Práctica #5: Ciclos numéricos y captura básica de cadenas

Adolfo Linares López
José Mario Piñón Arroyo

OBJETIVOS:
- Implementar ciclos numéricos en ensamblador.
- Realizar capturas básicas de cadena en EMU8086 y DosBOX.

Temas Cubiertos:
- Ciclos numéricos.
- Captura básica de cadenas.

Materiales Necesarios:
- emu8086
- DOSBox con MASM
- Documentation for emu8086

DESARROLLO:

1 – CICLOS NUMÉRICOS

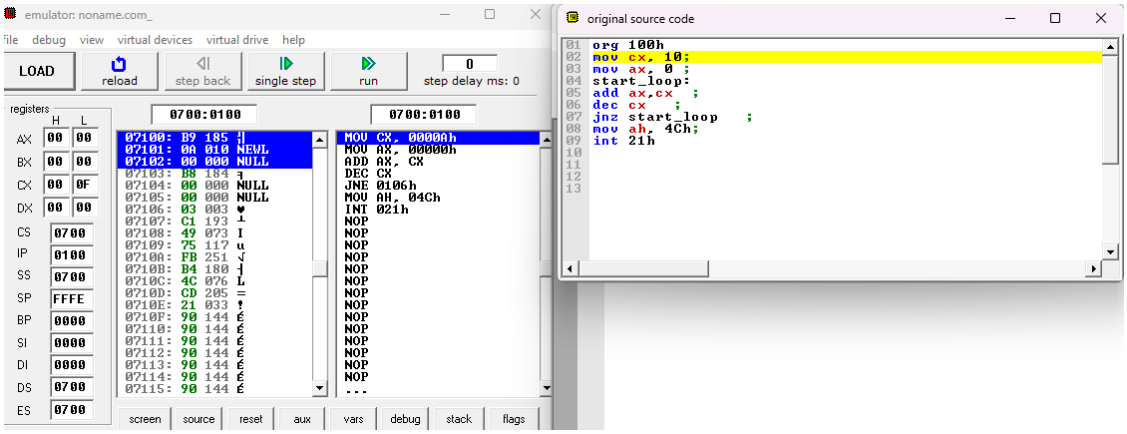
Instrucciones: Correr los siguientes códigos en EMU8086.

Ejercicio 1.

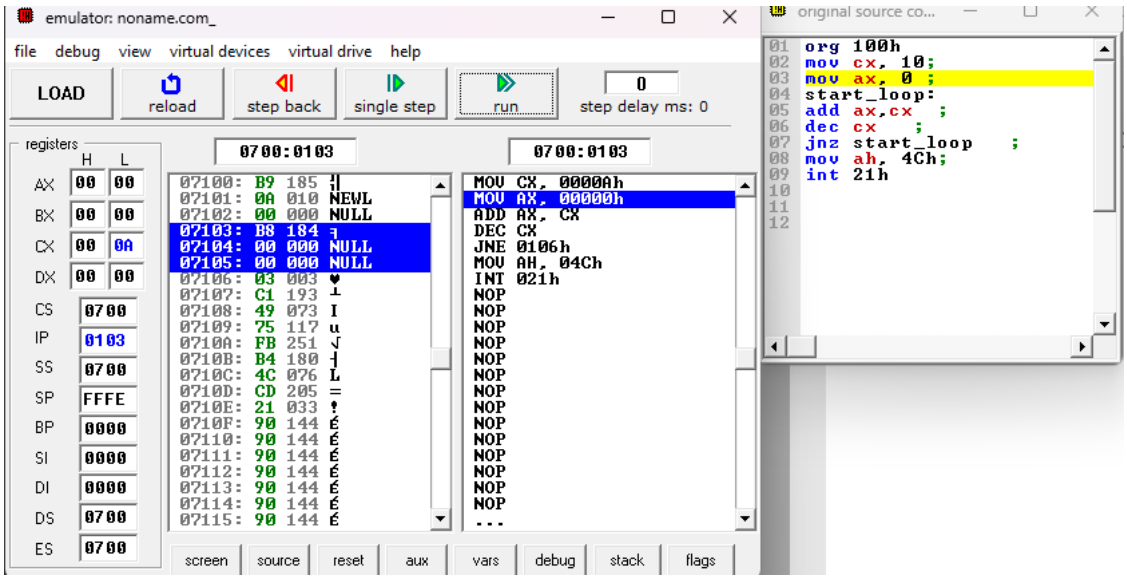
```
org 100h
mov cx, 10          ; Inicializar el contador
mov ax, 0           ; Inicializar el acumulador
start_loop:
    add ax, cx      ; Sumar CX a AX
    dec cx          ; Decrementar CX
    jnz start_loop  ; Saltar a start_loop si CX no es cero
mov ah, 4Ch         ; Terminar el programa
int 21h
```

Preguntas:

- Describir cada acción realiza cada línea de código.

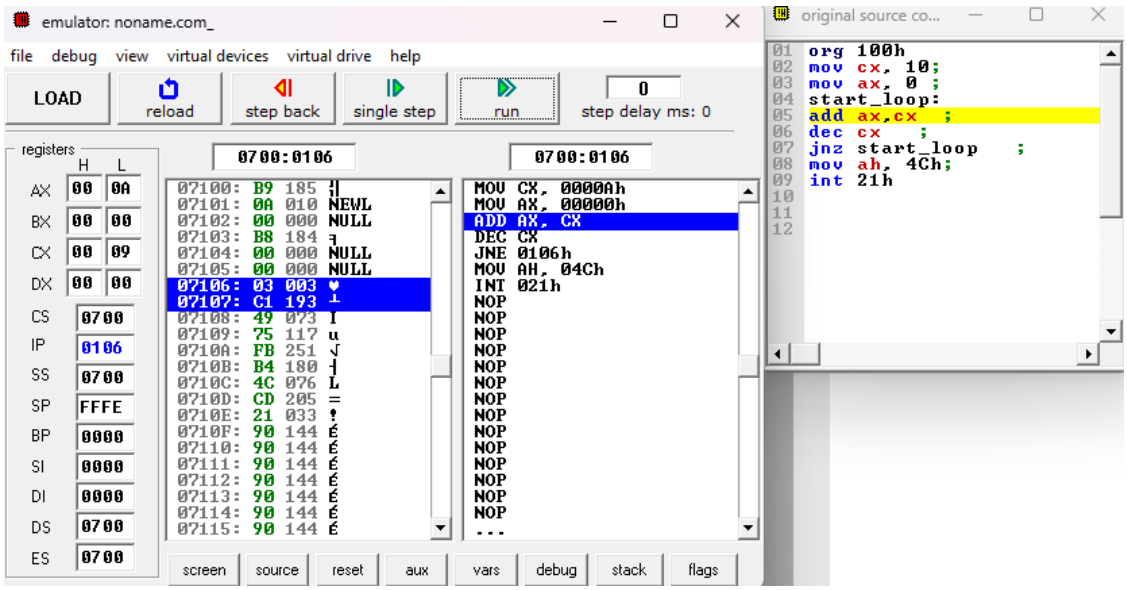


org 100h
Acción: Establece la dirección base del programa en 0x100.
Propósito: En programas COM de DOS, el código comienza en la dirección 0x100 después de la cabecera de 256 bytes.

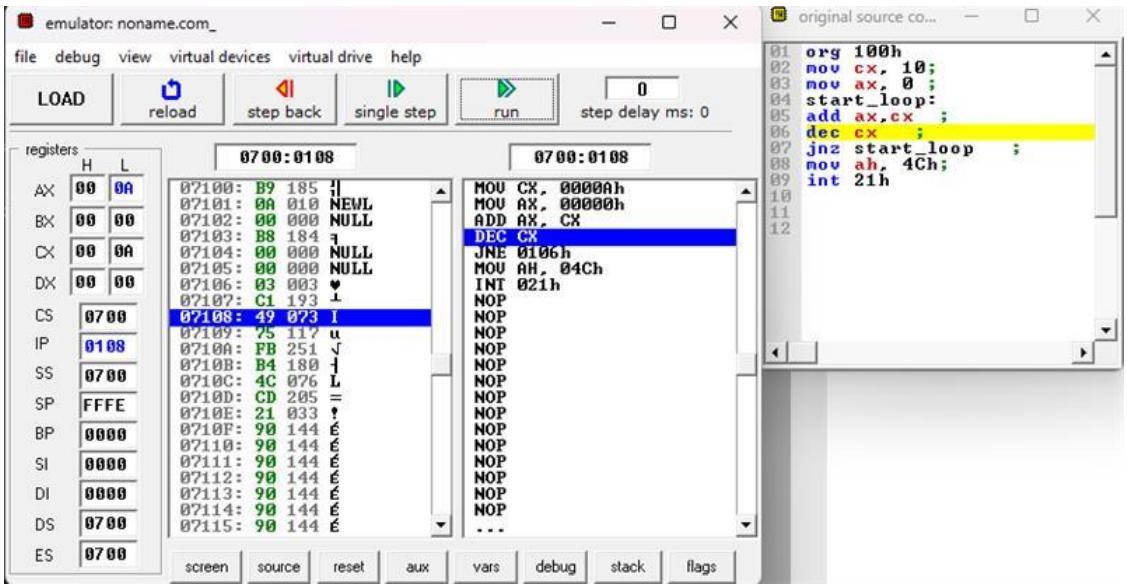


mov cx, 10
Acción: Carga el valor 10 en el registro CX.

