar: Recibe un carácter del teclado, el cual se retorna en AL. La ejecución se pausa en espera del carácter. Una vez recibido, se despliega en pantalla.
5. getch: Similar al anterior, pero el carácter no tiene eco en pantalla.
6. gotoxy: Coloca el cursor en la posición dada por BX, donde BH es la posición X (columna) y BL la posición Y (renglón).