Universidad de San Carlos de Guatemala
Primer semestre
Facultad de ingeniería
Escuela de Ciencias y Sistemas
Laboratorio Arquitectura de Computadores y Ensambladores 1


# MANUAL TECNICO

**Roberto Carlos Gómez Donis 202000544**

**Guatemala, 28 de junio de 2023.**

El presente manual técnico tiene como objetivo brindar una guía detallada sobre el código en lenguaje ensamblador proporcionado. En este manual, se explicarán las diferentes secciones del código y su funcionalidad. A medida que avancemos, se proporcionarán descripciones claras y ejemplos para facilitar su comprensión. El programa comienza en la etiqueta "inicio". A continuación, se cargan las direcciones de memoria de las cadenas que se desean mostrar en los registros DX y se utiliza la instrucción "mov AH, 09" para indicar que se desea mostrar el contenido de la cadena en pantalla. Finalmente, se realiza la interrupción 21h mediante la instrucción "int 21", lo que provoca que el sistema operativo muestre el mensaje en pantalla.

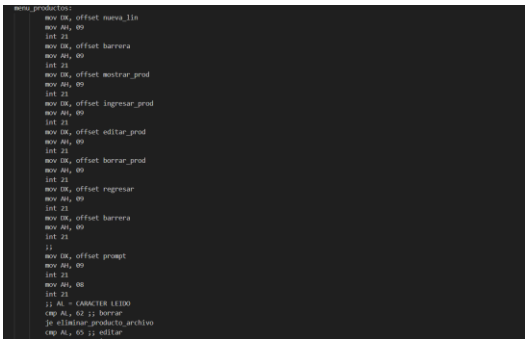
```
inicio:
        mov DX, offset barrera_menu
        mov AH, 09
        int 21
        mov DX, offset usac
        mov AH, 09
        int 21
        mov DX, offset facultad
        mov AH, 09
        int 21
        mov DX, offset curso
        mov AH, 09
        int 21
        mov DX, offset nombre
        mov AH, 09
        int 21
        mov DX, offset carne
        mov AH, 09
        int 21
        mov DX, offset barrera
        mov AH, 09
        int 21
        ;;
```

Se utiliza la instrucción mov para cargar la dirección de memoria de la cadena "nueva_lin" en el registro DX. Luego, se utiliza mov AH, 09 para indicar que se desea mostrar el contenido de la cadena en pantalla utilizando la interrupción 21h del sistema operativo Después de mostrar las opciones del menú, se muestra la cadena "prompt" que solicita al usuario ingresar una opción. Esto se logra utilizando las mismas instrucciones mov, mov AH, 09 e int 21. Si el valor en AL coincide con 70 (código ASCII para la letra 'F'), se salta a la etiqueta "menu_productos". Si coincide con 76 ('L'), se salta a la etiqueta "menu_ventas". Si coincide con 68 ('D'), se salta a la etiqueta "generar_catalogo". Si coincide con 73 ('I'), se salta a la etiqueta "fin". De lo contrario, se salta de nuevo a la etiqueta "menu_principal" para mostrar nuevamente el menú.

```
menu_principal:
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        mov DX, offset barrera
        mov AH, 09
        int 21
        ;; Menú
        mov DX, offset productos
        mov AH, 09
        int 21
        mov DX, offset ventas
        mov AH, 09
        int 21
        mov DX, offset herramientas
        mov AH, 09
        int 21
        mov DX, offset salir
        mov AH, 09
        int 21
        mov DX, offset barrera
        mov AH, 09
        int 21
        mov DX, offset prompt
        mov AH, 09
        int 21
        mov AH, 08
        int 21
        ;; AL = CARACTER LEIDO
        cmp AL, 70
        je menu_productos
        cmp AL, 76
        je menu_ventas
        cmp AL, 68
        je generar_catalogo
        cmp AL, 73
        je fin
```

A continuación, se realizan comparaciones utilizando la instrucción cmp para verificar qué opción ha seleccionado el usuario. Dependiendo del valor en el registro AL, se ejecutan diferentes secciones del código. Si el valor en AL coincide con 62 (código ASCII para la letra 'b'), se salta a la etiqueta "eliminar_producto_archivo".

- Si coincide con 65 ('e'), se ejecuta una comparación adicional.
- Si coincide con 69 ('i'), se salta a la etiqueta "ingresar_producto_archivo".
- Si coincide con 6d ('m'), se salta a la etiqueta "mostrar_productos_archivo".
- Si coincide con 72 ('r'), se salta a la etiqueta "menu_principal".
- De lo contrario, se salta nuevamente a la etiqueta "menu_productos" para mostrar nuevamente el menú.