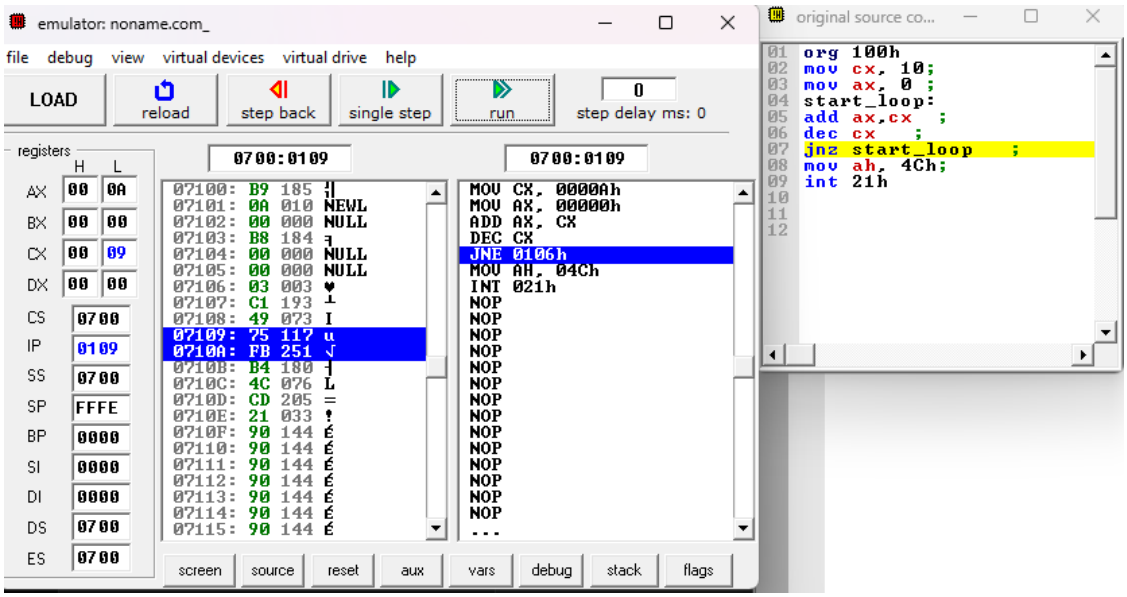
mov ax, 0
Acción: Asigna el valor 0 al registro AX.
Propósito: Inicializa el acumulador donde se sumarán los valores.



start_loop:
add ax, cx
Acción: Suma el valor actual de CX al registro AX.
Propósito: Acumula la suma progresiva de los valores decrecientes de CX

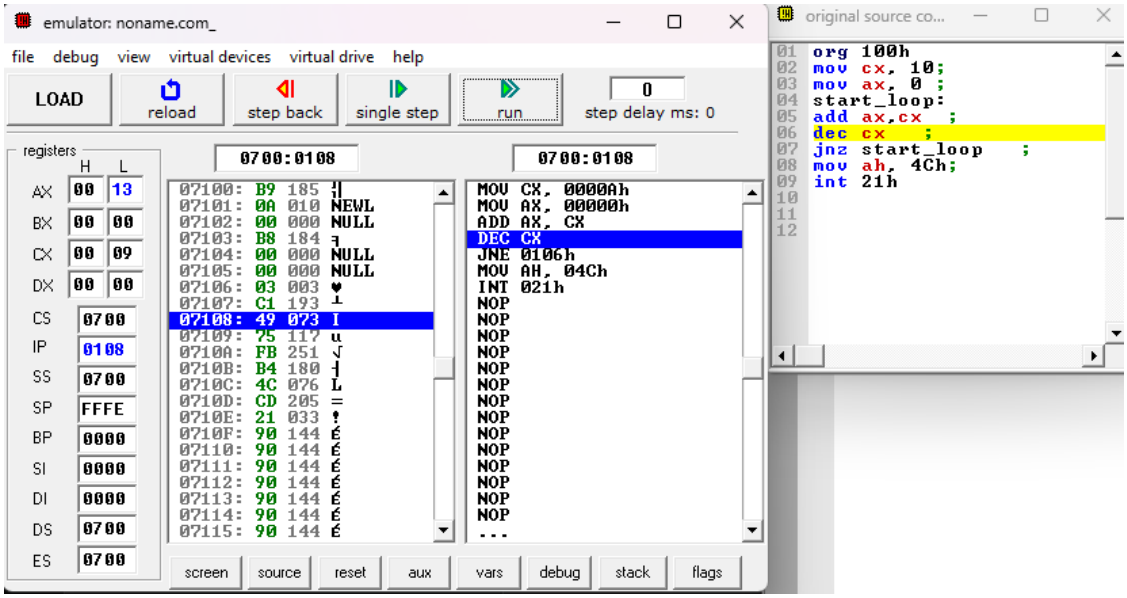


dec cx
Acción: Decrementa el valor de CX en 1.
Propósito: Reduce el contador para eventualmente salir del ciclo



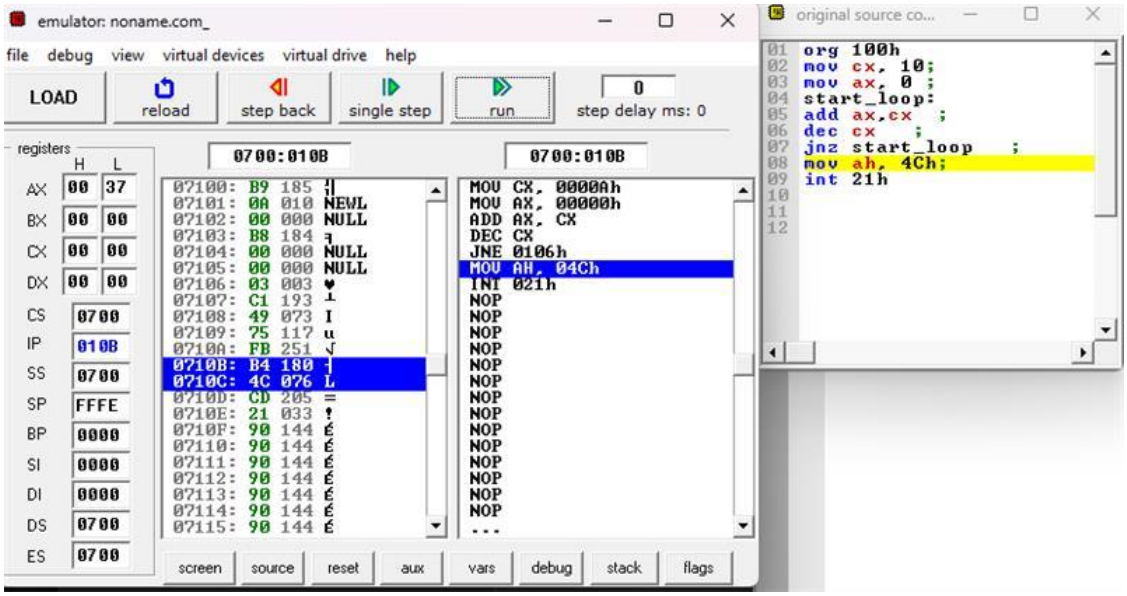
jnz start_loop

Acción: Verifica si CX es diferente de cero (Not Zero).
Si CX ≠ 0, salta a start_loop para repetir el ciclo.
Si CX = 0, el ciclo termina.

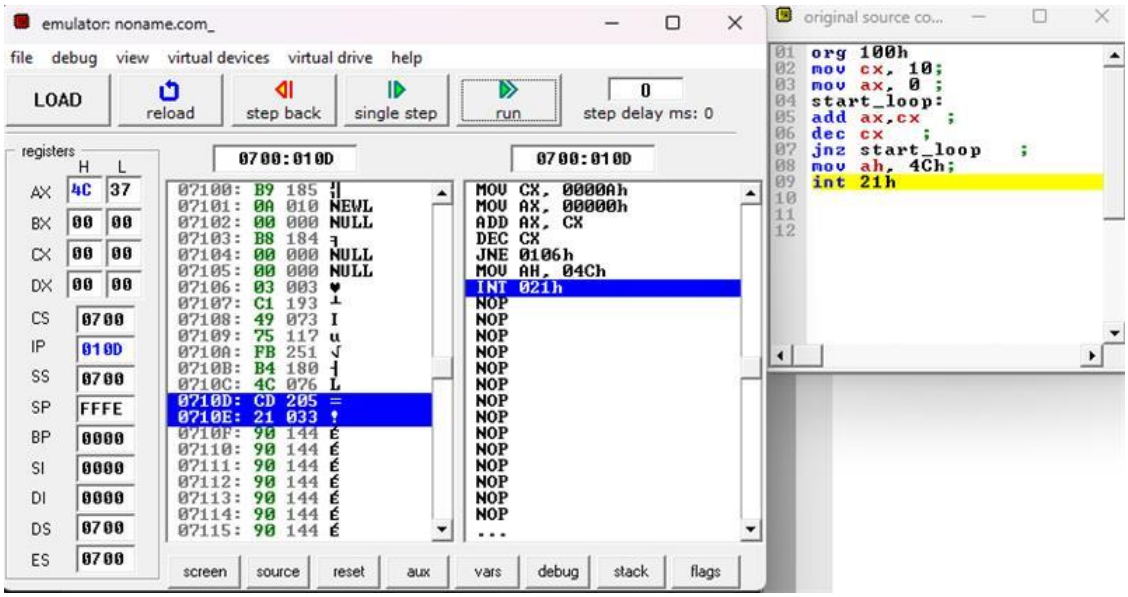


dec cx

Se repite el ciclo 9 veces



mov ah, 4Ch
Acción: Carga el valor 4Ch en el registro AH.
Propósito: Prepara la llamada para finalizar el programa



int 21h
Acción: Llama a la interrupción del sistema DOS.
Propósito: Termina el programa y devuelve el control al sistema operativo

- ¿Qué valor tendrá AX al final del ciclo y por qué?

