Arellano Granados Angel Mariano

218123444

Seminario de Traductores de Lenguajes I

I7026 D02

Reporte de Actividades 6

Actividad 6 – Parte 1 – 4

Actividad 6 – Parte 1

Descripción

1. Investiga las siguientes funciones o servicios.
    1. Función 4C00 de la int 21H
    2. Función 09H de la int 21H
    3. Función 0AH de la int 10H
    4. Función 02H de la int 10H
    5. Función 06H de la int 10H
2. Explique lo que sucede en el siguiente segmento.

```
.DATA
ENTRADA LABEL BYTE
LONMAX db 20
LONREAL db ?
INTROD db 21 DUP (' ')
MEN db 'INDRODUCRIR NOMBRE: ','$'
```

3. Documente cada línea del procedimiento principal.
4. Documente el resto del programa y explique las siguientes líneas.
    a. *DESPL0 PROC NEAR*
    b. *TECLA0 PROC NEAR*
    c. *CAMPA PROC NEAR* (debería ser un sonido)
    d. *CENTRAR PROC NEAR*
    e. *PANT0 PROC NEAR*
    f. *CURS0 PROC NEAR*
5. Ejecute y reporte lo que sucede.

Desarrollo y Resultados

1. Investiga las siguientes funciones o servicios

a) Int 21/AH=4Ch

TERMINATE WITH RETURN CODE

AH = 4Ch

AL = return code

b) Int 21/AH=09h

WRITE STRING TO STANDARD OUTPUT

AH = 09h

DS: DX -> '$'-terminated string

Return:

AL = 24h (the '$' terminating the string, despite official docs which

state that nothing is returned)

c) Int 10/AH=0Ah

WRITE CHARACTER ONLY AT CURSOR POSITION

AL = character to display

BH = page number

BL = attribute or color

CX = number of times to write character

d) Int 10/AH=02h

SET CURSOR POSITION

AH = 02h

BH = page number

DH = row

DL = column

e) Int 10/AH=06h

SCROLL UP WINDOW

AH = 06h

AL = number of lines by which to scroll up

BH = attribute used to write blank lines at bottom of window

CH, CL = row, column of window's upper left corner

DH, DL = row, column of window's lower right corner

2. Explique lo que sucede en el siguiente segmento

Se declara el segmento de datos, dentro se declara una etiqueta vacía llamada ENTRADA de tipo BYTE y cuatro variables llamadas LONMAX con el valor de 20d, LONREAL sin valor específico, INTROD que duplica 21 veces el carácter ‘ ‘ y MEN que contiene la cadena 'INDRODUCRIR NOMBRE: ','$'.

3. Documente cada línea del procedimiento principal

BEGIN PROC FAR

Declara el procedimiento BEGIN de tipo FAR.

MOV AX, @DATA

Mueve la dirección de memoria del segmento de datos a AX.

MOV DS, AX

Mueve dirección de memoria del segmento de datos de AX a DS.

OTRO: CALL PANT0

Inicia una etiqueta llamada OTRO que llama al procedimiento PANT0.

MOV DX,0502H

Mueve el número 0502H al registro DX.

CALL CURS0

Llama al procedimiento CURS0.

CALL DESPL0

Llama al procedimiento DESPL0.

CALL TECLA0

Llama al procedimiento TECLA0.

CMP LONREAL,00

Compara LONREAL con el número 00 para saber si no está vacía.

JE SALIR

Si en la comparación anterior los números coinciden salta a SALIR, sino continua con las demás instrucciones.

CALL CAMPA

Llama al procedimiento CAMPA.

CALL CENTRAR

Llama al procedimiento CENTRAR.

JMP OTRO

Regresa a la etiqueta OTRO al inicio del procedimiento ciclando el programa, donde solo se podrá terminar si ingresamos una cadena vacía.

SALIR: MOV AX,4C00H

Si se ingresa una cadena vacía saltara aquí donde coloca 4C00H en AX.

INT 21

Con la interrupción 21H termina el programa con un mensaje.

BEGIN ENDP

Declara el final del procedimiento BEGIN.

4. Documente el resto del programa y explique las siguientes líneas

DESPL0 PROC NEAR

Muestra la cadena MEN en la consola en la posición del cursor.

TECLA0 PROC NEAR

Solicita por consola una cadena y la almacena en INTROD y su longitud en LONREAL.

CAMPA PROC NEAR

Coloca un carácter '$' al final de INTROD y debería sonar un sonido.

CENTRAR PROC NEAR

Calcula la posición del cursor para mostrar la cadena en el centro de la consola, vuelve a cambiar la posición del cursor con los nuevos datos e imprime la cadena en el centro.

PANT0 PROC NEAR

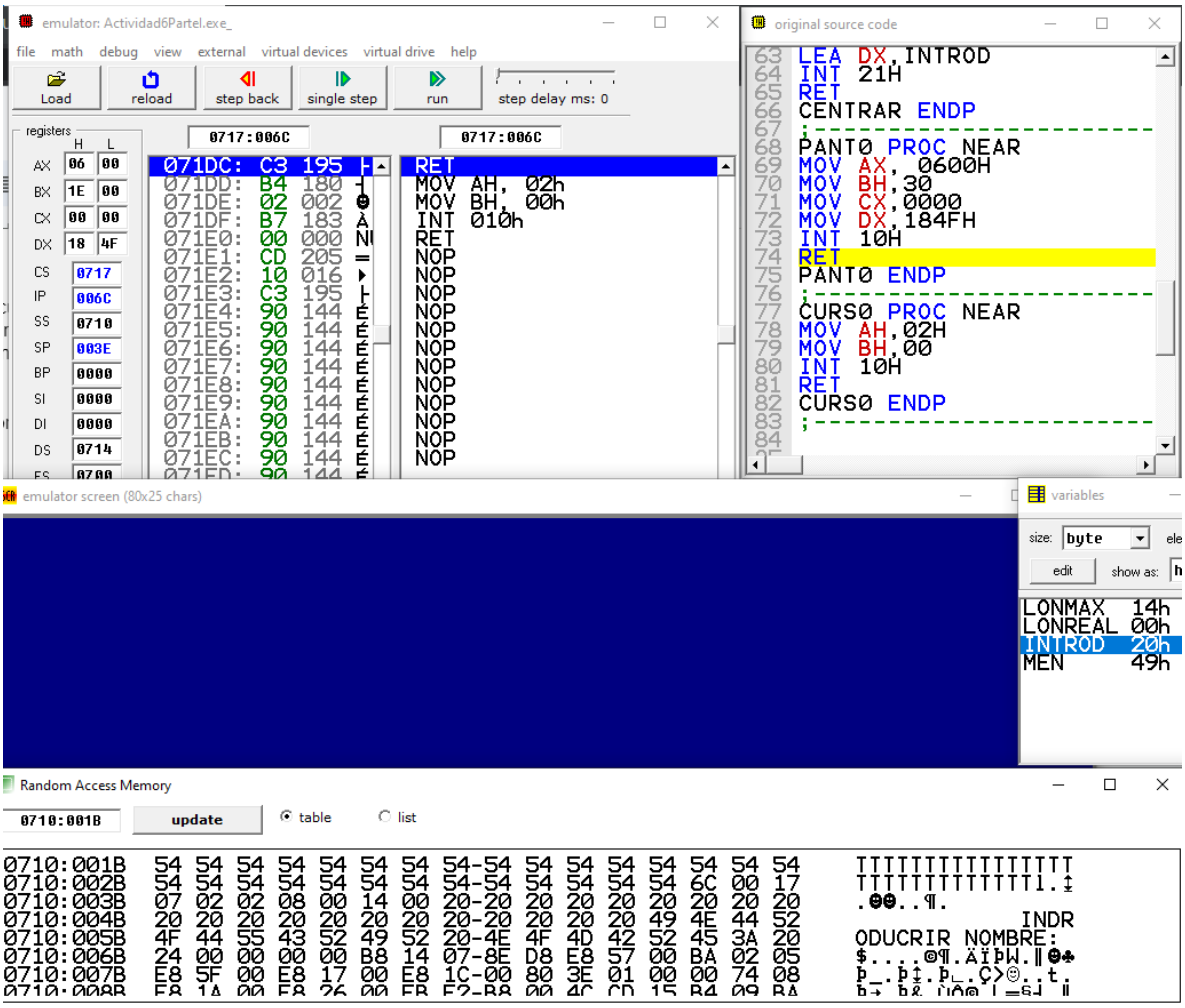
Cambia el scroll de la consola para poner algunas líneas en blanco.