Este fragmento del código se encarga de mostrar las instrucciones necesarias para ingresar un nuevo producto en el archivo. El siguiente paso sería permitir que el usuario ingrese los datos del producto y realizar las operaciones necesarias para almacenarlo correctamente en el archivo.

```
ingresar_producto_archivo:
        mov DX, offset titulo_producto
        mov AH, 09
        int 21
        mov DX, offset sub_prod
        mov AH, 09
        int 21
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
```

En este fragmento de código, se muestra el proceso de solicitud del código de producto al usuario y se realiza una validación del tamaño del código ingresado.

```
pedir_de_nuevo_codigo:
        mov DX, offset prompt_code
        mov AH, 09
        int 21
        mov DX, offset buffer_entrada
        mov AH, 0a
        int 21
        ;;; verificar que el tamaño del codigo no sea mayor a 5
        mov DI, offset buffer_entrada
        inc DI
        mov AL, [DI]
        cmp AL, 00
        je  pedir_de_nuevo_codigo
        cmp AL, 05
        jb  aceptar_tam_cod  ;; jb --> jump if below
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        jmp pedir_de_nuevo_codigo
        ;;; mover al campo codigo en la estructura producto
```

En este punto del código, se ha aceptado el tamaño del código ingresado y se ha posicionado en el contenido del buffer de entrada para acceder al código del producto. A partir de aquí, se pueden realizar operaciones adicionales, como la lectura del código y su procesamiento.

```
aceptar_tam_cod:
        mov SI, offset cod_prod
        mov DI, offset buffer_entrada
        inc DI
        mov CH, 00
        mov CL, [DI]
        inc DI  ;; me posiciono en el contenido del buffer
```

En este punto del código, se ha copiado exitosamente el código del producto desde el buffer de entrada hacia otra ubicación de memoria. A partir de aquí, se pueden realizar operaciones adicionales con el código copiado.

```
copiar_codigo:  mov AL, [DI]
      mov [SI], AL
      inc SI
      inc DI
      loop copiar_codigo  ;; restarle 1 a CX, verificar que CX no sea 0, si no es 0 va a la etiqueta,
      ;;; la cadena ingresada en la estructura
      ;;;
      mov DX, offset nueva_lin
      mov AH, 09
      int 21
      ;;; PEDIR NOMBRE
```

En este punto del código, se ha solicitado al usuario que ingrese el nombre del producto y se ha validado el tamaño del nombre ingresado. A partir de aquí, se pueden realizar operaciones adicionales con el nombre ingresado.

```
pedir_de_nuevo_nombre:
      mov DX, offset prompt_name
      mov AH, 09
      int 21
      mov DX, offset buffer_entrada
      mov AH, 0a
      int 21
      ;;; verificar que el tamaño del codigo no sea mayor a 5
      mov DI, offset buffer_entrada
      inc DI
      mov AL, [DI]
      cmp AL, 00
      je  pedir_de_nuevo_nombre
      cmp AL, 20
      jb  aceptar_tam_nom
      mov DX, offset nueva_lin
      mov AH, 09
      int 21
      jmp pedir_de_nuevo_nombre
```

En este punto del código, se ha aceptado y almacenado el tamaño del nombre del producto. El siguiente paso sería utilizar esta información para realizar operaciones adicionales con el nombre ingresado

```
aceptar_tam_nom:
        mov SI, offset cod_name
        mov DI, offset buffer_entrada
        inc DI
        mov CH, 00
        mov CL, [DI]
        inc DI  ;; me posiciono en el contenido del buffer
```

En este punto del código, se ha copiado con éxito el nombre del producto ingresado por el usuario en una ubicación de memoria específica. A continuación, se pueden realizar operaciones adicionales con el nombre del producto copiado.

```
copiar_nombre:  mov AL, [DI]
      mov [SI], AL
      inc SI
      inc DI
      loop copiar_nombre  ;; restarle 1 a CX, verificar que CX no sea 0, si no es 0 va a la etiqueta,
      ;;; la cadena ingresada en la estructura
      ;;;
      mov DX, offset nueva_lin
      mov AH, 09
      int 21
```