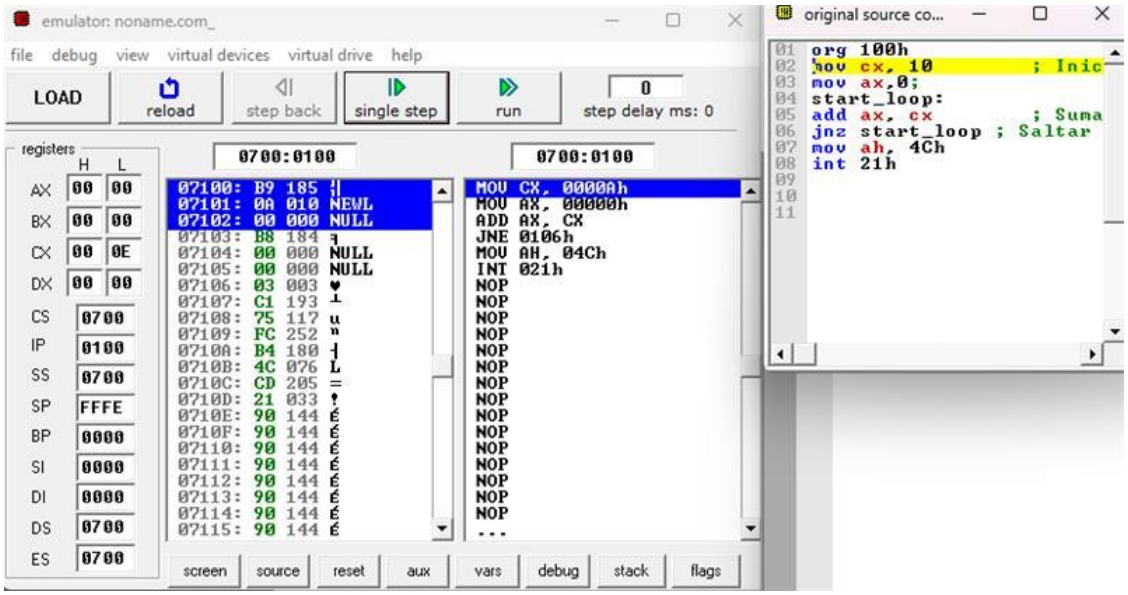
R = El ciclo suma los valores decrecientes de CX a AX, desde 10 hasta 1, realizando la siguiente operación: 10+9+8+...+110 + 9 + 8 + ... + 110+9+8+...+1

Ejercicio 2.

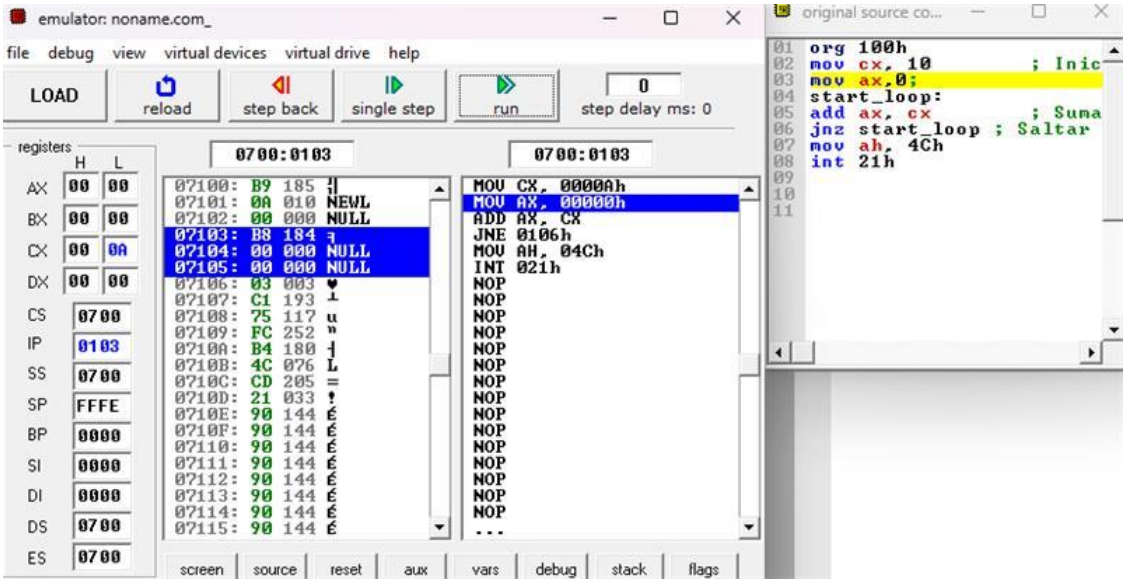
```
org 100h
mov cx, 10          ; Inicializar el contador
mov ax, 0           ; Inicializar el acumulador
start_loop:
    add ax, cx      ; Sumar CX a AX
    dec cx          ; Decrementar CX
    jnz start_loop  ; Saltar a start_loop si CX no es cero
mov ah, 4Ch         ; Terminar el programa
int 21h
```

Preguntas:

- Describir cada acción realiza cada línea de código.

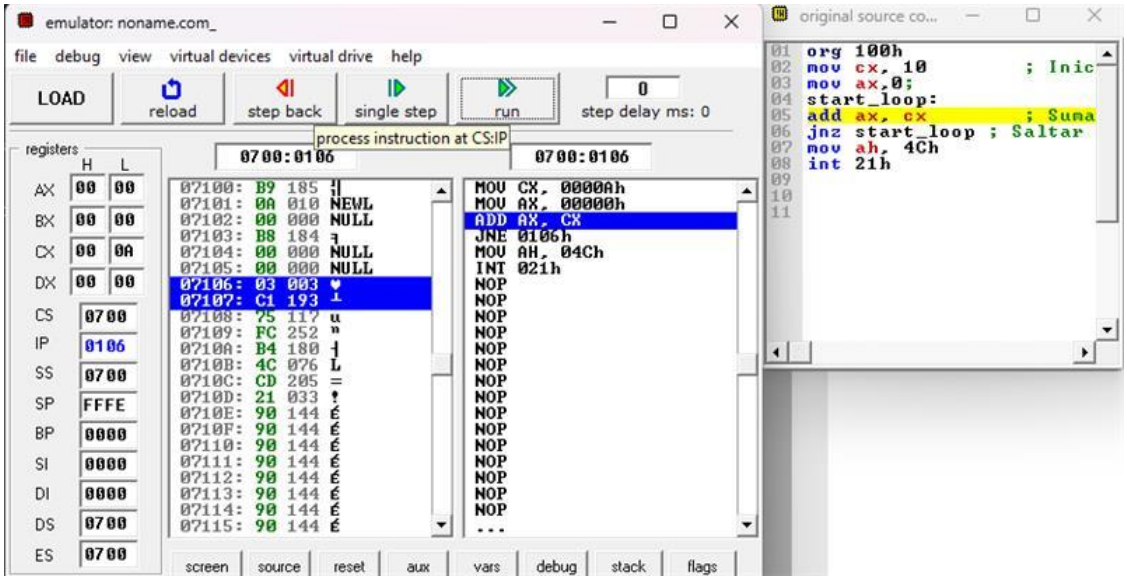


org 100h
Acción: Establece la dirección base del programa en 0x100.
Propósito: En programas COM de DOS, el código comienza en la dirección 0x100 después de la cabecera de 256 bytes.

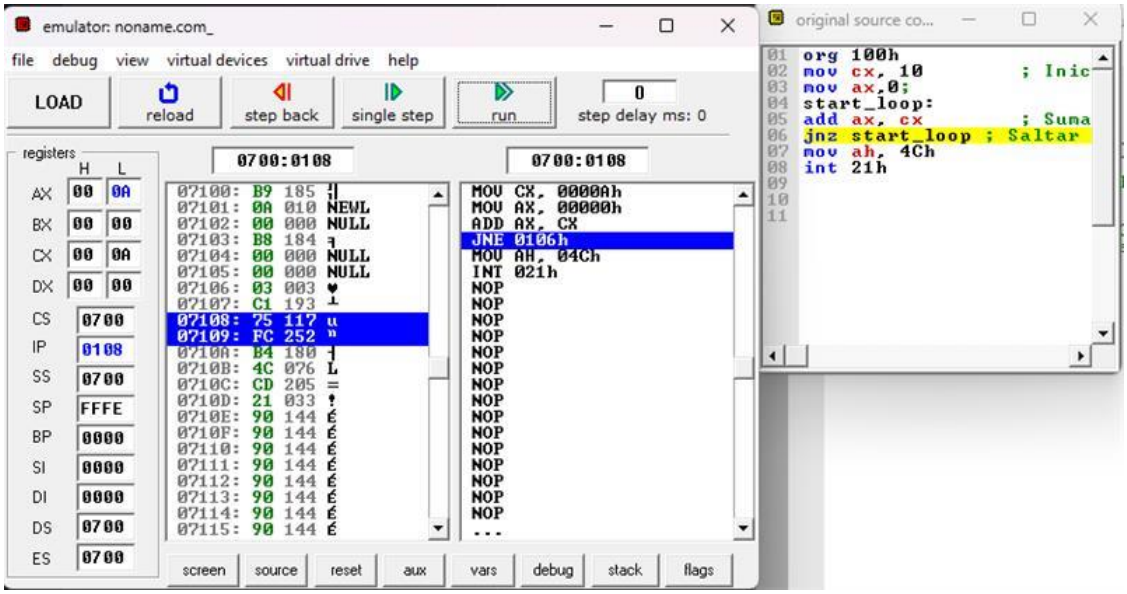


mov cx, 10
Acción: Carga el valor 10 en el registro CX.
Propósito: Inicializa el contador del ciclo.

mov ax, 0
Acción: Asigna el valor 0 al registro AX.
Propósito: Inicializa el acumulador donde se sumarán los valores.