CURS0 PROC NEAR

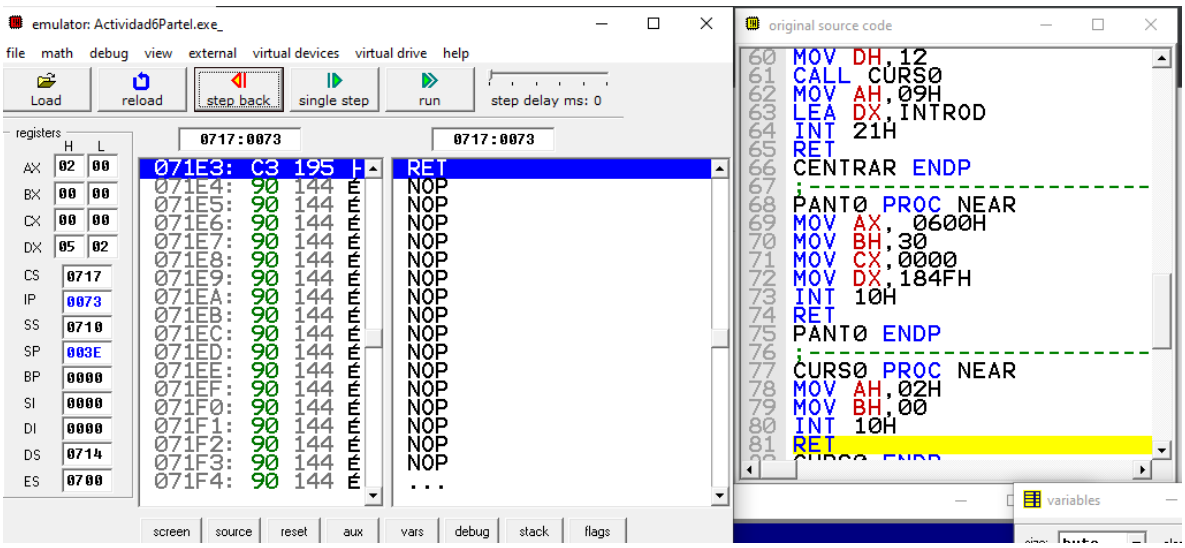
Cambia la posición del cursor en la consola, se usa dos veces en el programa una para colocar el cursor al inicio de la consola y otro para colocarlo al centro de la consola.

5. Ejecute y reporte lo que sucede

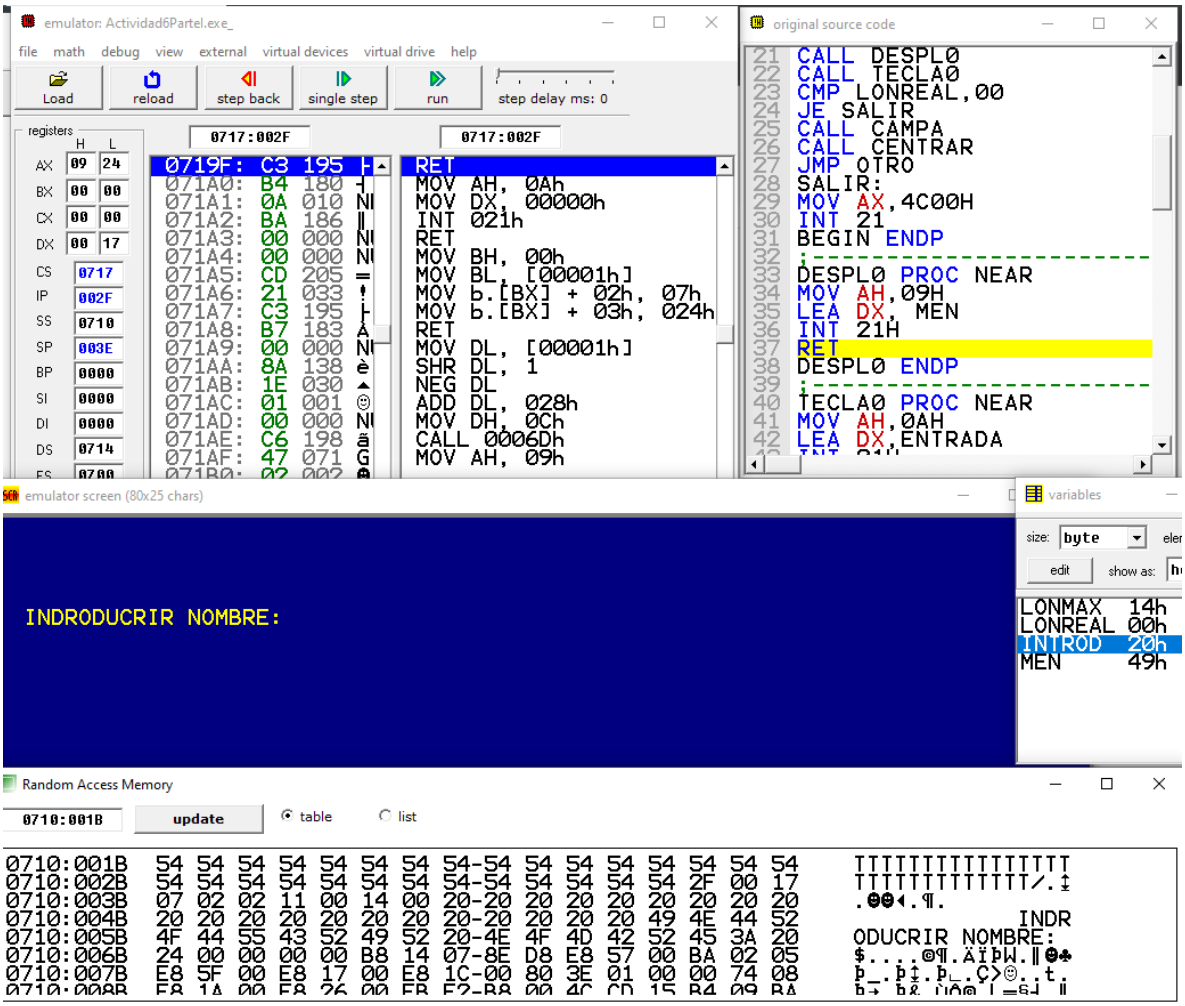
Inicia el segmento de datos y llama a PANT0:



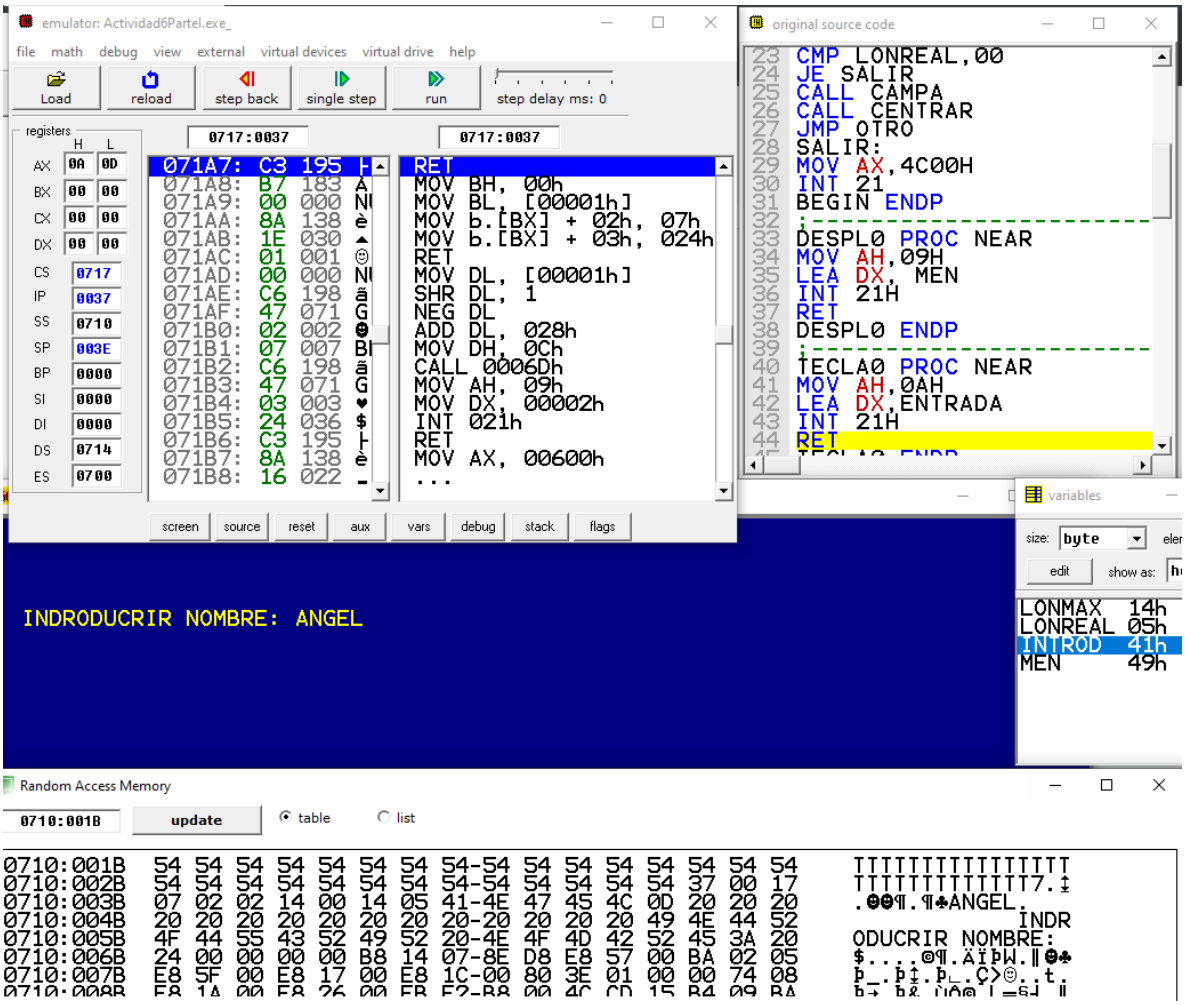
Después llama a CURS0:



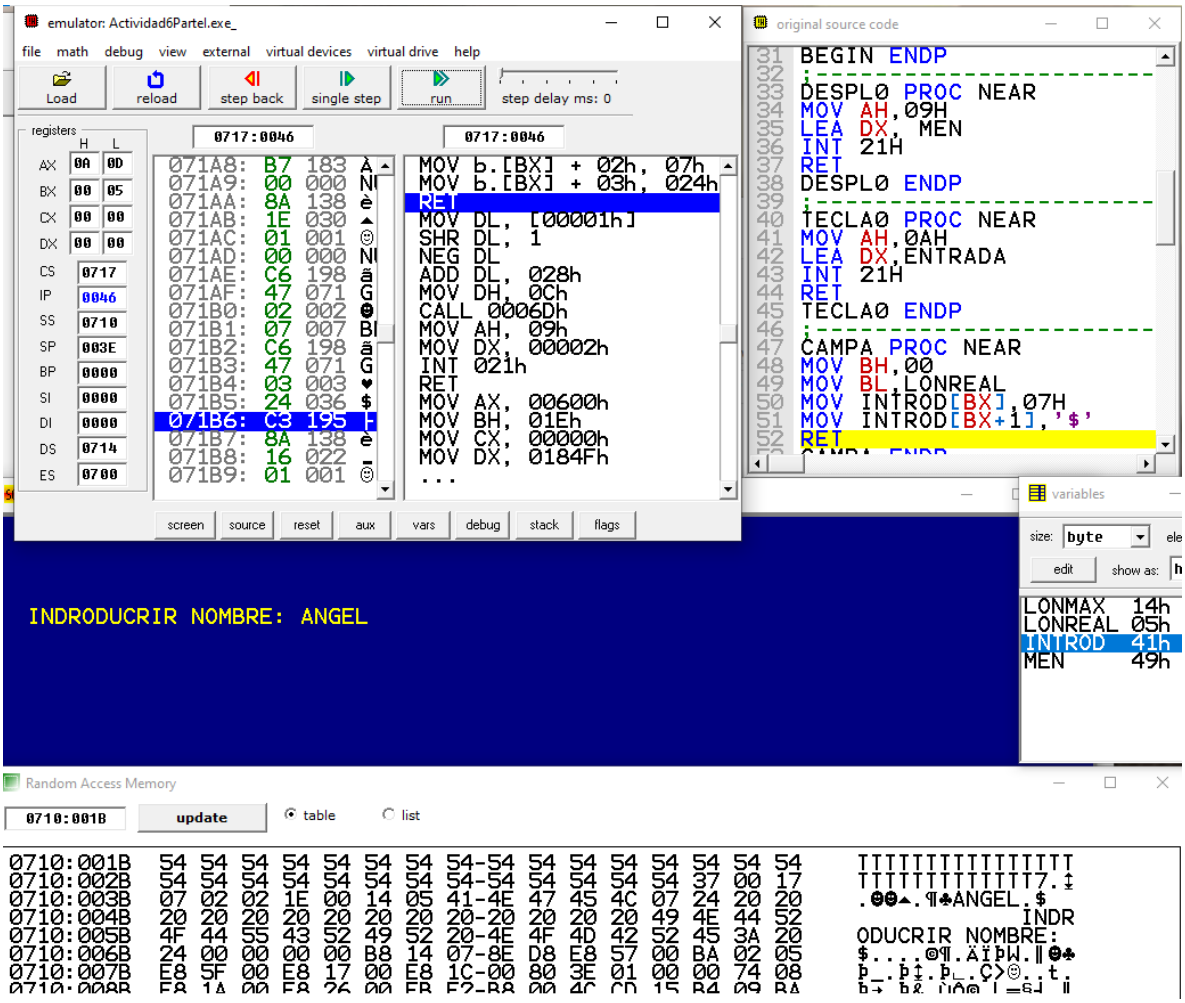
Después llama a DESPL0:



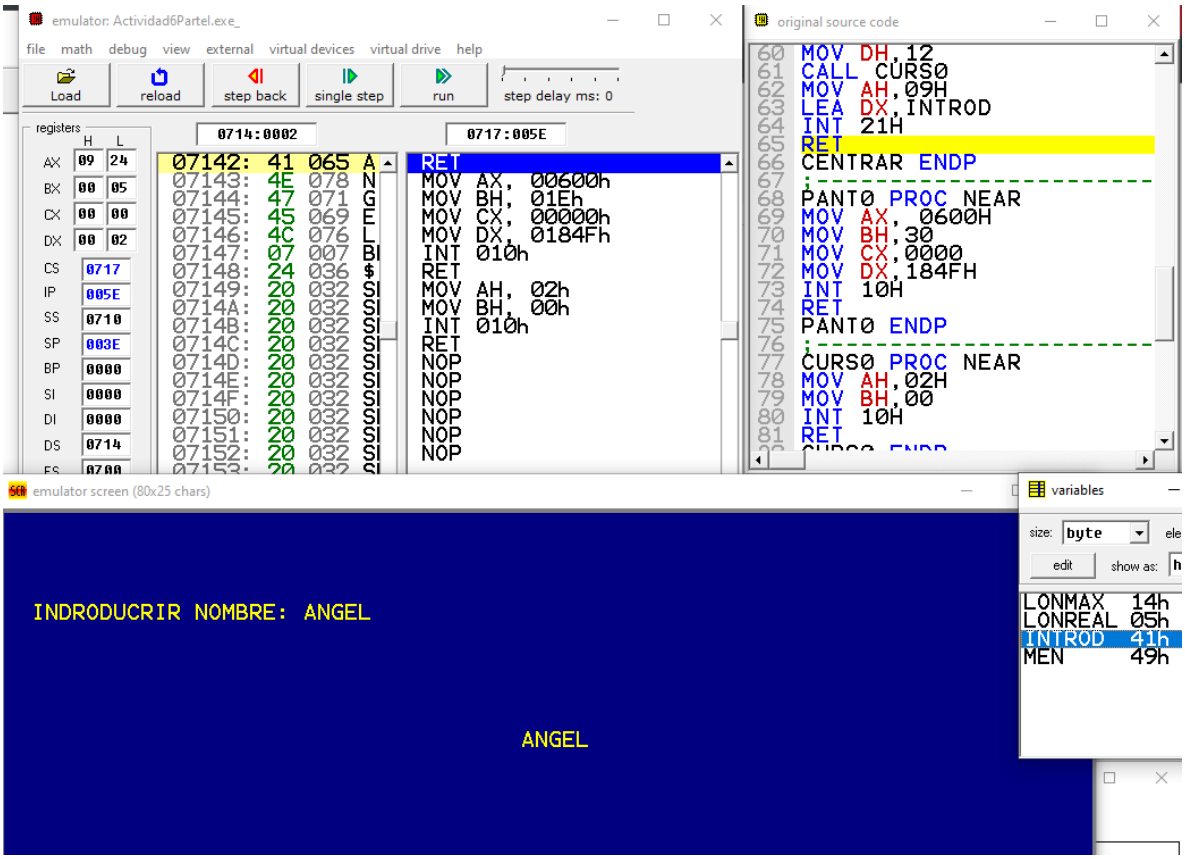
Después llama a TECLA0 se aprecia que cambian las variables y la memoria:



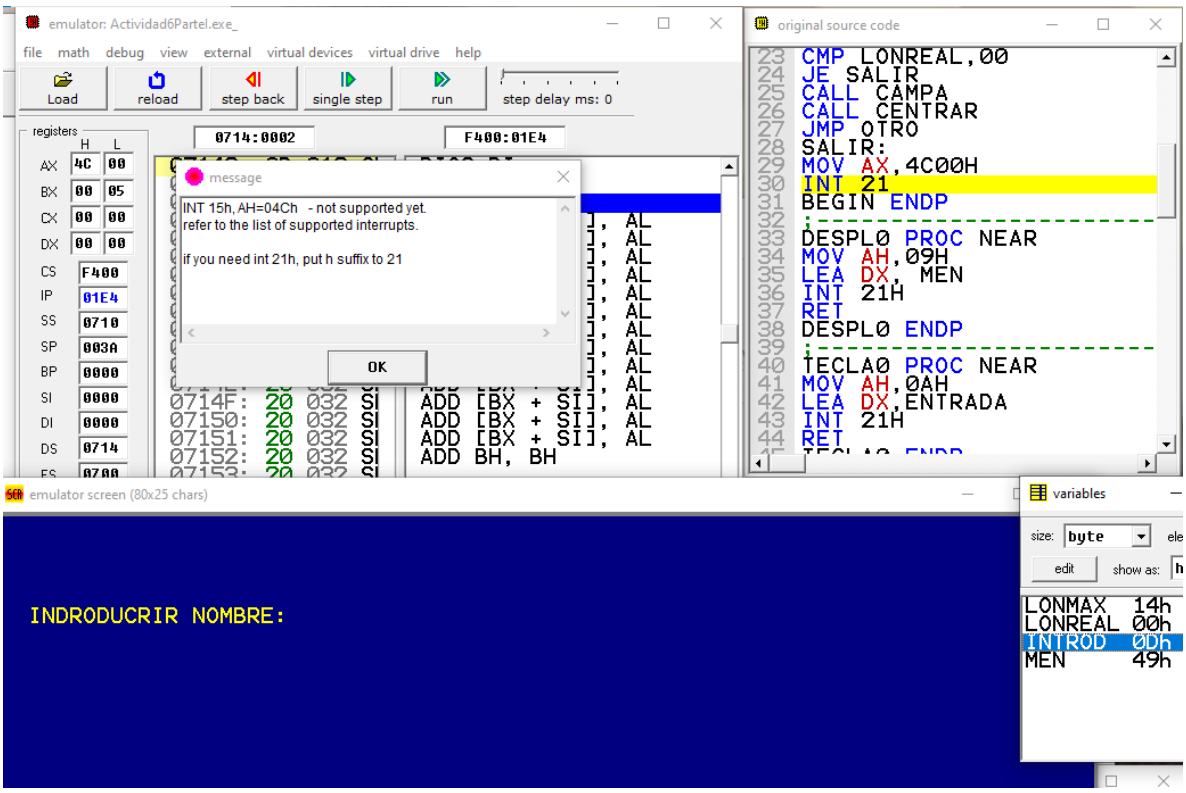
Al no ingresar una cadena vacía llama a CAMPA que cambia la memoria:



Por último, llama a CENTRAR que muestra la cadena al centro:



El programa se cicla, pero si se ingresa la cadena vacía termina:



Código:

```
PAGE 60,120
TITLE PROG8.EXE
.model small
.stack 64
;---------------------------------------
.DATA
ENTRADA LABEL BYTE
LONMAX db 20
LONREAL db ?
INTROD db 21 DUP (' ')
MEN db 'INDRODUCRIR NOMBRE: ','$'
;---------------------------------------
.CODE
BEGIN PROC FAR
    MOV AX,@DATA
    MOV DS,AX
    OTRO:
        CALL PANT0
        MOV DX,0502H
        CALL CURS0
        CALL DESPL0
        CALL TECLA0
        CMP LONREAL,00
        JE SALIR
        CALL CAMPA
        CALL CENTRAR
        JMP OTRO
    SALIR:
        MOV AX,4C00H
        INT 21
    BEGIN ENDP
;---------------------------------------
DESPL0 PROC NEAR
    MOV AH,09H
    LEA DX, MEN
    INT 21H
    RET
    DESPL0 ENDP
;---------------------------------------
TECLA0 PROC NEAR
    MOV AH,0AH
    LEA DX,ENTRADA
    INT 21H
    RET
    TECLA0 ENDP
;---------------------------------------
CAMPA PROC NEAR
     MOV BH,00
     MOV BL,LONREAL
     MOV INTROD[BX],07H
     MOV INTROD[BX+1],'$'
     RET
     CAMPA ENDP
;---------------------------------------
```

```
;---------------------------------------
CENTRAR PROC NEAR
     MOV DL,LONREAL
     SHR DL,1  S
     NEG DL
     ADD DL,40
     MOV DH,12
     CALL CURS0
     MOV AH,09H
     LEA DX,INTROD
     INT 21H
     RET
     CENTRAR ENDP
;---------------------------------------
PANT0 PROC NEAR
     MOV AX, 0600H
     MOV BH,30
     MOV CX,0000
     MOV DX,184FH
     INT 10H
     RET
     PANT0 ENDP
;---------------------------------------
CURS0 PROC NEAR
     MOV AH,02H
     MOV BH,00
     INT 10H
     RET
     CURS0 ENDP
;---------------------------------------
```

Reflexión

En esta aprendí el como personalizar la consola de salida para que sea mas agradable a la vista o dar formatos especiales a las cadenas que imprimíamos.

Actividad 6 – Parte 2

Descripción

Realice un programa que pide tres datos y tras pedir los tres datos entonces los muestre, e.g., código, nombre, carrera.

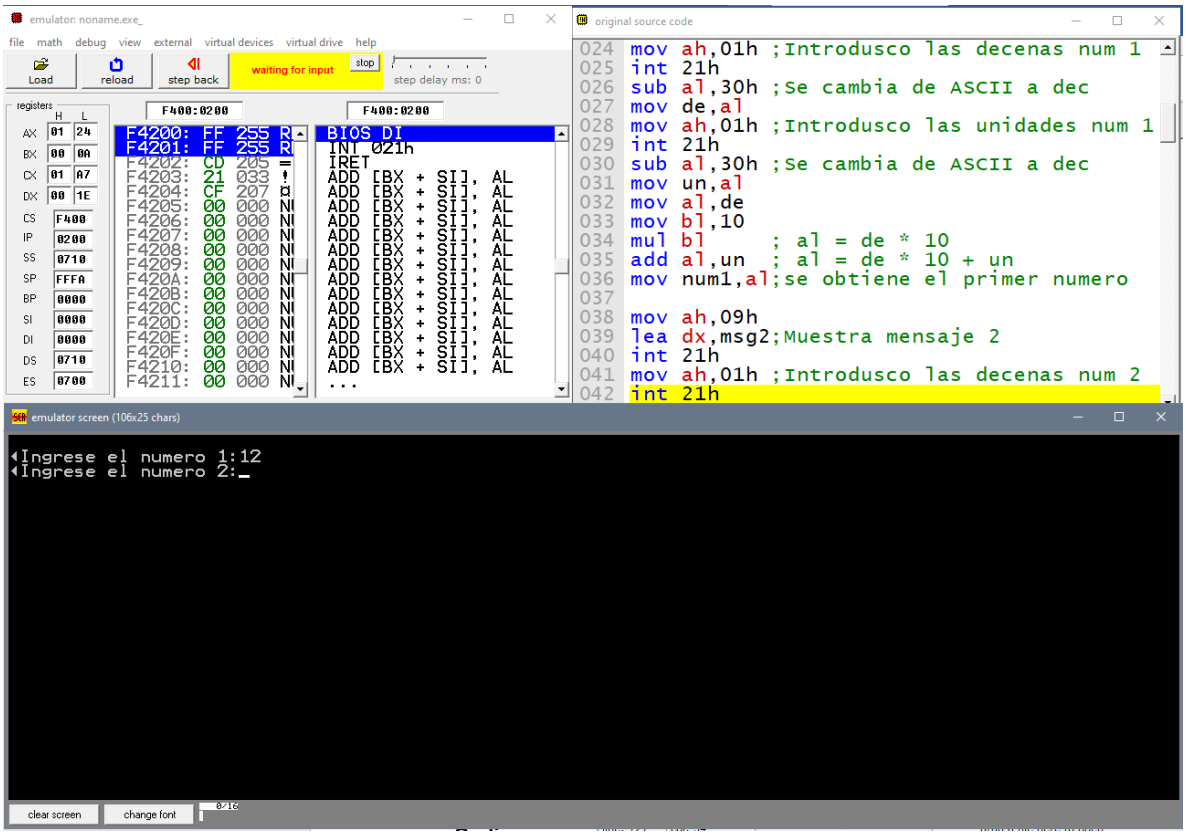
Desarrollo y Resultados

Planteamiento:

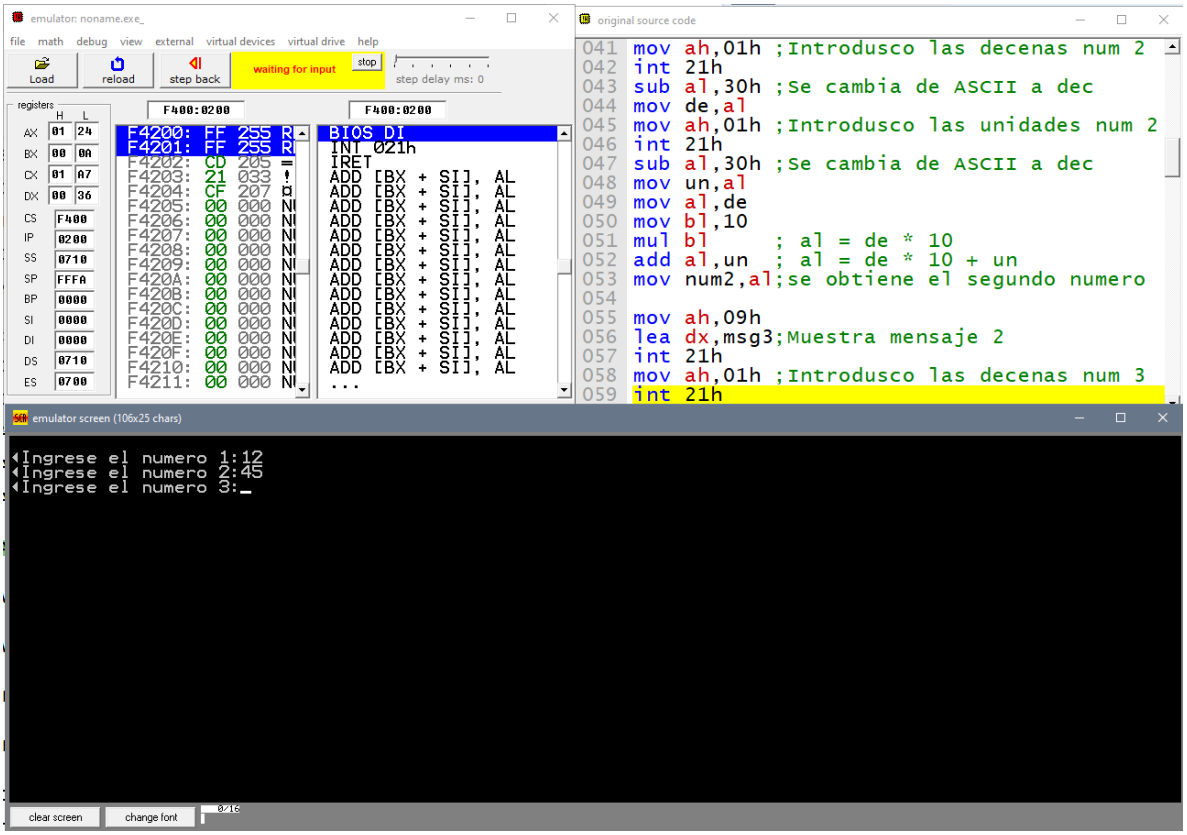
Para esta actividad se realizará un programa que reciba tres números de dos cifras por consola, y los mostrará al final en tres cifras utilizando las restas y sumas para cambiar los números al ASCII y almacenarlos para después usar la función amm para descomponer los números y mostrarlos digito a digito.

Ejecución:

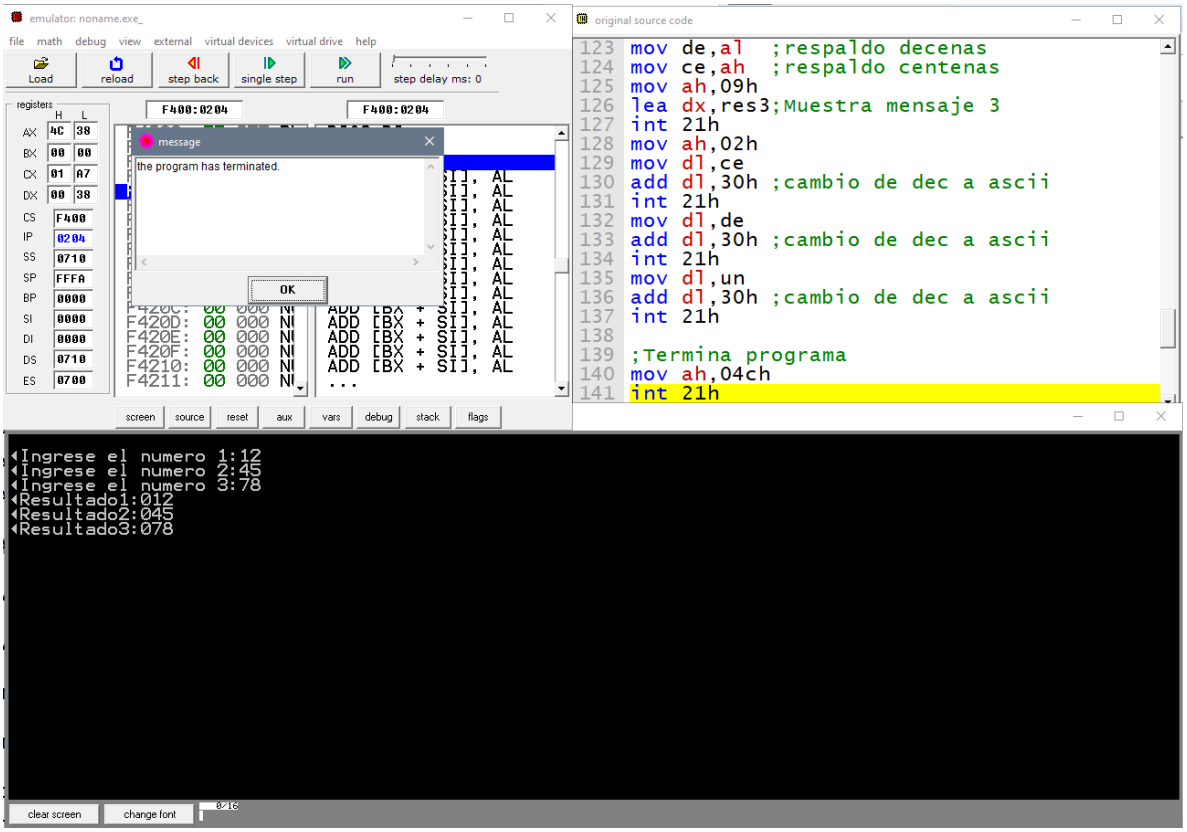
Recibe el primer número:



Recibe el segundo número:



Recibe el tercer número e inmediatamente muestra los 3 números recibidos:



Código:

```
.model .stack
.data
un db 0
de db 0
ce db 0
num1 db 0
num2 db 0
num3 db 0
msg1 db 10,13,17,'Ingrese el numero 1:','$'
msg2 db 10,13,17,'Ingrese el numero 2:','$'
msg3 db 10,13,17,'Ingrese el numero 3:','$'
res1 db 10,13,17,'Resultado1:','$'
res2 db 10,13,17,'Resultado2:','$'
res3 db 10,13,17,'Resultado3:','$'
.code
;obtencion de datos
mov ax,data
mov ds,ax  ;Se inicia en segmento de datos
mov ah,09h
lea dx,msg1;Muestra mensaje 1
int 21h
mov ah,01h ;Introdusco las decenas num 1
int 21h
sub al,30h ;Se cambia de ASCII a dec
mov de,al
mov ah,01h ;Introdusco las unidades num 1
int 21h
sub al,30h ;Se cambia de ASCII a dec
mov un,al
```

```
mov al,de
mov bl,10
mul bl     ; al = de * 10
add al,un  ; al = de * 10 + un
mov num1,al;se obtiene el primer numero
mov ah,09h
lea dx,msg2;Muestra mensaje 2
int 21h
mov ah,01h ;Introdusco las decenas num 2
int 21h
sub al,30h ;Se cambia de ASCII a dec
mov de,al
mov ah,01h ;Introdusco las unidades num 2
int 21h
sub al,30h ;Se cambia de ASCII a dec
mov un,al
mov al,de
mov bl,10
mul bl     ; al = de * 10
add al,un  ; al = de * 10 + un
mov num2,al;se obtiene el segundo numero
mov ah,09h
lea dx,msg3;Muestra mensaje 2
int 21h
mov ah,01h ;Introdusco las decenas num 3
int 21h
sub al,30h ;Se cambia de ASCII a dec
mov de,al
mov ah,01h ;Introdusco las unidades num 3
int 21h
```

```
sub al,30h ;Se cambia de ASCII a dec
mov un,al
mov al,de
mov bl,10
mul bl     ; al = de * 10
add al,un  ; al = de * 10 + un
mov num3,al;se obtiene el segundo numero
;impresion en 3 caracteres
xor ax,ax  ;lipiamos registros
xor bx,bx
mov al,num1;recuperamos el numero 1
aam
mov un,al  ;respaldo unidades
mov al,ah
aam
mov de,al  ;respaldo decenas
mov ce,ah  ;respaldo centenas
mov ah,09h
lea dx,res1;Muestra mensaje 1
int 21h
mov ah,02h
mov dl,ce
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,de
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,un
add dl,30h ;cambio de dec a ascii
int 21h
```

```
mov al,num2;recuperamos el numero 2
aam
mov un,al  ;respaldo unidades
mov al,ah
aam
mov de,al  ;respaldo decenas
mov ce,ah  ;respaldo centenas
mov ah,09h
lea dx,res2;Muestra mensaje 2
int 21h
mov ah,02h
mov dl,ce
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,de
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,un
add dl,30h ;cambio de dec a ascii
int 21h
mov al,num3;recuperamos el numero 3
aam
mov un,al  ;respaldo unidades
mov al,ah
aam
mov de,al  ;respaldo decenas
mov ce,ah  ;respaldo centenas
mov ah,09h
lea dx,res3;Muestra mensaje 3
int 21h
```

```
mov ah,02h
mov dl,ce
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,de
add dl,30h ;cambio de dec a ascii
int 21h
mov dl,un
add dl,30h ;cambio de dec a ascii
int 21h
;Termina programa
mov ah,04ch
int 21h
end
```

Reflexión

Abordar este problema fue de gran dificultad al inicio, pero tras alterar el código que use para hacer una activada en la catedra pude adaptarlo a los requerimientos dados y reforzar mis conocimiento previos.

Actividad 6 – Parte 3

Descripción

Ejecute y documente el siguiente programa, explicando cómo funcionan los procedimientos de retardo.

Desarrollo y Resultados

Procedimientos:

PANT1: Cambia el color del marco de la pantalla colocando líneas vacías del color azul.

PANT0: Cambia el color del fondo de la consola dependiendo del número que contenga en ese momento la variable COLOR.

CURSOR0 y CURSOR1: Mueven el cursor de escritura, cada uno a un punto diferente de la consola.

DESP: Imprime la cadena RETARDO en la posición del cursor de diferentes colores.

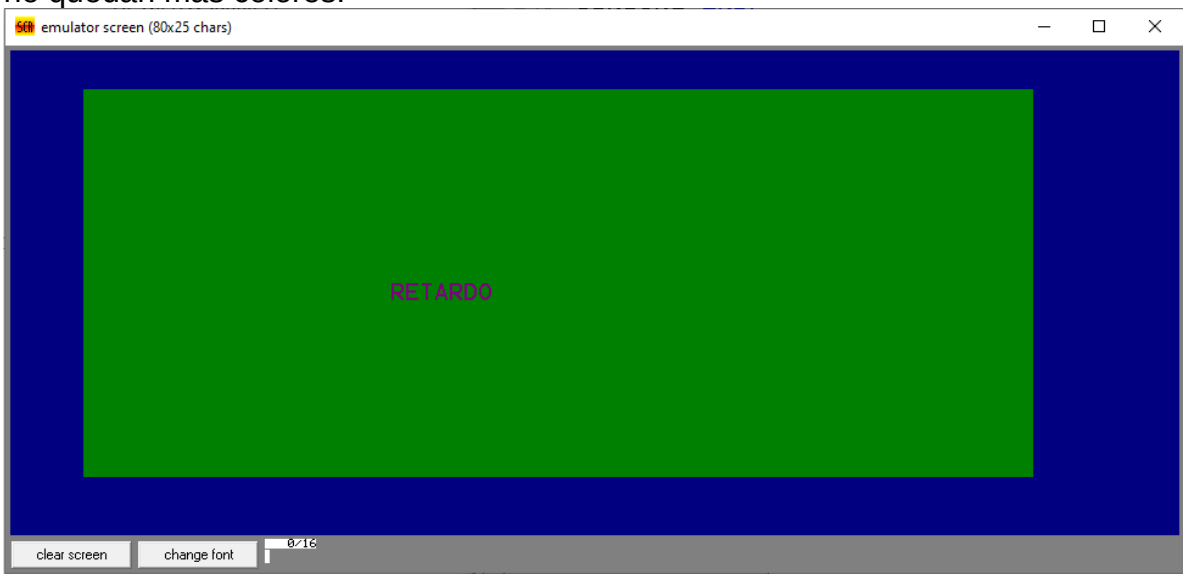
CAMBIO: Se encarga de rotar los colores y ciclar el programa hasta pasar por todos los colores.

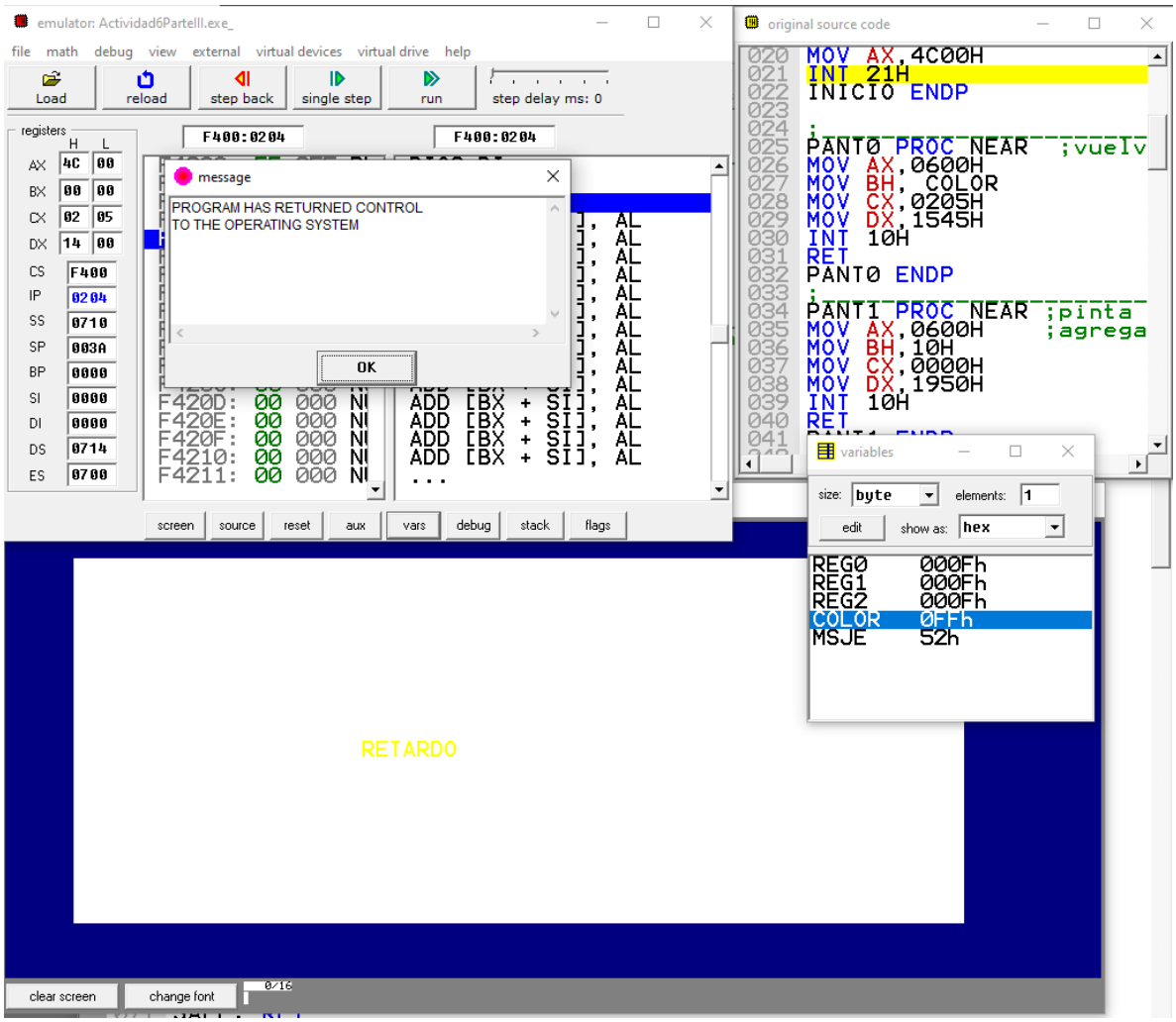
RETARd: Mantiene el programa ciclado durante un breve periodo de tiempo dictado por las variables REG0, REG1 y REG2 decrementados una a una hasta llegar a 0000h, creando la ilusión de un retardo.