Una vez que se ha validado el tamaño del precio ingresado por el usuario, se puede continuar con el procesamiento del precio según lo requerido en el programa.

```
pedir_de_nuevo_precio:
        mov DX, offset prompt_price
        mov AH, 09
        int 21
        mov DX, offset buffer_entrada
        mov AH, 0a
        int 21
        ;;; verificar que el tamaño del codigo no sea mayor a 5
        mov DI, offset buffer_entrada
        inc DI
        mov AL, [DI]
        cmp AL, 00
        je  pedir_de_nuevo_precio
        cmp AL, 06  ;; tamaño máximo del campo
        jb  aceptar_tam_precio ;; jb --> jump if below
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        jmp pedir_de_nuevo_precio
```

Una vez que se ha aceptado y obtenido el tamaño del precio, se está listo para copiar el contenido del precio en la estructura del producto.

```
aceptar_tam_unidades:
        mov SI, offset cod_units
        mov DI, offset buffer_entrada
        inc DI
        mov CH, 00
        mov CL, [DI]
        inc DI  ;; me posiciono en el contenido del buffer
```

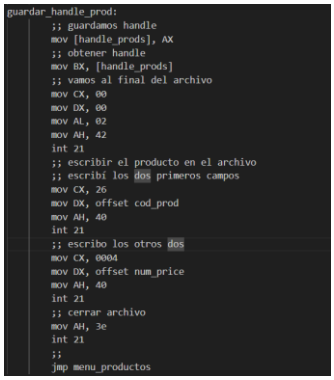
Este fragmento de código muestra cómo se copian las unidades del producto ingresadas por el usuario en la estructura del producto, se procesa el número de unidades y se prepara el archivo para su escritura.

```
copiar_unidades:
        mov AL, [DI]
        mov [SI], AL
        inc SI
        inc DI
        loop copiar_unidades  ;; restarle 1 a CX, verificar que CX no sea 0, si no es 0 va a la etiqueta,
        ;;
        mov DI, offset cod_units
        call cadenaAnum
        ;; AX -> numero convertido
        mov [num_units], AX
        ;;
        mov DI, offset cod_units
        mov CX, 0005
        call memset
        ;; finalizó pedir datos de producto
        ;;
        ;;
        ;;
        ;;
        ;; GUARDAR EN ARCHIVO
        ;; probar abrirlo normal
        mov AL, 02
        mov AH, 3d
        mov DX, offset archivo_prods
        int 21
        ;; si no lo cremos
        jc  crear_archivo_prod
        ;; si abre escribimos
        jmp guardar_handle_prod
```

En esta sección del código, se crea el archivo de productos en caso de que no exista previamente

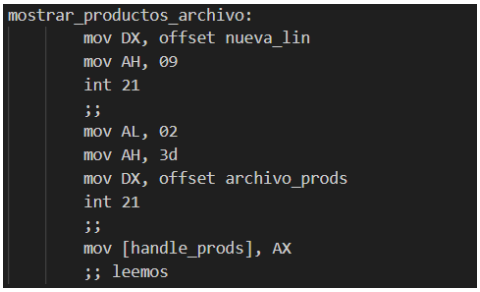
```
crear_archivo_prod:
        mov CX, 0000
        mov DX, offset archivo_prods
        mov AH, 3c
        int 21
```

En esta sección del código, se guardan los datos del producto ingresado en el archivo de productos. Se asegura de posicionar el puntero al final del archivo, escribir los campos correspondientes y cerrar el archivo adecuadamente.



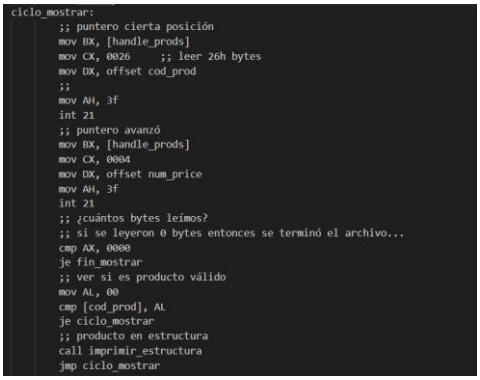
```
guardar_handle_prod:
        ;; guardamos handle
        mov [handle_prods], AX
        ;; obtener handle
        mov BX, [handle_prods]
        ;; vamos al final del archivo
        mov CX, 00
        mov DX, 00
        mov AL, 02
        mov AH, 42
        int 21
        ;; escribir el producto en el archivo
        ;; escribí los dos primeros campos
        mov CX, 26
        mov DX, offset cod_prod
        mov AH, 40
        int 21
        ;; escribo los otros dos
        mov CX, 0004
        mov DX, offset num_price
        mov AH, 40
        int 21
        ;; cerrar archivo
        mov AH, 3e
        int 21
        ;;
        jmp menu_productos
```