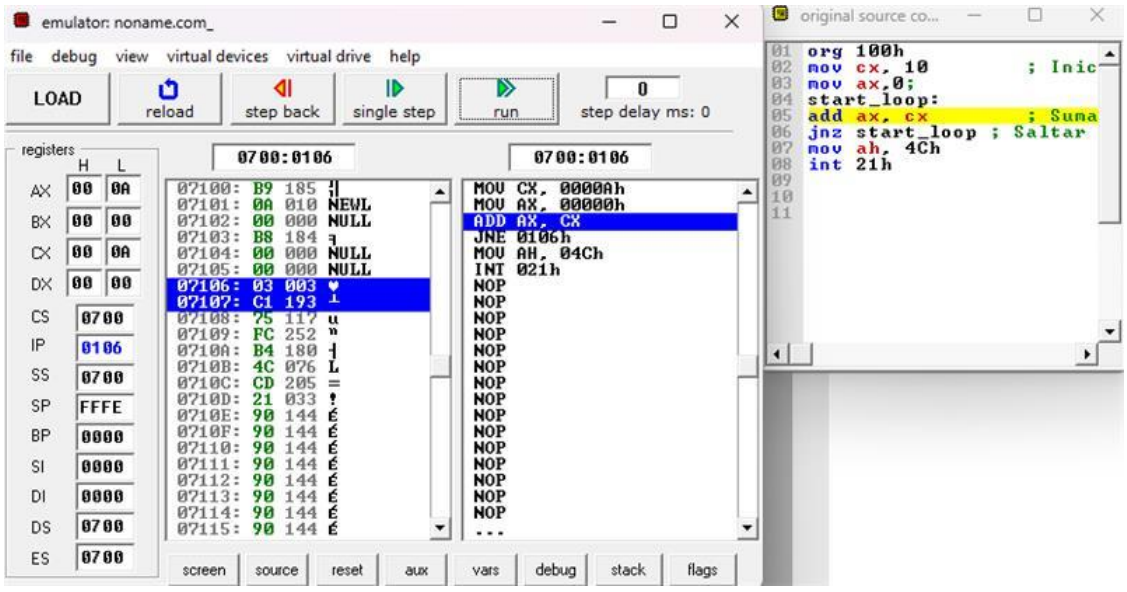


start_loop:
add ax, cx
Acción: Suma el valor actual de CX al registro AX.
Propósito: Acumula la suma progresiva de los valores decrecientes de CX.



dec cx
Acción: Decrementa el valor de CX en 1.
Propósito: Reduce el contador para eventualmente salir del ciclo.

jnz start_loop
Acción: Verifica si CX es diferente de cero (Not Zero).
Si CX ≠ 0, salta a start_loop para repetir el ciclo.
Si CX = 0, el ciclo termina.



mov ah, 4Ch
Acción: Carga el valor 4Ch en el registro AH.
Propósito: Prepara la llamada para finalizar el programa

int 21h
Acción: Llama a la interrupción del sistema DOS.
Propósito: Termina el programa y devuelve el control al sistema operativo.

- ¿Qué instrucción se usa para la multiplicación en ensamblador?

  La instrucción para la multiplicación en ensamblador x86 es: MUL operando

  - MUL realiza una multiplicación de números sin signo.
  - Si multiplicas un registro de 8 bits (como AL) por otro valor de 8 bits, el resultado se almacena en AX (16 bits).
  - Si multiplicas un registro de 16 bits (como AX) por otro valor de 16 bits, el resultado se almacena en DX:AX (32 bits).

- ¿Qué valor tendrá AX al final del ciclo si se multiplica del 5 al 1?

Si quieres multiplicar los valores del 5 al 1, necesitas modificar tu código para hacer multiplicaciones en lugar de sumas. El ciclo multiplicaría:

5×4×3×2×1=1205 \times 4 \times 3 \times 2 \times 1 = 1205×4×3×2×1=120

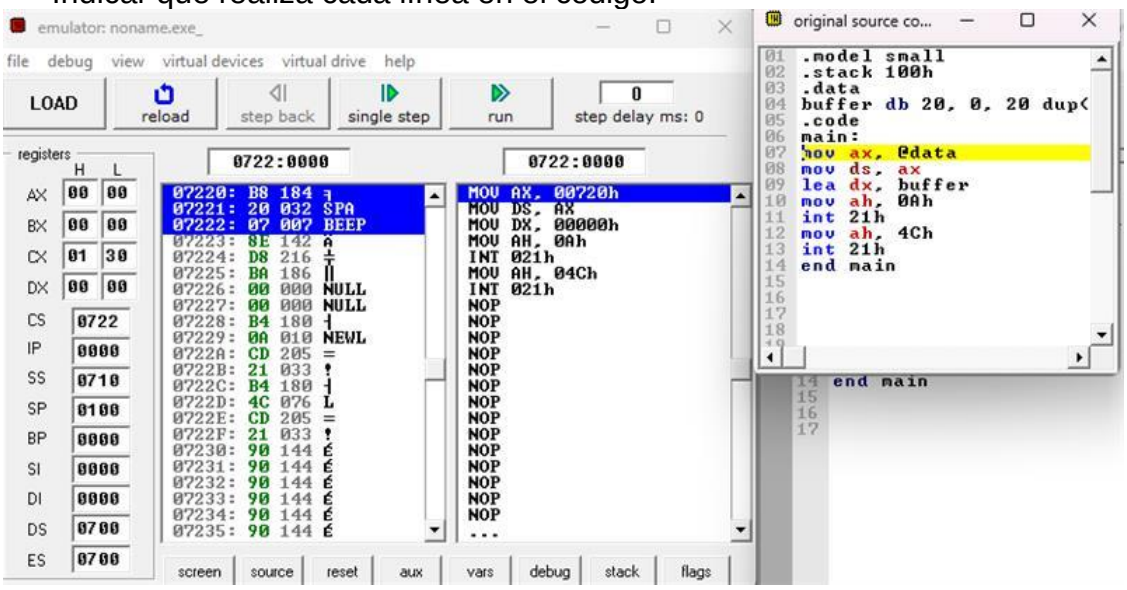
El valor final de AX será **120**.

2 – CAPTURA BÁSICA DE CADENAS

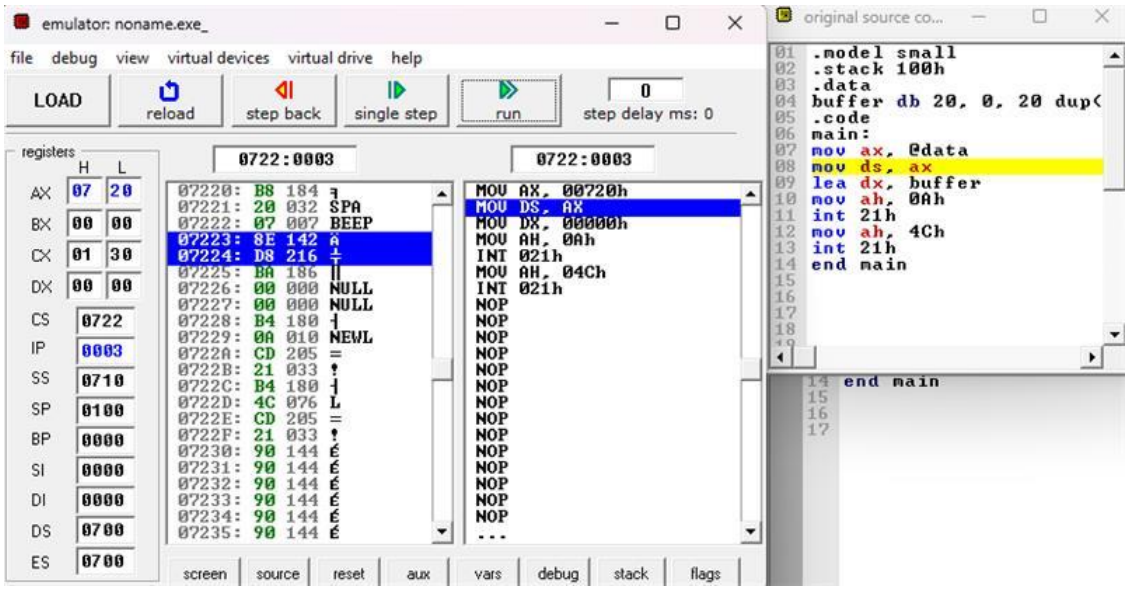
Ejercicio 1: En EMU8086 y DosBOX realizar la captura una cadena de caracteres desde el teclado y la almacena en la memoria.

```
.model small
.stack 100h
.data
   buffer db 20, 0, 20 dup('$')
.code
main:
   mov ax, @data
   mov ds, ax
   lea dx, buffer
   mov ah, 0Ah
   int 21h
   mov ah, 4Ch
   int 21h
end main
```

- Indicar que realiza cada línea en el código.

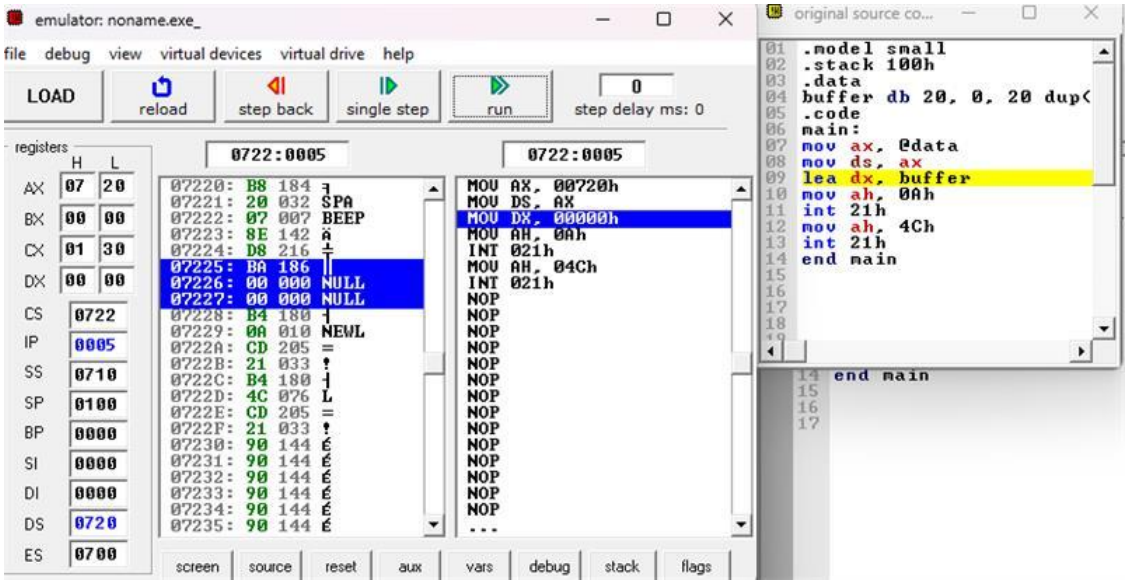


.model small
Acción: Define el modelo de memoria del programa como small.
.stack 100h
Acción: Reserva 256 bytes (100h en hexadecimal) para la pila
.data
buffer db 20, 0, 20 dup('$')
Acción: Declara un buffer para capturar la cadena de caracteres.
.code
main:
Acción: Define la sección de código e indica el punto de entrada (main).