RETARi: Funciona igual que el RETARd pero esta incrementa las variables hasta un punto específico y continua a la siguiente también crea la ilusión de un retardo.

Programa sin retardos:

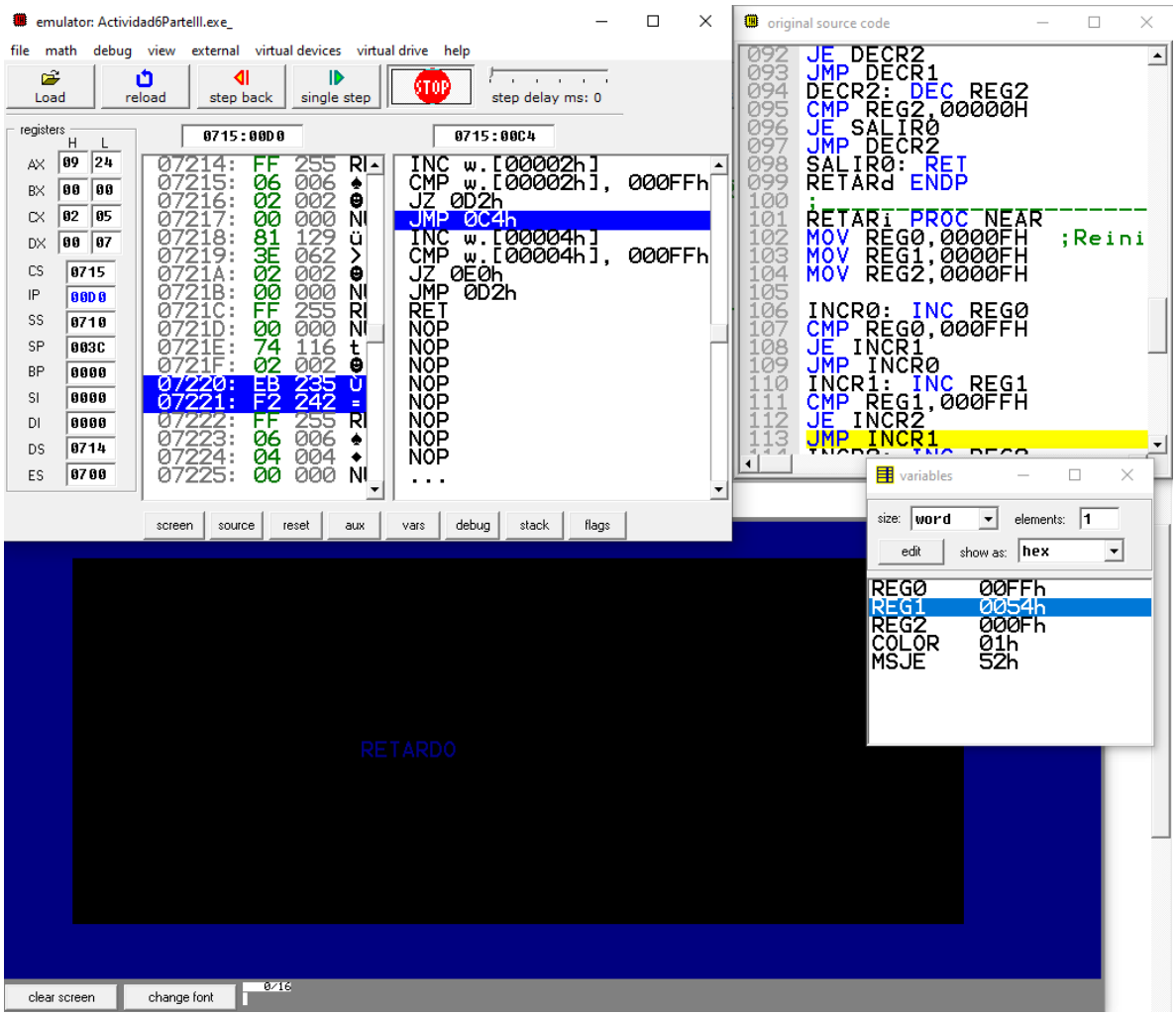
El programa sin retardos se mantiene cambiando el color del fondo de la consola y del mensaje RETARDO de manera casi instantánea hasta que ya no quedan más colores.



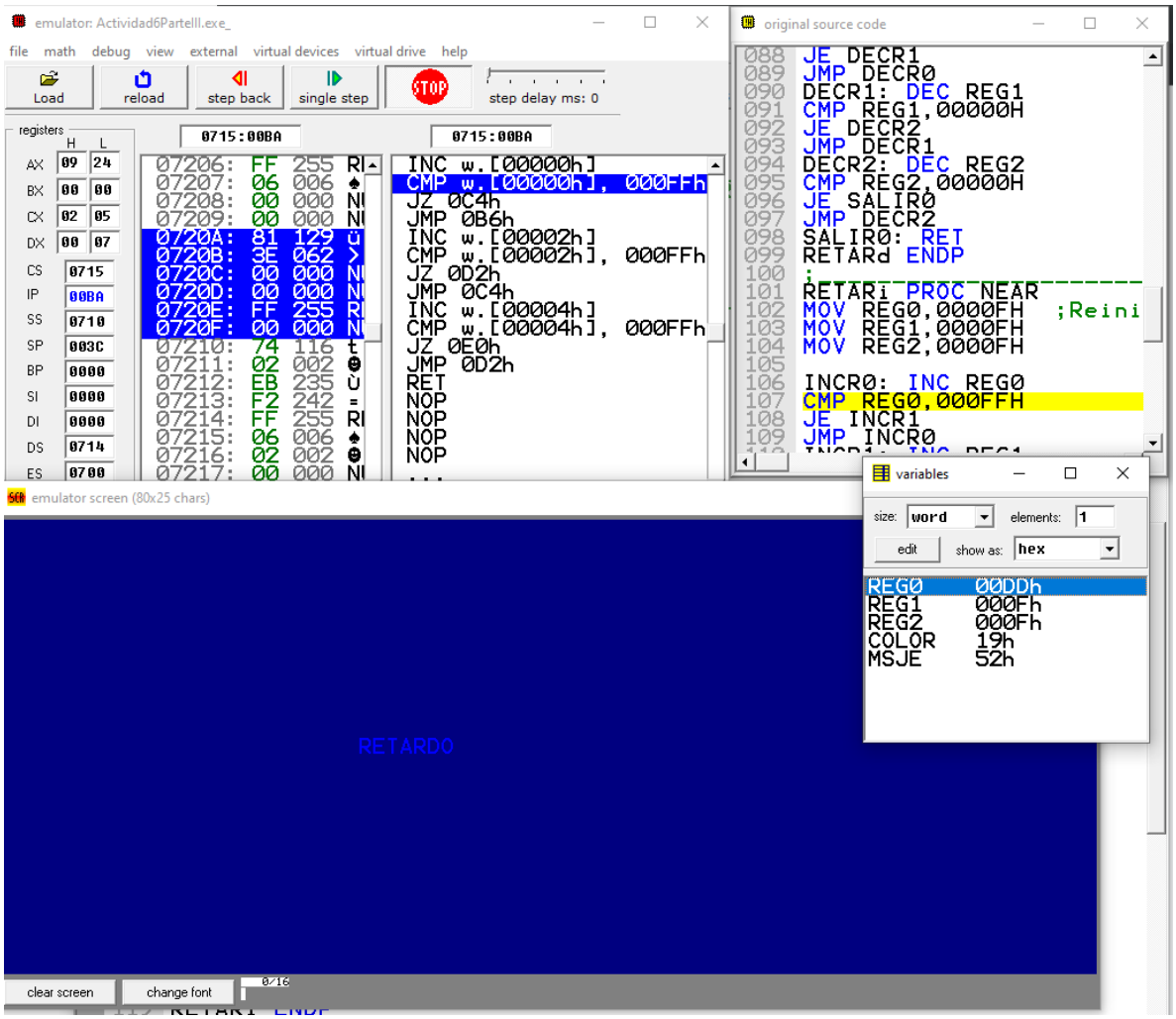


Programa con retardos:

Al quitar comentarios una o más de las líneas de retardo obtenemos un constante cambio en las variables REG0, REG1 y REG2 que hasta que no finalice los procesos de RETARd, RETARi o ambos no cambia de color el mensaje y el fondo creando la ilusión de un retraso en el código que aumenta el tiempo de espera entre cambio y cambio.



Hicieron falta de algunos cambios en el código dado que al ser una emulación tardaba demasiado el retardo original, como disminuir los valores de REG0, REG1 y REG2, cambiar los procedimientos RETARd y RETARi para que reinician los valores de las variables REG0, REG1 y REG2 desde el inicio para evitar que se pusieran valores muy grandes y en RETARi disminuir el valor de comparación a un menor como 00FFH.



Código:

```
PAGE 60,132
TITLE PRACT10.EXE
.MODEL SMALL
;_____________________________________
.STACK 64
.DATA
REG0 DW 0000FH
REG1 DW 0000FH
REG2 DW 0000FH
COLOR DB 00H
MSJE DB "RETARDO","$"
;_____________________________________
.CODE
INICIO PROC FAR
MOV AX, @DATA
MOV DS, AX
CALL PANT1
```

```
CALL CAMBIO
CALL CURSOR1
MOV AX,4C00H
INT 21H
INICIO ENDP

.
;_____________________________________________
PANT0 PROC NEAR  ;vuelve a cambiar el color del marco de PANT1
MOV AX,0600H
MOV BH, COLOR
MOV CX,0205H
MOV DX,1545H
INT 10H
RET
PANT0 ENDP
.
;_____________________________________________
PANT1 PROC NEAR ;pinta un marcoazul en la consola
MOV AX,0600H    ;agregando lineas vacias
MOV BH,10H
MOV CX,0000H
MOV DX,1950H
INT 10H
RET
PANT1 ENDP
.
;_____________________________________________
CURSOR0 PROC NEAR;mueve el cursor de escritura al
MOV AH,02H      ;centro de la pantalla
MOV BH,00H
MOV DX,0C1AH
INT 10H
RET
CURSOR0 ENDP
.
;_____________________________________________
CURSOR1 PROC NEAR; mueve el cursor de escritura a
MOV AH,02H      ;otro punto de la pantalla
MOV BH,00H
MOV DX,1400H
INT 10H
RET
CURSOR1 ENDP
.
;_____________________________________________
CAMBIO PROC NEAR
CALL PANT0
CALL CURSOR0
CALL DESP
CALL RETARd
;CALL RETARi
```

```
;CALL RETARd     ;Anular Retardos para visualizar fenomenos
;CALL RETARi
INC COLOR        ;ingrementa la variable color
CMP COLOR,0FFH
JE SALE
JMP CAMBIO
SALE: RET
CAMBIO ENDP
;_____________________________________________
DESP PROC NEAR   ;Imprime el mensaje RETARDO en el cursor
MOV AH,09H
LEA DX, MSJE
INT 21H
RET
DESP ENDP
;_____________________________________________
RETARd PROC NEAR
MOV REG0,0000FH  ;Reiniciar variables
MOV REG1,0000FH
MOV REG2,0000FH

DECR0: DEC REG0
CMP REG0,00000H
JE DECR1
JMP DECR0
DECR1: DEC REG1
CMP REG1,00000H
JE DECR2
JMP DECR1
DECR2: DEC REG2
CMP REG2,00000H
JE SALIR0
JMP DECR2
SALIR0: RET
RETARd ENDP
;_____________________________________________
RETARi PROC NEAR
MOV REG0,0000FH  ;Reiniciar variables
MOV REG1,0000FH
MOV REG2,0000FH

INCR0: INC REG0
CMP REG0,000FFH
JE INCR1
JMP INCR0
INCR1: INC REG1
CMP REG1,000FFH
```

```
JE INCR2
JMP INCR1
INCR2: INC REG2
CMP REG2,000FFH
JE SALIR1
JMP INCR2
SALIR1: RET
RETARi ENDP
;_____
END INICIO
```

Reflexión

En algunos programas es muy conveniente el uso de retardos para crear pantallas de carga, animaciones o esperar otros procesos, existen funciones en otros lenguajes de programación que hacen esto como la función sleep en C++, pero en ensamblador no por lo que hay que ponerse creativos para crear la ilusión de un retardo.

Actividad 6 – Parte 4

Descripción

Investigar cómo vincular el lenguaje C con el lenguaje ensamblador para escribir programas mixtos.

Desarrollo y Resultados

LA SENTENCIA ASM

La sentencia asm permite incluir código ensamblador dentro del programa C, utilizando los mnemónicos normales del ensamblador. Sin embargo, el uso de esta posibilidad está más o menos limitado según la versión del compilador.

En Turbo C 2.0, los programas que utilizan este método es necesario salir a la línea de comandos para compilarlos con el tradicional compilador de línea, lo cual resulta poco atractivo.

En Turbo C++ 1.0, se puede configurar adecuadamente el compilador para que localice el Turbo Assembler y lo utilice automáticamente para ensamblar, sin necesidad de salir del entorno integrado. Sin embargo, es a partir del Borland C++ cuando se puede trabajar a gusto: en concreto, la versión Borland C++ 2.0 permite ensamblar sin rodeos código ensamblador incluido dentro del listado C.

La sintaxis de asm se puede entender con un ejemplo:

```
main()
{
  int dato1, dato2, resultado;


  printf("Dame dos números: ");
  scanf("%d %d", &dato1, &dato2);

    asm push ax; push cx;
    asm mov  cx,dato1
    asm mov  ax,0h
  mult:
    asm add  ax,dato2
    asm loop mult
    asm mov  resultado,ax
    asm pop  cx; pop ax;


    printf("Su producto por el peor método da: %d", resultado);
}
```

Como se ve en el ejemplo, los registros utilizados son convenientemente preservados para no alterar el valor que puedan tener en ese momento (importante para el compilador). También puede observarse lo fácil que resulta acceder a las variables. Ah, cuidado con BP: el registro BP es empleado mucho por el compilador y no conviene tocarlo (ni siquiera guardándolo en la pila).

Esta es la única sintaxis soportada por el Turbo C 2.0; sin embargo, en las versiones más modernas del compilador se admiten las llaves '{' y '}' para agrupar varias sentencias asm:

```
    asm {
      push ax; push cx;
      mov  cx,dato1
      mov  ax,0h }
  mult: asm {
      add  ax,dato2
      loop mult
      mov  resultado,ax
      pop  cx; pop ax;
      }
```

Fuentes:

UAM. (n.d.). Capítulo XIII: EL ENSAMBLADOR Y EL LENGUAJE C. Arantxa II.

http://arantxa.ii.uam.es/~gdrivera/labetcii/curso0001/udigxiii.htm#:~:text=La%20sentencia%20asm%20permite%20incluir,seg%C3%BAn%20la%20versi%C3%B3n%20del%20compilador

Reflexión

La combinación del lenguaje ensamblador y C nos permite aprovechar los conocimientos obtenidos en este curso en un ambiente que ya dominamos como es el lenguaje C, aprovechando las ventajas de cada uno y en especial la velocidad que nos da el ensamblador.