A continuación, se debe implementar la lógica para leer los productos del archivo y mostrarlos en pantalla. Esta parte del código está incompleta y se debe agregar la lógica correspondiente para realizar esta operación. Puedes utilizar las funciones de la interrupción 21h, como la función 3Fh para leer datos del archivo y la función 09h para mostrar los datos en pantalla.



```
mostrar_productos_archivo:
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        ;;
        mov AL, 02
        mov AH, 3d
        mov DX, offset archivo_prods
        int 21
        ;;
        mov [handle_prods], AX
        ;; leemos
```

Se verifica si el producto leído es válido. En este caso, se compara el contenido de cod_prod con el valor 0 para determinar si es un producto válido. Si es igual a 0, se salta a la etiqueta ciclo_mostrar para continuar con la siguiente iteración del ciclo. Si el producto es válido, se llama a la función imprimir_estructura para mostrar la información del producto en pantalla. Se vuelve al inicio del ciclo utilizando la instrucción jmp ciclo_mostrar para seguir leyendo y mostrando los productos restantes en el archivo.



```
ciclo_mostrar:
        ;; puntero cierta posición
        mov BX, [handle_prods]
        mov CX, 0026    ;; leer 26h bytes
        mov DX, offset cod_prod
        ;;
        mov AH, 3f
        int 21
        ;; puntero avanzó
        mov BX, [handle_prods]
        mov CX, 0004
        mov DX, offset num_price
        mov AH, 3f
        int 21
        ;; ¿cuántos bytes leímos?
        ;; si se leyeron 0 bytes entonces se terminó el archivo...
        cmp AX, 0000
        je fin_mostrar
        ;; ver si es producto válido
        mov AL, 00
        cmp [cod_prod], AL
        je ciclo_mostrar
        ;; producto en estructura
        call imprimir_estructura
        jmp ciclo_mostrar
```

El código proporcionado parece formar parte de un programa más grande que involucra la gestión de productos. Se sugiere revisar el resto del programa para comprender cómo se utiliza este código y cómo se relaciona con otras secciones.

```
fin_mostrar:
        jmp menu_productos
eliminar_producto_archivo:
        mov DX, 0000
        mov [puntero_temp], DX
pedir_de_nuevo_codigo2:
        mov DX, offset prompt_code
        mov AH, 09
        int 21
        mov DX, offset buffer_entrada
        mov AH, 0a
        int 21
        ;;
        mov DI, offset buffer_entrada
        inc DI
        mov AL, [DI]
        cmp AL, 00
        je  pedir_de_nuevo_codigo2
        cmp AL, 05
        jb  aceptar_tam_cod2  ;; jb --> jump if below
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        jmp pedir_de_nuevo_codigo2
        ;;; mover al campo codigo en la estructura producto
```

Se realiza una lectura/escritura en el archivo "archivo_prods" utilizando la función 3Dh de la interrupción 21h. El archivo se abre en modo lectura/escritura y el handle resultante se guarda en la variable "handle_prods".

```
aceptar_tam_cod2:
        mov SI, offset cod_prod_temp
        mov DI, offset buffer_entrada
        inc DI
        mov CH, 00
        mov CL, [DI]
        inc DI  ;; me posiciono en el contenido del buffer
copiar_codigo2: mov AL, [DI]
        mov [SI], AL
        inc SI
        inc DI
        loop copiar_codigo2  ;; restarle 1 a CX, verificar que CX no sea 0, si no es 0 va a la etiqueta,
        ;;; la cadena ingresada en la estructura
        ;;;
        mov DX, offset nueva_lin
        mov AH, 09
        int 21
        ;;
        mov AL, 02              ;;;<<<<<  lectura/escritura
        mov DX, offset archivo_prods
        mov AH, 3d
        int 21
        mov [handle_prods], AX
```