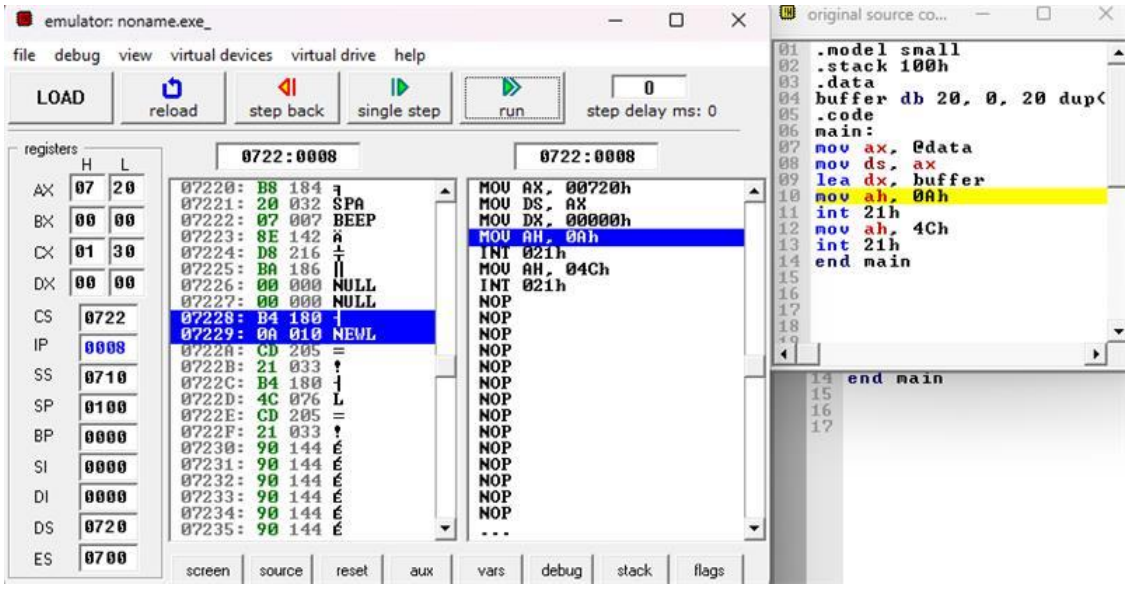


mov ax, @data
Acción: Carga la dirección del segmento de datos (@data) en el registro AX.

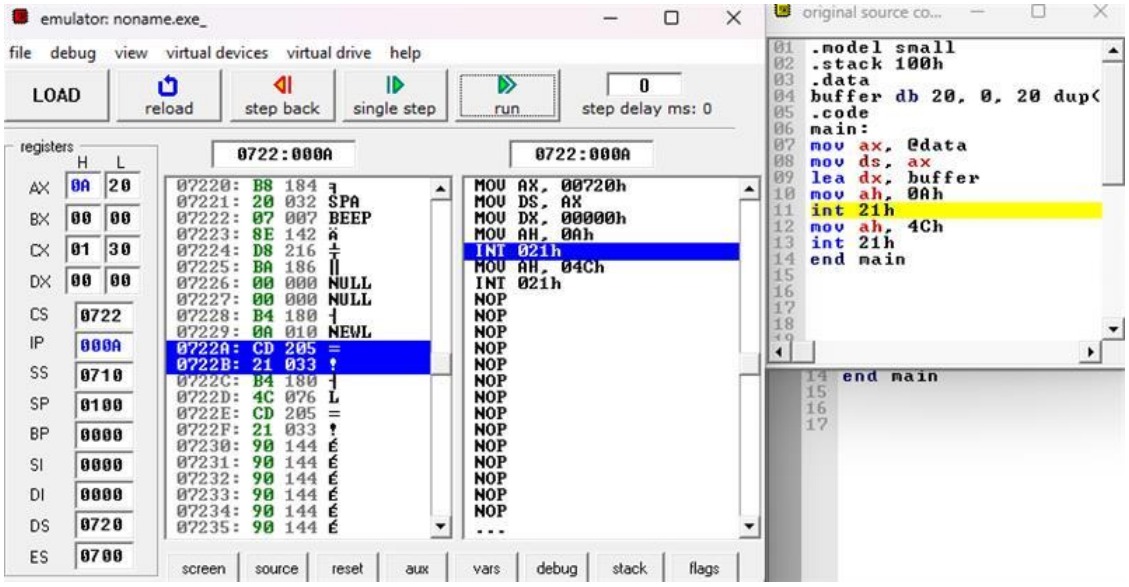
mov ds, ax
Acción: Carga el valor de AX (que contiene la dirección del segmento de datos) en el registro DS



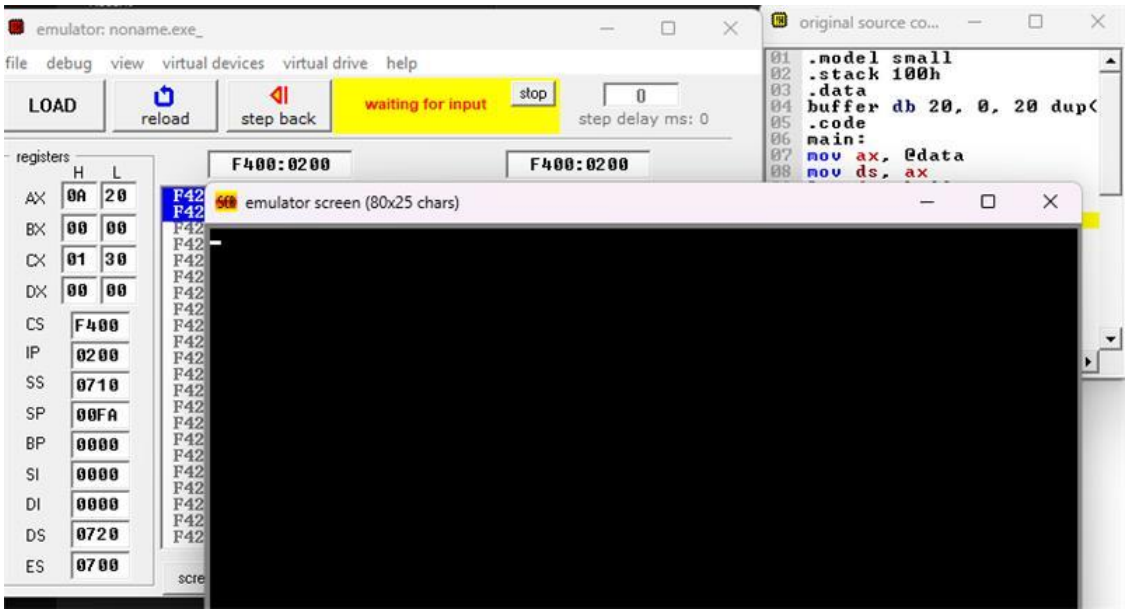
lea dx, buffer
Acción: Carga la dirección del buffer en el registro DX.



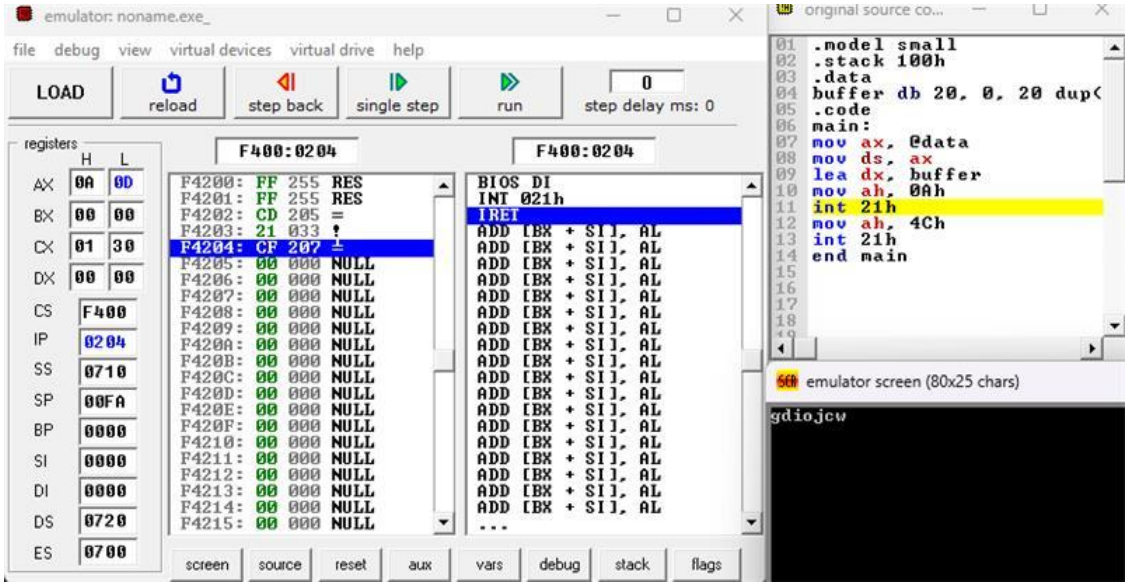
mov ah, 0Ah
Acción: Carga el valor 0Ah en AH.



