A5

Apéndice A5
Programas de ejemplo

Se han escrito varios programas en ensamblador que muestran el funcionamiento de los componentes que incorpora el simulador. Realizar el negativo de una imagen de niveles de gris cargada en memoria o implementar el juego de la serpiente (*snake* en inglés) son algunos ejemplos que pasamos a comentar en las secciones siguientes.

Demostración:

Nombre: **negativo.asm**
Procesador: **8085**
Dispositivos: **Pantalla Grafica (256)**
Líneas: **28**

Descripción:

Programa de muestra que invierte la imagen actual en pantalla.

```
.org 100H


mvi H, 10H
mvi L, 00H  ; en HL la posicion de memoria

otro:
mvi a, FFh
SUB M

mov M, a
INX H
;comprueba parte alta
mvi a, 4EH
cmp H
JZ comprueba_LO
jmp otro

comprueba_LO:           ; comprueba parte baja
mvi a, 80h
cmp L
JZ fin
jmp otro

fin:
hlt
```

Demostración:

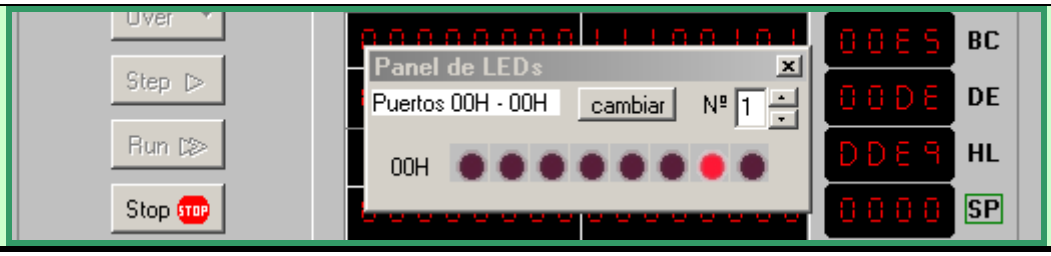
Nombre: **leds.asm**
Procesador: **8085**
Dispositivos: **Panel de Leds (1 linea)**
Generador de Interrupciones
Líneas: **47**

Descripción:

Programa que genera un movimiento ordenado y oscilante de una luz mediante un vector de leds.

```
;LUZ COCHE FANTASTICO
;
; requiere  led en puerto 0
;           interrupcion rst 7.5 cada 1 segundo


.org 1000h

mvi c,128
mov d,0
salto:
mov a,c
out 0h
jmp salto



.org 003ch
;Subrutina cada 1 segundo
sub:
mov a,d
cpi 0
jnz mub
```

```
mov a,c           ;direccion -->
rar
mov c,a
mov a,c
cpi 0
jnz rub
mvi c,1
mvi d,1

mub:
mov a,c           ;direccion <--
ral
mov c,a
mov a,c
cpi 0
jnz rub
mvi c,128
mvi d,0

rub:
ei
ret
```



Ejemplo de una utilidad:

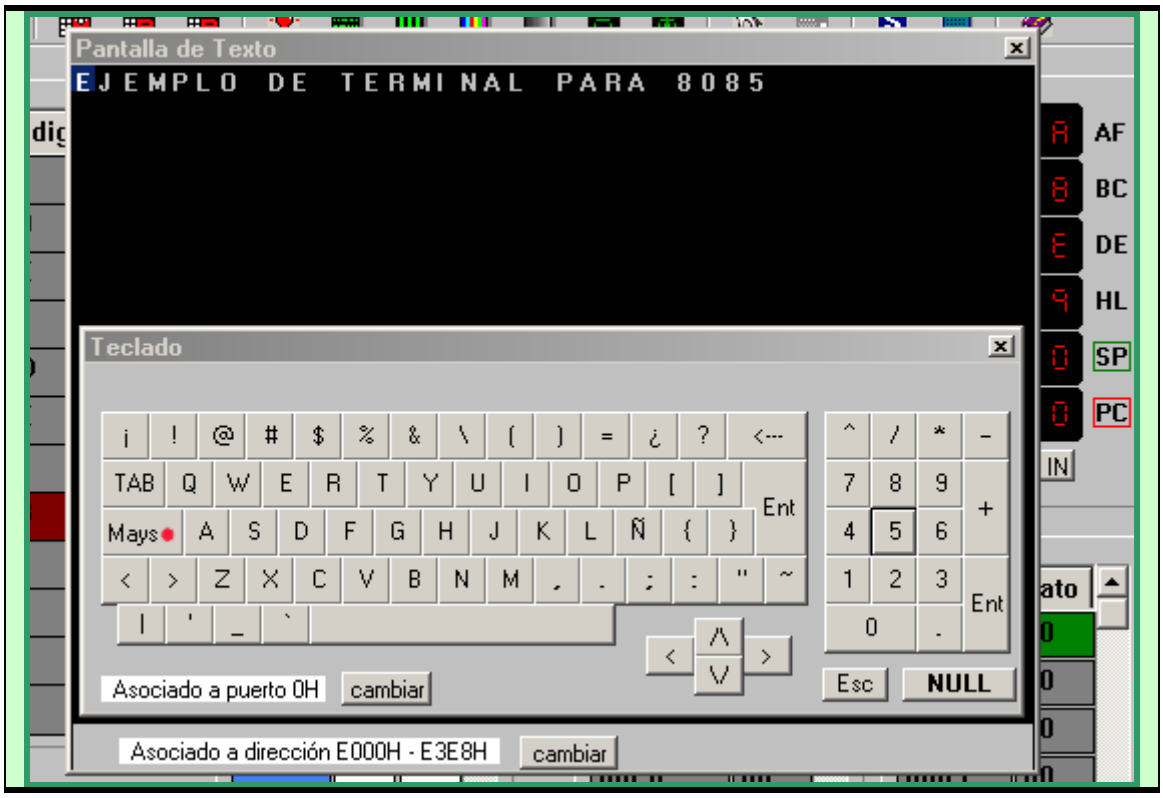
Nombre: **pantalla.asm**
Procesador: **8085**
Dispositivos: **Pantalla de Texto**
Teclado
Generador de Interrupciones por teclado
Líneas: **59**

Descripción:

Simulación de un terminal de texto.

```
; Ejemplo de programa
; Simulador de terminal
; Asociado a interrupción TRAP

.define
   texto E000h
   tamtexto 25*40
.org 1000h

; -------------------------------
; PROGRAMA PRINCIPAL
```

```
; --------------------------------

mvi B, E0h
mvi C, 00h
call clear_all
bucle:
      jmp bucle


.org 0024h              ; Direccion de interrupción TRAP

; --------------------------------
; RUTINA QUE LEE DEL TECLADO Y ESCRIBE EN MEMORIA
; --------------------------------
      in 00h
      cpi 0
      jz no_tecla
      stax B
      inx B
no_tecla:
      ret

clear_all:
      LXI H, texto                  ;cargamos origen
      LXI D, texto+tamtexto         ;cargamos fin
   repite_c:
      MVI A,32
      MOV M,A                       ;borrar punto de memoria
      INX H                         ;incrementar direccion
      call comparador
      cpi 1
      jz fin_clear
      jmp repite_c
   fin_clear:
      ret

comparador:                   ;compara DE con HL (en 16 bits). Devuelve
A=1 si igual
      MOV A,E
      CMP L
      JNZ no_igual
      MOV A,D
      CMP H
      JNZ no_igual
      MVI A, 1
      ret
   no_igual:
      MVI A,0
      ret
```



Ejemplo de una utilidad:

Nombre: **Reloj.asm**
Procesador: **8085**
Dispositivos: **Visualizador de 7 segmentos**
Generador de Interrupciones
Líneas: **83**

Descripción:

Programa que convierte a un 8085 en un reloj digital con segundero y minutero.

```
; Ejemplo de programa
; Reloj digital
; Asociado a interrupción TRAP

.data DDh
      DB 77h, 44h, 3Eh, 6Eh, 4Dh, 6Bh, 7Bh, 46h, 7Fh, 4Fh
      ;digitos sin punto
      DB F7h, C4h, BEh, EEh, CDh, EBh, FBh, C6h, FFh, CFh
      ;digitos con punto

.org 1000h

; --------------------------------
; PROGRAMA PRINCIPAL
; --------------------------------
      mvi B, 00h
      mvi C, DDh
      mvi D, 00h
```

```
      mvi E, DDh
      mvi L, E7h
      mvi H, DDh
      mvi A, 77h
      out 6d
      out 7d
      out 4d
      mvi A, F7h
      out 5d
bucle:
      jmp bucle


.org 0024h              ; Direccion de interrupción TRAP

; -------------------------------
; RUTINA QUE AUMENTA EL TIEMPO
; -------------------------------
      ldax B
      cpi 4Fh
      jz suma_segundo
      inx B
      ldax B
      out 7d
      ret
suma_segundo:
      mvi C,DDh
      ldax B
      out 7d
      ldax D
      cpi 6Bh
      jz suma_minuto1
      inx D
      ldax D
      out 6d
      ret
suma_minuto1:
      mvi E, DDh
      ldax D
      out 6d
      mov D, C
      mov C, L
      ldax B
      cpi CFh
      jz suma_minuto2
      inx B
      ldax B
      out 5d
      mov L, C
      mov C, D
      mov D, 00h
      ret
suma_minuto2:
      mvi L, E7h
      mov D, C
      mov C, L
      ldax B
      out 5d
      mov D, C
      mov C, H
      inx B
```

```
      ldax B
      out 4d
      mov H, C
      mov C, D
      mov D, 00h
      ret
```

Ejemplo de un Juego Interactivo:

Nombre: **Snake.asm**
Procesador: **8085**
Dispositivos: **Teclado**
Pantalla Grafica
Generador de Interrupciones
Líneas: **236**

DESCRIPCIÓN:

Conocido juego de la serpiente, consiste en comer los puntos de comida si mordernos a nosotros mismos.

```
; Ejemplo de Programa en ensamblador para el simulador de 8085
; SNAKE 8085
; Comer sin mordenos a nosotros mismos

.define
      memVideo A000h                  ;Origen de la memoria de Video
      sizeVideo 160*100               ;Tamaño de la memoria de Video
      mitadVideo memVideo+sizeVideo/2  ;Posicion intermedia
      teclado 0h                      ;Puerto del teclado
      up    (-160)&FFFFh
      down  160
      left  -1
      right 1
      tecla_up    1Eh
      tecla_down  1Fh
      tecla_left  11h
      tecla_right 10h
      comienzo    mitadVideo+80

.data 0b
cuanto: dB 10h
cola: dW comienzo
pos: dB 0
pos_pantalla: dW
memVideo+580H,memVideo+1000H,memVideo+2500H,memVideo+3000H
```

```
.org 500H
;     call clear_all
      LXI H, comienzo
      call pon_comida
  repite:
      IN teclado
      jmp repite


comparador:                   ;compara DE con HL (en 16 bits). Devuelve
A=1 si igual
      MOV A,E
      CMP L
      JNZ no_igual
      MOV A,D
      CMP H
      JNZ no_igual
      MVI A, 1
      ret
   no_igual:
      MVI A,0
      ret

compar_inf:                   ;compara DE con HL (en 16 bits). Devuelve
A=1 si menor DE
      MOV A,D
      CMP H
      JM menor
      JZ comp_menor
      MVI A, 0
      ret
   menor:
      MVI A,1
      ret
   comp_menor:
      MOV A,E
      CMP L
      JM menor
      MVI A, 0
      ret

compar_sup:                   ;compara DE con HL (en 16 bits). Devuelve
A=1 si mayor DE
      MOV A,H
      CMP D
      JM menor
      JZ comp_menor2
      MVI A, 0
      ret
   menor2:
      MVI A,1
      ret
   comp_menor2:
      MOV A,L
      CMP E
      JM menor2
      MVI A, 0
      ret
```

```
clear_all:
      LXI H, memVideo              ;cargamos origen memoria video
      LXI D, memVideo+sizeVideo    ;cargamos fin memoria video
   repite_c:
      MVI A,0
      MOV M,A                      ;borrar punto de memoria
      INX H                        ;incrementar direccion
      call comparador
      cpi 1
      jz fin_clear
      jmp repite_c
   fin_clear:
      ret


moverse:                       ;en registro A el movimiento
      CPI tecla_up
      cz haz_arriba
      CPI tecla_down
      cz haz_abajo
      CPI tecla_left
      cz haz_izqda
      CPI tecla_right
      cz haz_decha
      ret

comprobador:                   ;comprueba que no se excede de la memoria
de video

      LXI D, memVideo
      call compar_sup
      cpi 1                    ;Si es 0 es q DE no es mayor que HL
      jz fin
      LXI D, memVideo+sizeVideo
      call compar_inf
      cpi 1                    ;Si es 0 es que DE no es menor que HL
      jz fin
      ret


haz_arriba:                    ;moverse arriba
      LXI D, up
      call pon_punto
      ret

haz_abajo:                     ;moverse abajo
      LXI D, down
      call pon_punto
      ret

haz_izqda:                     ;moverse izda
      LXI D, left
      call pon_punto
      ret

haz_decha:                     ;moverse decha
      LXI D, right
      call pon_punto
      ret

pon_punto:
```

```
      DAD d
      call comprobador
      mov A,M
      cpi FFh
      jz fin
      cpi FFh/2
      CZ pon_comida
      MVI M, FFh              ;pintar
      call comp_borra
      ret

comp_borra:
      push psw
      LDA cuanto
      cpi 0
      jz borra_punto          ; si el cuanto es 0 hay que borrar
      DCR a
      STA cuanto              ; decrementamos y almacenamos el cuanto
      pop psw
      ret

  borra_punto:
      push h
      push d
      lhld cola
      LXI d, up               ; mirar arriba
      DAD d
      mov a,M
      cpi FFh                 ;si hay punto hay que borrarlo
      jz elimina
      lhld cola
      LXI d, down             ; mirar abajo
      DAD d
      mov a,M
      cpi FFh                 ;si hay punto hay que borrarlo
      jz elimina
      lhld cola
      LXI d, left             ; mirar izquierda
      dad d
      mov a,M
      cpi FFh                 ;si hay punto hay que borrarlo
      jz elimina
      lhld cola
      LXI d, right            ; mirar izquierda
      DAD d
   elimina:
      MVI a, 00h
      mov M,a
      SHLD cola               ;almacenamos la nueva cola
      pop d
      pop h
      pop psw
      ret                     ;regresa al call de comp_borra

pon_comida:
      push psw
      push d

      LDA cuanto
      adi 10h
      STA cuanto
```

```
      LDA pos                 ;cargamos la posicion actual
      rlc
      Mov e,a
      rrc
      Mvi d,0
      INR a                   ;incrementamos el desplazamiento
      ANI 11b                 ;impedimos que sea mayor que 4
      STA pos                 ;guardamos la posicion  actual

      push h
      LXI h, pos_pantalla
      DAD d
      Mov E,M
      INX h
      Mov D,M
      XCHG                    ;ya tenemos la direccion en HL
      mvi a, FFh/2
      mov M, a                ;coloreamos el punto

      pop h                   ;recuperamos lo guardado
      pop d
      pop psw
      ret

fin:
      hlt

.org 3Ch          ;Interrupcion del timer (RST 7.5)
      call moverse
      EI
      Ret
```