A continuación, se verifica si el producto encontrado es válido comparando el campo "cod_prod" con el valor 0 en el registro AL. Si es igual a cero, se salta a la etiqueta "ciclo_encontrar" para buscar el siguiente producto. Después de comparar las cadenas, se compara el valor de DL con 0FFh para verificar si las cadenas son iguales. Si son iguales, se salta a la etiqueta "borrar_encontrado" para borrar el producto encontrado.

```
ciclo_encontrar:
        mov BX, [handle_prods]
        mov CX, 26
        mov DX, offset cod_prod
        moV AH, 3f
        int 21
        mov BX, [handle_prods]
        mov CX, 4
        mov DX, offset num_price
        moV AH, 3f
        int 21
        cmp AX, 0000   ;; se acaba cuando el archivo se termina
        je finalizar_borrar
        mov DX, [puntero_temp]
        add DX, 2a
        mov [puntero_temp], DX
        ;;; verificar si es producto válido
        mov AL, 00
        cmp [cod_prod], AL
        je ciclo_encontrar
        ;;; verificar el código
        mov SI, offset cod_prod_temp
        mov DI, offset cod_prod
        mov CX, 0005
        call cadenas_iguales
        ;;;; <<
        cmp DL, 0ff
        je borrar_encontrado
        jmp ciclo_encontrar
```

El puntero temporal se ajusta para apuntar al producto encontrado en el archivo. Se borra el contenido del producto en el archivo, reemplazándolo con bytes nulos. El puntero se restablece a su posición original. El archivo se cierra. En resumen, esta sección busca el producto en el archivo, lo elimina reemplazando su contenido con bytes nulos y cierra el archivo.

```
borrar_encontrado:
        mov DX, [puntero_temp]
        sub DX, 2a
        mov CX, 0000
        mov BX, [handle_prods]
        mov AL, 00
        mov AH, 42
        int 21
        ;;; puntero posicionado
        mov CX, 2a
        mov DX, offset ceros
        mov AH, 40
        int 21
finalizar_borrar:
        mov BX, [handle_prods]
        mov AH, 3e
        int 21
        jmp menu_productos
```

"menu_ventas": Esta sección muestra el título "Ventas" en la pantalla utilizando la interrupción del servicio de DOS "int 21h, AH=09h". "menu_herramientas": Esta sección es similar a "menu_ventas", pero en este caso muestra el título "Herramientas" y un subtexto relacionado con las herramientas. Después de mostrar estos mensajes, también salta a la etiqueta "fin".

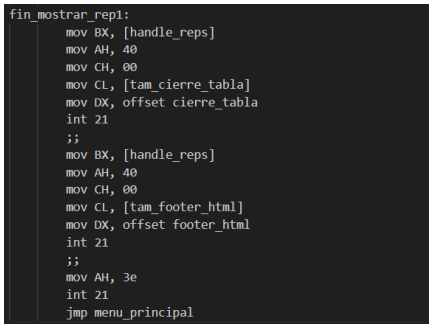
```
menu_ventas:
        mov DX, offset titulo_ventas
        mov AH, 09
        int 21
        mov DX, offset sub_vent
        mov AH, 09
        int 21
        jmp fin
menu_herramientas:
        mov DX, offset titulo_herras
        mov AH, 09
        int 21
        mov DX, offset sub_herr
        mov AH, 09
        int 21
        jmp fin
```

Se utiliza la interrupción del servicio de DOS "int 21h" con la función AH=3Ch para crear un nuevo archivo. El nombre y la extensión del archivo se encuentran en la variable "nombre_rep1". El archivo creado se asigna a un identificador de archivo almacenado en la variable "handle_reps".

```
generar_catalogo:
        mov AH, 3c
        mov CX, 0000
        mov DX, offset nombre_rep1
        int 21
        mov [handle_reps], AX
        mov BX, AX
        mov AH, 40
        mov CH, 00
        mov CL, [tam_encabezado_html]
        mov DX, offset encabezado_html
        int 21
        mov BX, [handle_reps]
        mov AH, 40
        mov CH, 00
        mov CL, [tam_inicializacion_tabla]
        mov DX, offset inicializacion_tabla
        int 21
        ;;
        mov AL, 02
        mov AH, 3d
        mov DX, offset archivo_prods
        int 21
        ;;
        mov [handle_prods], AX
```

Se utiliza el identificador de archivo de productos almacenado en la variable "handle_prods" para leer datos del archivo. Se utiliza la interrupción del servicio de DOS "int 21h" con la función AH=3Fh para leer un bloque de datos del archivo. Se guarda el contenido leído en los campos "cod_prod" y "num_price" que representan el código del producto y el precio respectivamente.


En resumen, esta sección se encarga de finalizar el proceso de generación del catálogo en formato HTML escribiendo el cierre de la tabla y el footer en el archivo de reporte, cerrando el archivo y regresando al menú principal.




En resumen, la función "imprimir_estructura" modifica el campo "cod_name" del producto para incluir el símbolo del dólar, imprime los campos modificados y el precio convertido en pantalla, y luego realiza saltos de línea para separar visualmente los datos.