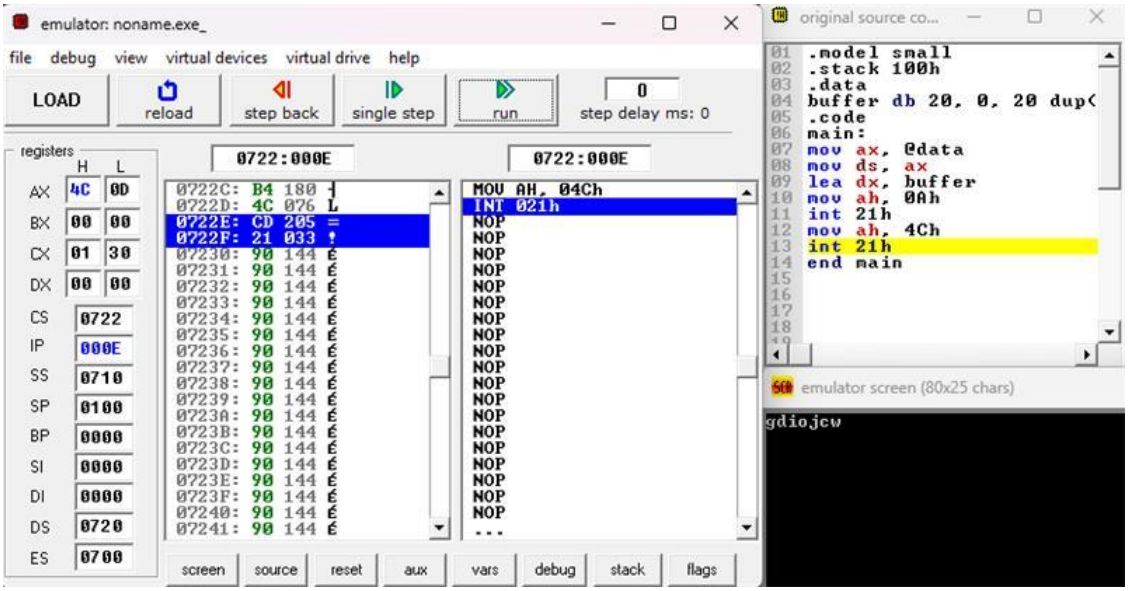
int 21h
Acción: Llama a la interrupción 21h con la función 0Ah activa.



mov ah, 4Ch
Acción: Carga el valor 4Ch en AH.



int 21h
Acción: Llama a la interrupción del sistema DOS



end main
Acción: Indica el final del programa y define el punto de inicio (main)

- ¿Qué función de la interrupción 21h se utiliza para capturar una cadena de caracteres?

La función utilizada es:

assemblyCopiarEditarmov ah, 0Ah
int 21h

- **AH = 0Ah**: Esta función permite capturar una cadena desde la entrada estándar.
- La cadena capturada se almacena en el buffer especificado (DX apunta a él).
- El primer byte del buffer define la longitud máxima permitida.
- El segundo byte almacenará la cantidad de caracteres capturados.
- La cadena capturada finaliza con un carácter nulo (0Dh o Carriage Return).

- ¿Cómo se define el buffer para capturar la cadena?

El buffer se define en la sección .data con la siguiente sintaxis:
assemblyCopiarEditarbuffer db 20, 0, 20 dup('$')
20: Tamaño máximo permitido para la cadena (**20 caracteres**).
0: Inicialmente, no hay caracteres capturados (se actualizará al capturar la cadena).
20 dup('$'): Reserva **20 bytes** adicionales con el símbolo $ como relleno.
El símbolo $ es el terminador de cadena en DOS.
La cadena capturada reemplazará los caracteres $ con los valores ingresados.

Byte 1: Tamaño máximo permitido.
Byte 2: Número real de caracteres capturados.
ytes restantes: Contiene la cadena capturada seguida por el terminador $.

Ejercicio 2: Completar el siguiente código para que**:** en EMU8086 y DosBOX realizar la captura de una cadena y desplegado de la misma.

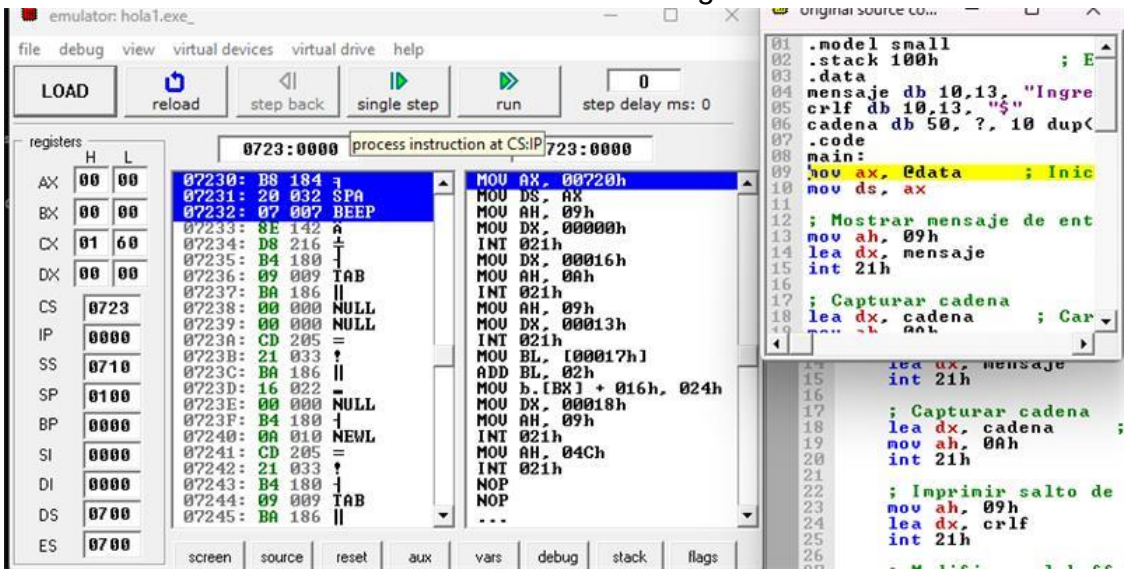
```
.model small
.data
    mensaje db 10,13, "ingresa cadena: $"
    crlf db 10,13, "$"
    cadena db 50,?,10 dup(' ')
.code
mov ax, @data
mov ds, ax
mov ah, 09h
lea dx, mensaje
int 21h
mov dx,
mov ah, 0ah
int 21h
mov ah, 09h
lea dx, crlf
int 21h
mov bl, cadena[1]
mov cadena[bx+2], '$'
mov dx,
mov ah, 09h
int 21h
    .exit
end
```

- ¿Qué elemento hacía falta para completar el código y por qué?

El código original tenía los siguientes problemas:

- Falta del segmento .stack:
    - En MASM, la directiva .stack es necesaria para definir el tamaño de la pila. Sin ella, el programa podría fallar debido a un desbordamiento o acceso incorrecto de memoria.
    - La línea que faltaba:
- Finalización incorrecta:
    - Usar .exit en lugar de mov ah, 4Ch / int 21h no es correcto para un entorno DOS.
    - En DOS, la combinación mov ah, 4Ch e int 21h finaliza el programa de forma segura, devolviendo el control al sistema operativo.
    -
- Líneas incompletas:
    - Algunas líneas tenían instrucciones como mov dx, sin especificar un valor, lo que causaba errores de sintaxis.

- Describir cada acción realiza cada línea de código.



asmCopiarEditar.model small
Define el modelo de memoria pequeño.
small permite un segmento de código y un segmento de datos, suficiente para programas simples de DOS.

asmCopiarEditar.stack 100h
Reserva 256 bytes (100h) para la pila.
Esto es necesario para evitar errores relacionados con el manejo de la pila durante la ejecución.

asmCopiarEditar.data

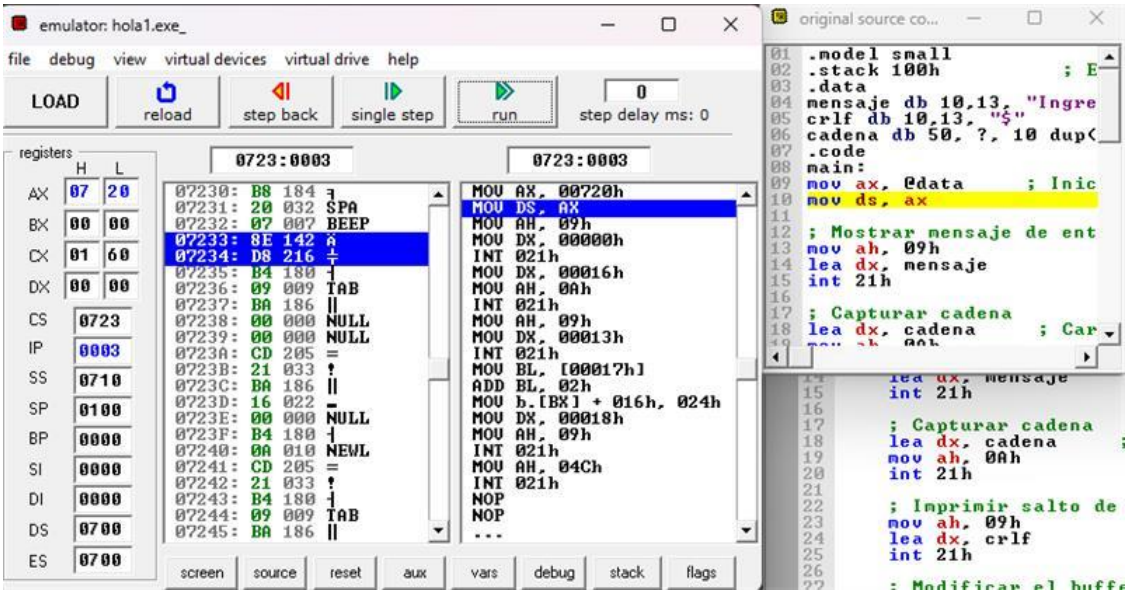
- mensaje db 10,13, "Ingresa cadena: $" ; Mensaje con salto de línea
- crlf db 10,13, "$" ; Salto de línea para impresión
- cadena db 50, ?, 10 dup(' ')

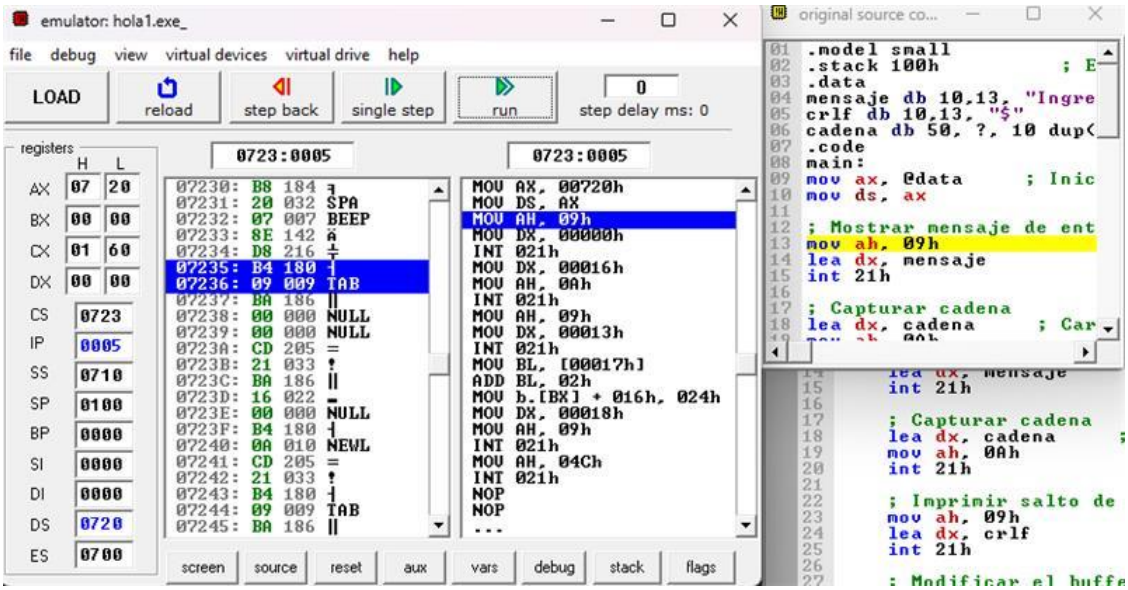
; Buffer para la cadena ingresada
Define los datos del programa:
mensaje: Contiene el texto que solicita la cadena, seguido de un salto de línea (10,13 → LF y CR).

- crlf: Caracteres de salto de línea ($ indica el final para la función 09h).
- cadena:
- 50 → Longitud máxima permitida.
- ? → Byte para la longitud real (el sistema lo llenará).
- 10 dup(' ') → Espacio reservado para la cadena ingresada.

asmCopiarEditar.code
main:
Inicia el segmento de código y el punto de entrada principal del programa.

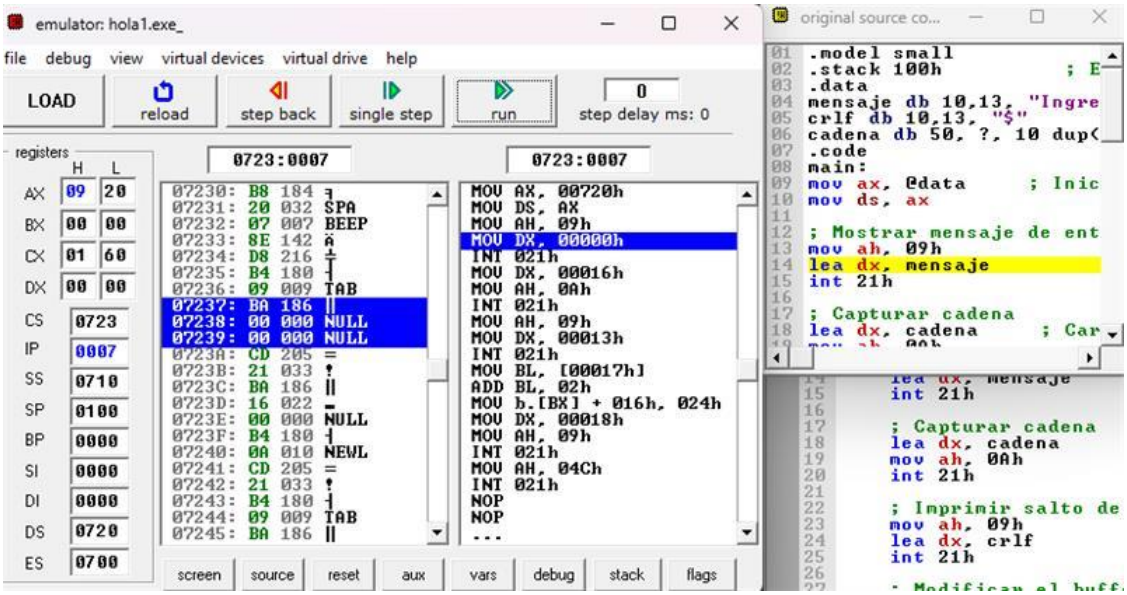
asmCopiarEditarmov ax, @data
mov ds, ax
Carga el segmento de datos (@data) en el registro DS, necesario para acceder a las variables definidas en .data.

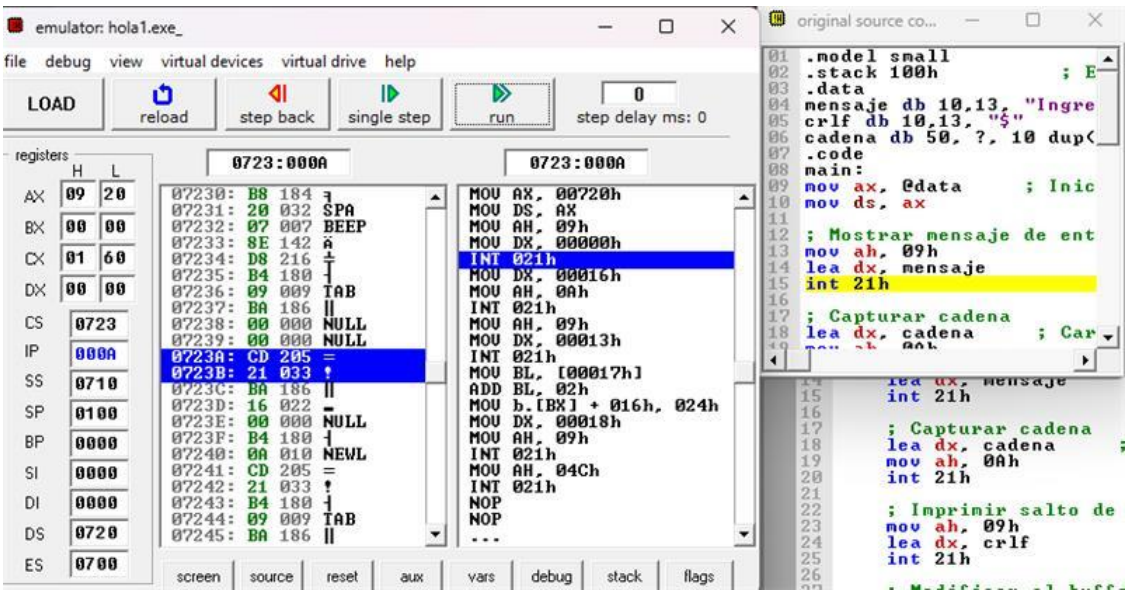




asmCopiarEditarmov ah, 09h
lea dx, mensaje
int 21h

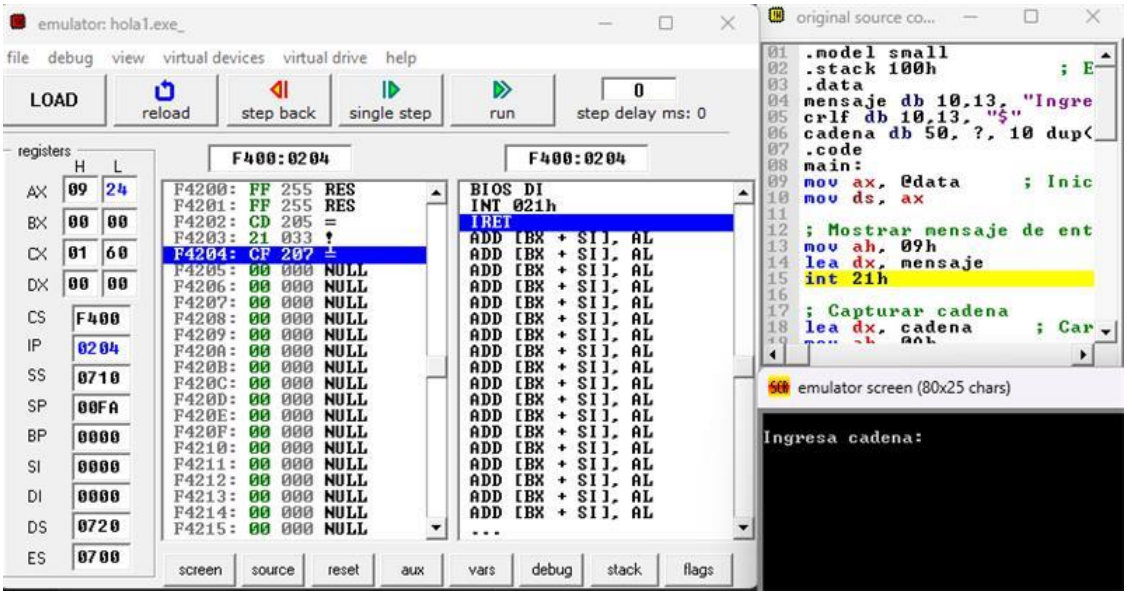
- Muestra el mensaje de entrada:
- mov ah, 09h → Cargar la función DOS para imprimir cadenas.
- lea dx, mensaje → Cargar la dirección del mensaje.
- int 21h → Llama a la interrupción para mostrar el mensaje.





asmCopiarEditarlea dx, cadena
mov ah, 0Ah
int 21h

- Captura la cadena ingresada:
- lea dx, cadena → Cargar la dirección del buffer para capturar la cadena.
- mov ah, 0Ah → Función DOS para capturar cadenas.
- int 21h → Llama a la interrupción para capturar la cadena.



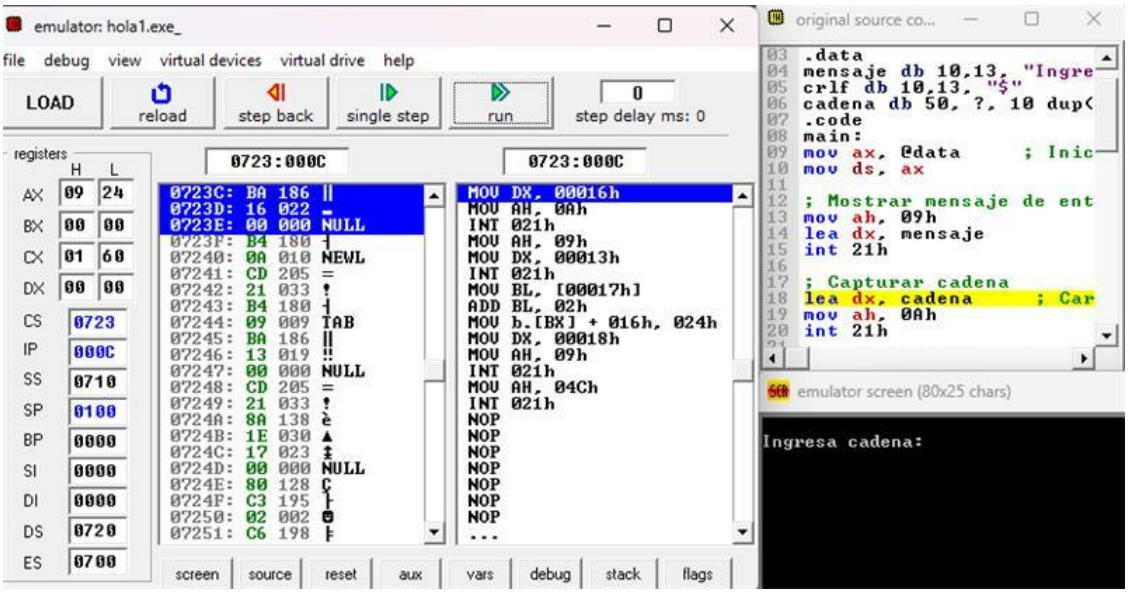
asmCopiarEditarmov ah, 09h

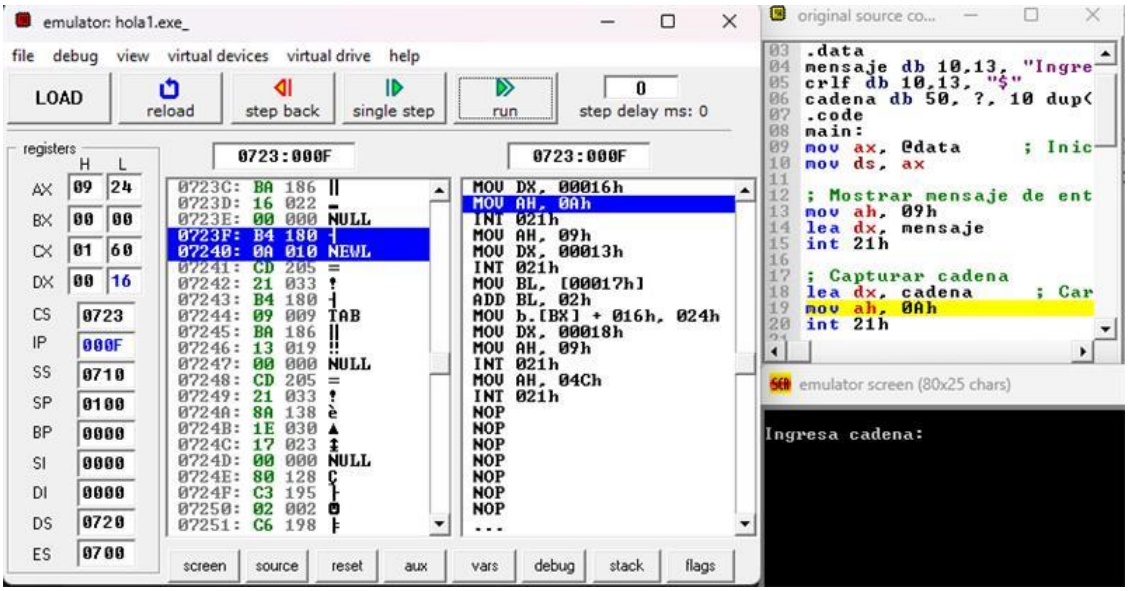
lea dx, crlf

int 21h

Imprime un salto de línea:

- lea dx, crlf → Carga la dirección del buffer que contiene el salto de línea.
- mov ah, 09h → Llama la función para imprimir.
- int 21h → Imprime el salto.

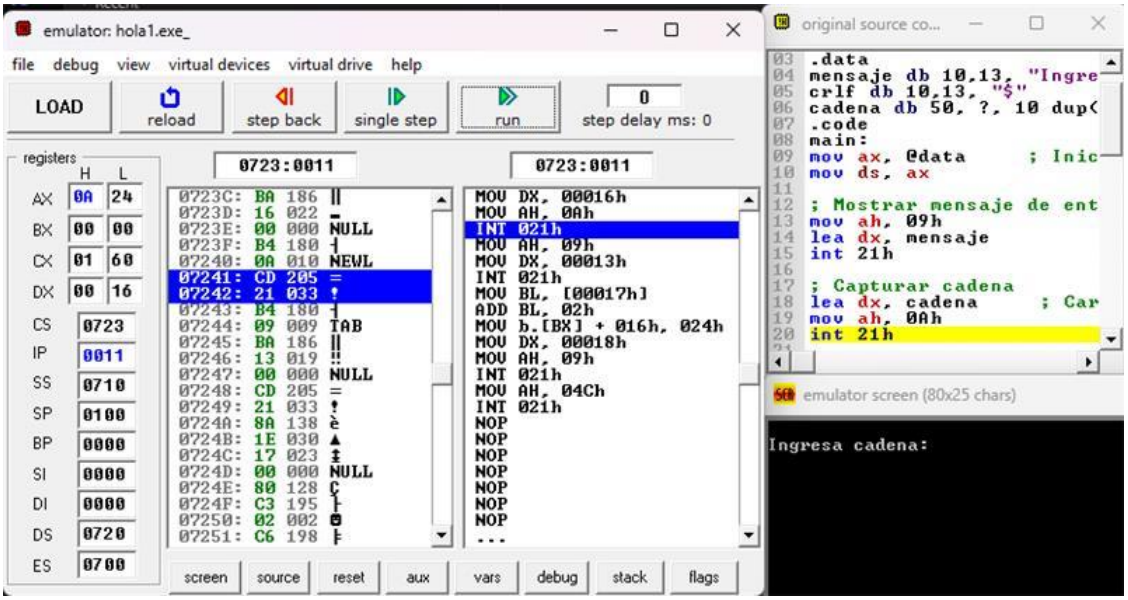


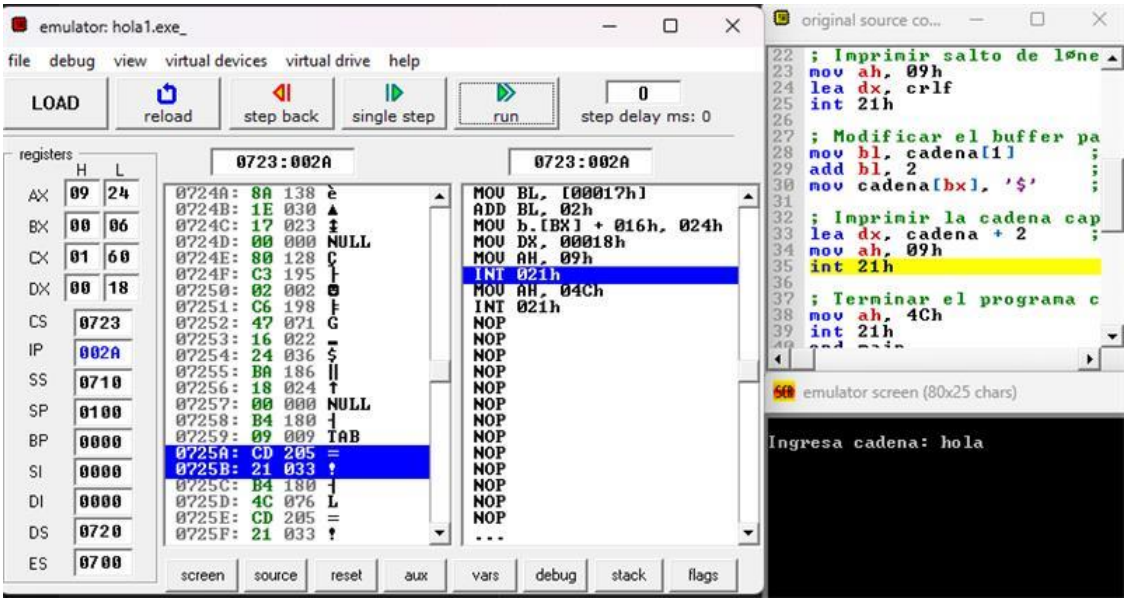


asmCopiarEditarmov bl, cadena[1] ; Obtener la longitud ingresada
add bl, 2 ; Posición después de la cadena
mov cadena[bx], '$' ; Agregar terminador de cadena

Modifica el buffer capturado:

- mov bl, cadena[1]:
- Obtiene la longitud real de la cadena ingresada (almacenada en el segundo byte del buffer).
- add bl, 2:
- Ajusta el índice a la posición correcta.
- Se suma 2 para:
- Saltar la longitud máxima (cadena[0]).
- Saltar la longitud real (cadena[1]).
- mov cadena[bx], '$':
- Agrega el terminador $ al final de la cadena para que se imprima correctamente con 09h.





asmCopiarEditarlea dx, cadena + 2
mov ah, 09h
int 21h

Imprime la cadena capturada:

- lea dx, cadena + 2 → Apunta a la posición real de la cadena (después de los bytes de longitud).
- mov ah, 09h → Llama a la función para imprimir.
- int 21h → Imprime la cadena ingresada.

asmCopiarEditarmov ah, 4Ch
int 21h

Finaliza el programa:

- mov ah, 4Ch → Función DOS para terminar el programa.
- int 21h → Regresa el control al sistema operativo.

- ¿Por qué es necesario el uso de incrementos en el índice en: *mov cadena[bx+2]*?

La razón por la que debes incrementar el índice con +2 es porque:

- El buffer de la cadena tiene un formato específico:
  - El primer byte (cadena[0]) → Almacena la longitud máxima del buffer (50 en este caso).
  - El segundo byte (cadena[1]) → Almacena la longitud real ingresada por el usuario.
  - A partir del tercer byte (cadena[2]) → Comienza la cadena real.

- Por qué sumas 2:
  - cadena[bx] → Accede a la posición donde agregarás el terminador $.
  - bx contiene la longitud real (cadena[1]).
  - Para acceder a la posición final de la cadena, debes sumar 2:
    - +1 para saltar la longitud máxima.
    - +1 para saltar la longitud real.
  - Ejemplo: Si ingresas hola (4 caracteres), cadena[1] vale 4, y la posición final será: