PROGRAMANDO EL MICROCONTROLADOR 8051 EN EL SIMULADOR: EdSim51™

César Martín Cruz Salazar

PROGRAMANDO EL MICROCONTROLADOR 8051 EN EL SIMULADOR EDSIM51

1era Edición

Lic. César Martín Cruz Salazar

Cómo programar el Microcontrolador 8051 en el simulador EDSIM51.
Primera edición.
César Martín Cruz Salazar

MARCAS COMERCIALES:
8051, MCS-51 son propiedad de Intel Corporation
EDSIM51 es propiedad de James Rogers
ATMEL es propiedad de Atmel Corporation
NXP es propiedad de NXP Semiconductors
Windows XP Professional, Windows Vista y Windows 7 son propiedad de Microsoft

Primera edición: Abril 2012

Este libro se escribió para ti, protégelo de la copia.

NOTA IMPORTANTE
La información contenida en esta obra tiene un fin exclusivamente didáctico.
El autor no será jurídicamente responsable por errores u omisiones, daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, y en los archivos adjuntos, ni por la utilización indebida que pudiera dársele.

Ventas de este libro:
www.elfuturoaqui.com
www.oferbook.com
Teléfono celular: 990595414 (Movistar)
Lima. Perú

INDICE

Págs.

INTRODUCCION

Este libro nace por el motivo que no hay suficiente información sobre ejemplos de programas en lenguaje ensamblador del 8051 en Internet y tampoco libros impresos sobre este tema en el Perú, entonces me propuse hacer una recopilación de programas que he desarrollado durante el dictado de cursos en la Facultad de Ciencias de la Universidad Nacional de Ingeniería (UNI).
Estos programas y ejercicios servirán de ejemplos y motivo de práctica para los estudiantes que se inscribirán en cursos en donde se requiera el aprendizaje del lenguaje ensamblador del 8051.

El libro incluye:
Programas sobre manejo de la memoria de datos y registros, manejo de leds, manejo de switches y displays de 7 segmentos. Se incluye también los programas desarrollados a partir de los algoritmos comunes tratados en los cursos de la carrera de Ciencia de la Computación.

El libro no incluye:
La teoría sobre el microcontrolador 8051 y electrónica digital porque considero que hay suficiente información sobre estos temas en Internet.

Notas sobre el ensamblador

: Después de una etiqueta, excepto para EQU

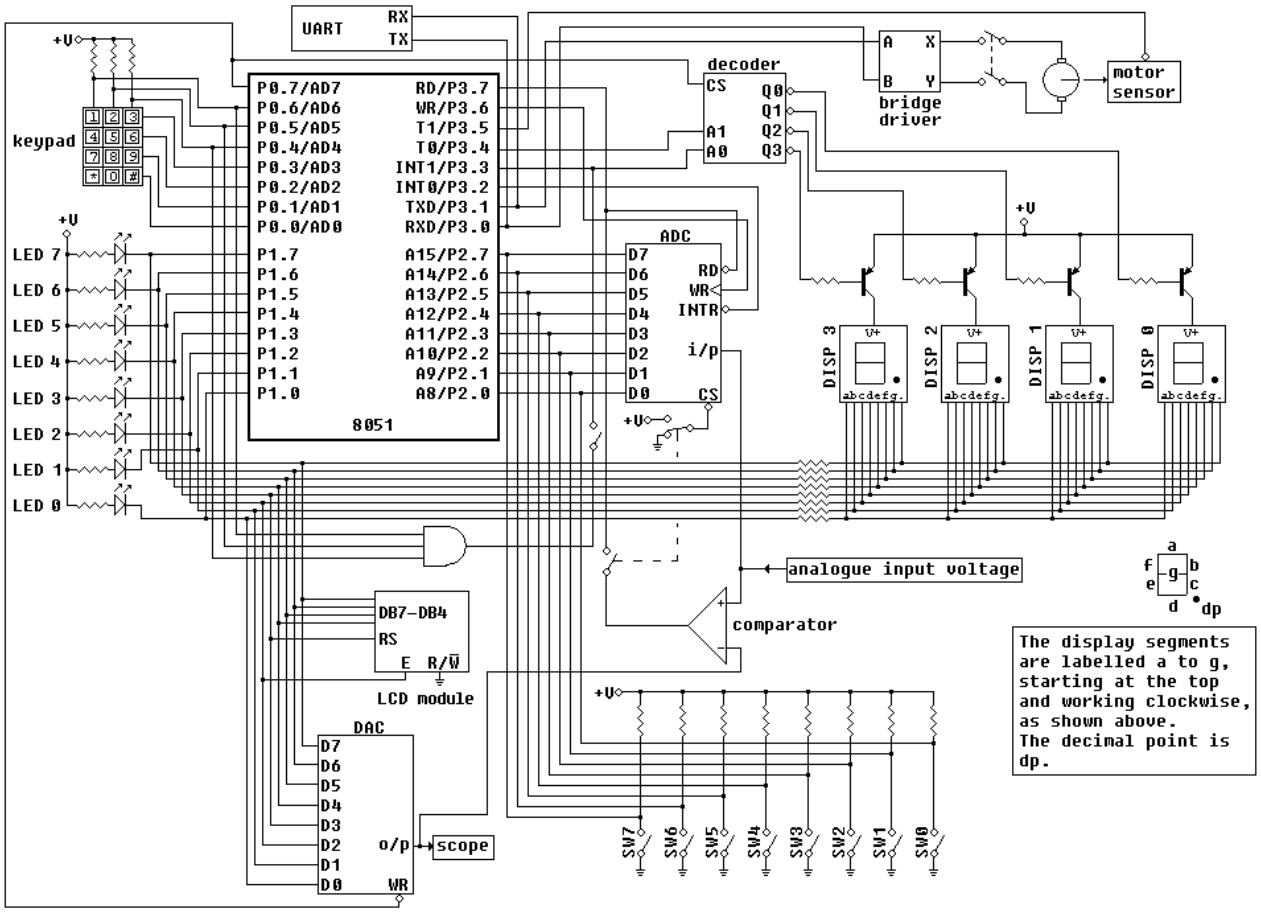
; Antes de un comentario

Las Pseudo Operaciones incluyen:
DB Define Byte; localiza datos de longitud de un byte en memoria
DW Define Word; localiza datos de longitud de una palabra en memoria
END Fin del programa
EQU Equiparar; define la etiqueta adjunta
ORG Fija el origen; localiza el código objeto subsecuente empezando en la dirección especificada
$ Valor corriente de la posición(contador de programa)

UTILIZACION DEL SIMULADOR EDSIM51

Para ejecutar todos los programas del libro se debe utilizar el **simulador EDSIM51** desarrollado por ***James Rogers***, que lo puede descargar de: http://www.edsim51.com. Existe también una guía de usuario en http://www.edsim51.com/simInstructions.html sobre cómo utilizar este simulador.

El diagrama esquemático de los periféricos en el simulador EDSIM51 que están conectados al 8051 es la siguiente :

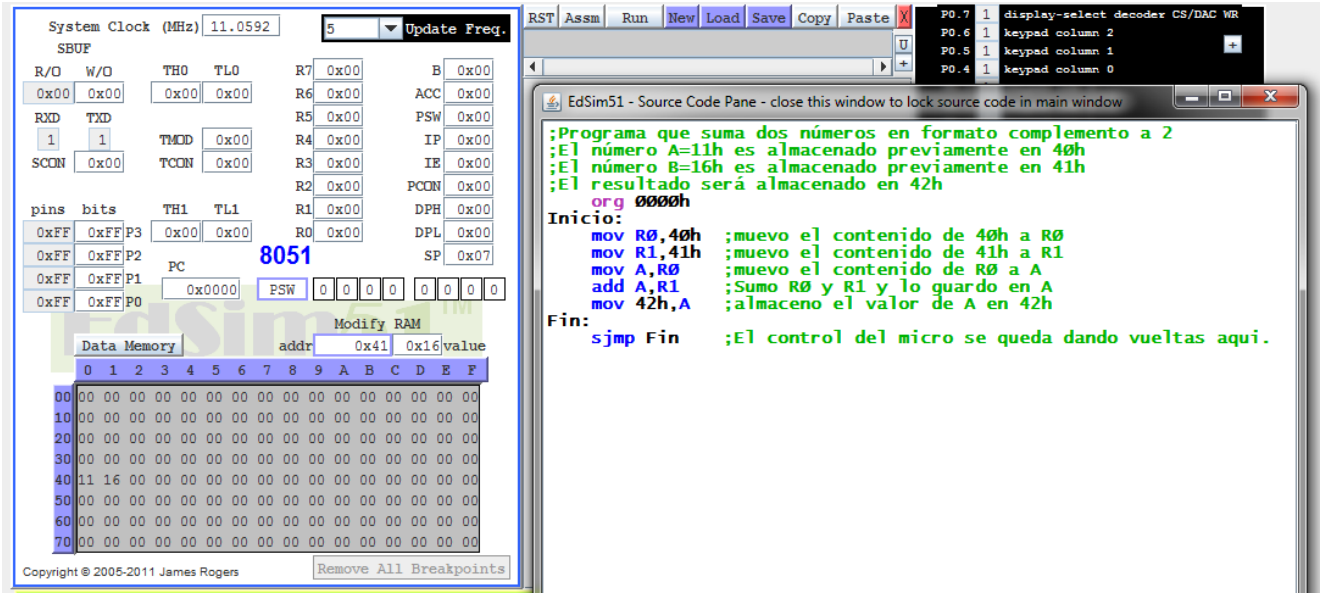


Programa #1:

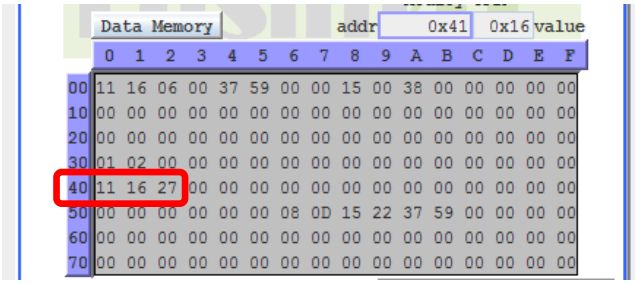
Programa que realiza la suma de dos números hexadecimales por ejemplo “A + B” en formato complemento a 2.
El listado es el siguiente:

```
;Programa que suma dos números en formato complemento a 2
;El número A=11h es almacenado previamente en 40h
;El número B=16h es almacenado previamente en 41h
;El resultado será almacenado en 42h
        org 0000h
Inicio:
        mov R0,40h      ;muevo el contenido de 40h a R0
        mov R1,41h      ;muevo el contenido de 41h a R1
        mov A,R0        ;muevo el contenido de R0 a A
        add A,R1        ;Sumo R0 y R1 y lo guardo en A
        mov 42h,A       ;almaceno el valor de A en 42h
Fin:
        sjmp Fin        ;El control del micro se queda dando vueltas aquí.
        END
```

En el EDSIM51:



Resultado:

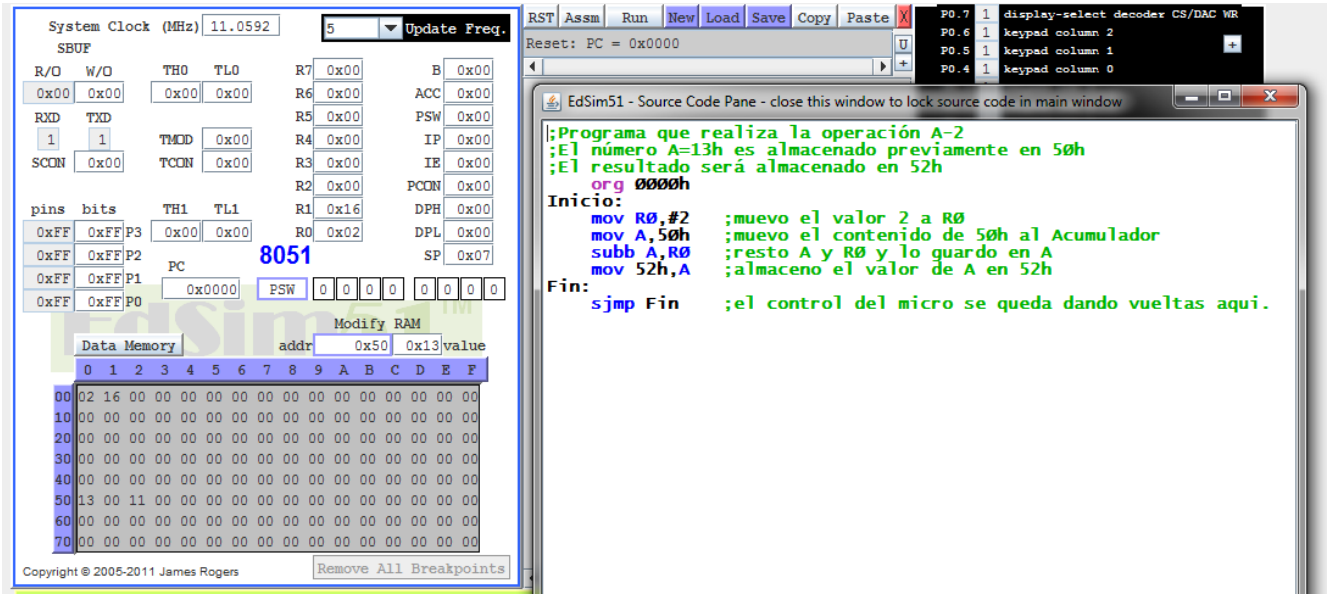


Programa #2:
Programa que realiza la operación aritmética "A – 2".

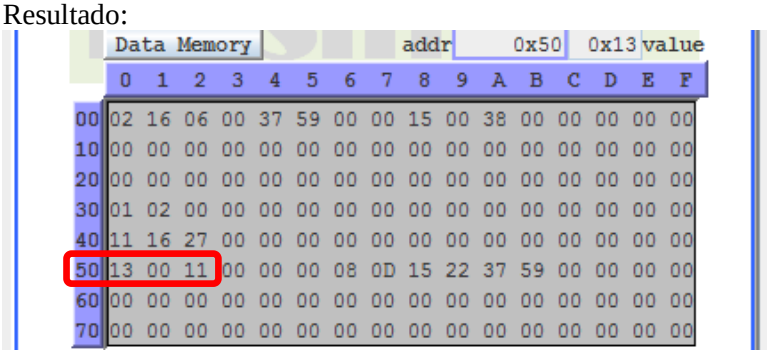
El listado es el siguiente:

```
;Programa que realiza la operación A-2
;El número A=13h es almacenado previamente en 50h
;El resultado será almacenado en 52h
        org 0000h
Inicio:
        mov R0,#2       ;muevo el valor 2 a R0
        mov A,50h       ;muevo el contenido de 50h al Acumulador
        subb A,R0       ;resto A y R0 y lo guardo en A
        mov 52h,A       ;almaceno el valor de A en 52h
Fin:
        sjmp Fin        ;el control del micro se queda dando vueltas aquí.
        END
```

En el EDSIM51:



Resultado:

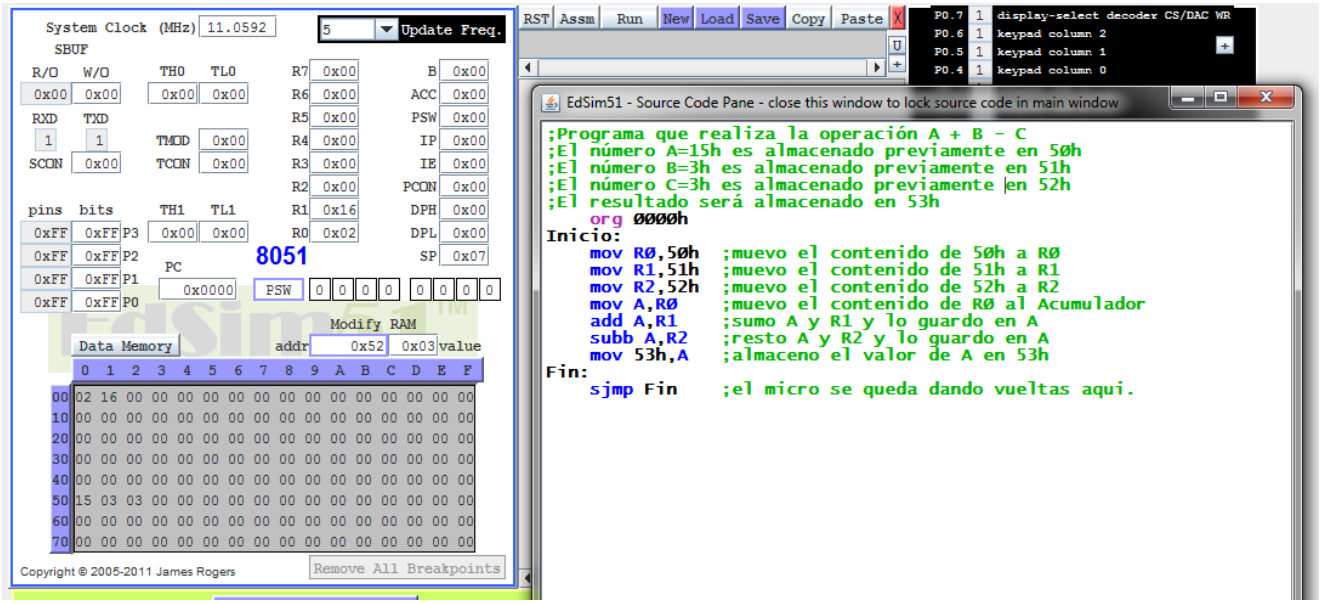


Programa #3:
Programa que efectúa la operación aritmética "A + B – C".

El listado es el siguiente:

```
;Programa que realiza la operación A + B - C
;El número A=15h es almacenado previamente en 50h
;El número B=3h es almacenado previamente en 51h
;El número C=3h es almacenado previamente en 52h
;El resultado será almacenado en 53h
        org 0000h
Inicio:
        mov R0,50h      ;muevo el contenido de 50h a R0
        mov R1,51h      ;muevo el contenido de 51h a R1
        mov R2,52h      ;muevo el contenido de 52h a R2
        mov A,R0        ;muevo el contenido de R0 al Acumulador
        add A,R1        ;sumo A y R1 y lo guardo en A
        subb A,R2       ;resto A y R2 y lo guardo en A
        mov 53h,A       ;almaceno el valor de A en 53h
Fin:
        sjmp Fin        ;el micro se queda dando vueltas aquí.
        END
```

En el EDSIM51:



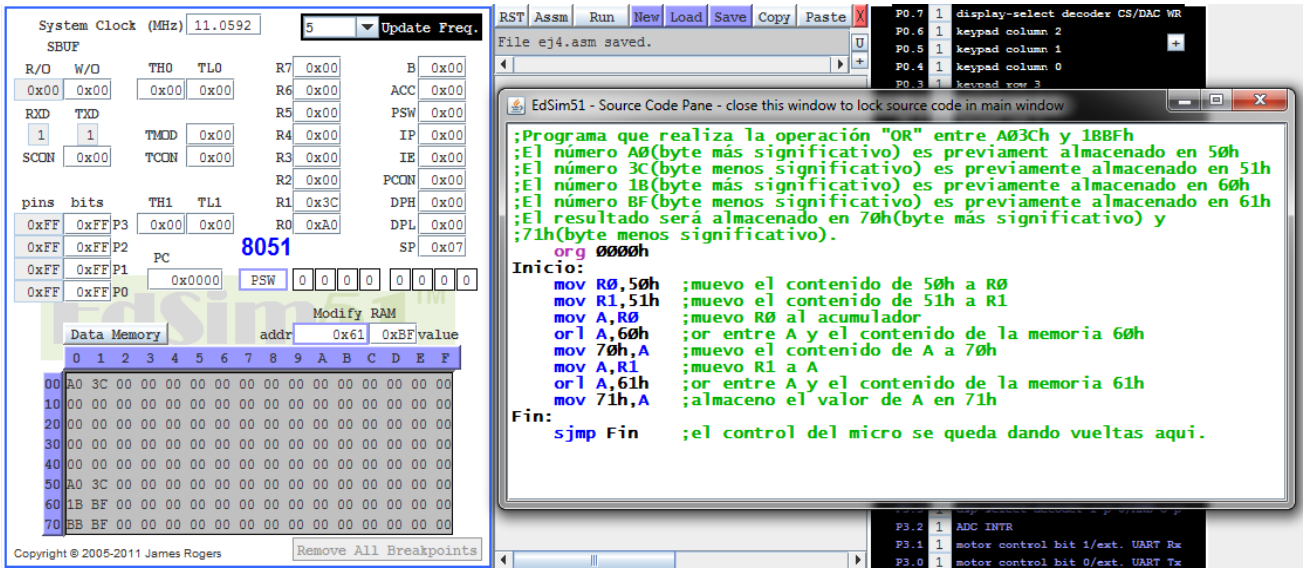
Ejercicio #1:
Realice un programa que efectúe la operación "A – B + C - D". Considere para "A" la dirección de memoria 60h, para "B" la dirección 61h, para "C" 62h y para "D" 63h.

Ejercicio #2:
Realice un programa que efectúe la operación "A – B + C". Considere las direcciones 50h, 51h y 52h para A, B y C. El resultado lo guarde en 53h y lo muestre en el Puerto P1. **Sugerencia:** Utilizar mov P1,53h.

Programa #4:
Programa que realiza la operación “OR” entre los números de 16 bits A03Ch y 1BBFh.
El listado es el siguiente:

```
;Programa que realiza la operación "OR" entre A03Ch y 1BBFh
;El número A0(byte más significativo) es almacenado en 50h
;El número 3C(byte menos significativo) es almacenado en 51h
;El número 1B(byte más significativo) es almacenado en 60h
;El número BF(byte menos significativo) es almacenado en 61h
;El resultado será almacenado en 70h(byte más significativo) y
;71h(byte menos significativo).
        org 0000h
Inicio:
        mov R0,50h      ;muevo el contenido de 50h a R0
        mov R1,51h      ;muevo el contenido de 51h a R1
        mov A,R0        ;muevo R0 al acumulador
        orl A,60h       ;or entre A y el contenido de la memoria 60h
        mov 70h,A       ;muevo el contenido de A a 70h
        mov A,R1        ;muevo R1 a A
        orl A,61h       ;or entre A y el contenido de la memoria 61h
        mov 71h,A       ;almaceno el valor de A en 71h
Fin:
        sjmp Fin        ;el control del micro se queda dando vueltas aquí.
        END
```

En el EDSIM51:



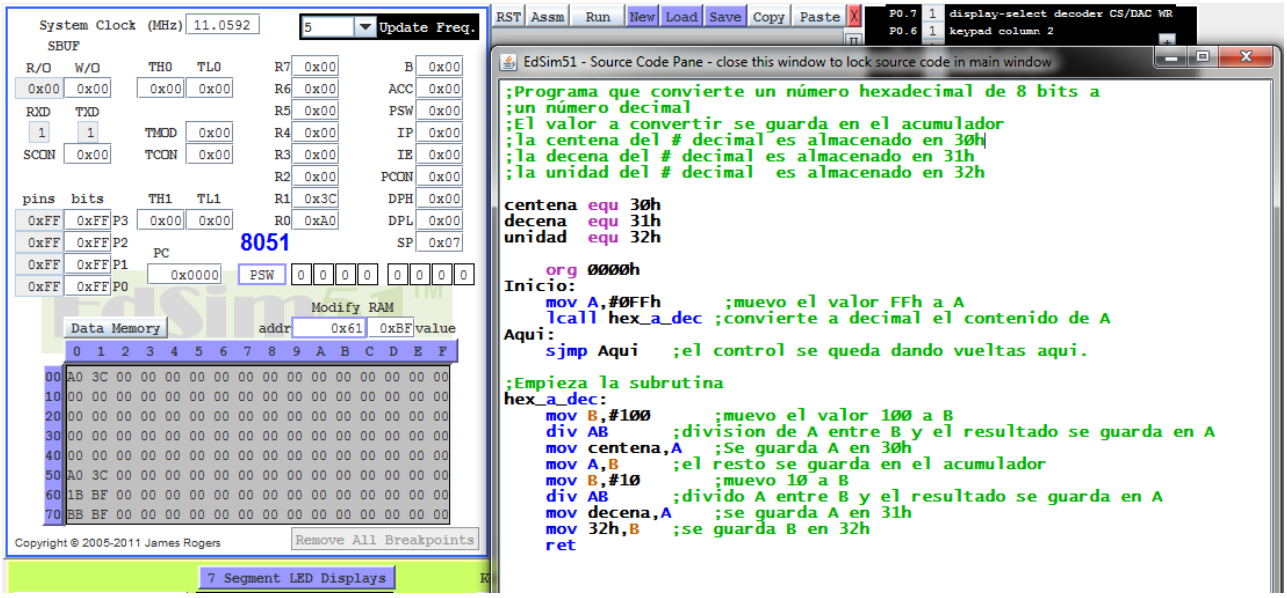
Ejercicio #3:
Hacer un programa que calcule la operación “XOR” entre los números hexadecimales de 16 bits CCDDh y F0EEh.
Ejercicio #4:
Hacer un programa que calcule la operación “AND” entre los números hexadecimales de 16 bits 99CCh y EEB0h.

Programa #5:
Programa que convierte un número hexadecimal de 8 bits a un número decimal de tres dígitos. Cada dígito ocupa una posición de memoria diferente.

El listado es el siguiente:

```
;Programa que convierte un número hexadecimal de 8 bits a
;un número decimal
;El valor a convertir se guarda en el acumulador
;la centena del # decimal es almacenado en 30h
;la decena del # decimal es almacenado en 31h
;la unidad del # decimal  es almacenado en 32h

centena equ 30h
decena  equ 31h
unidad  equ 32h

        org 0000h
Inicio:
        mov A,#0FFh     ;Muevo el valor FFh a A, este valor se va a convertir
        lcall hex_a_dec ;convierte a decimal el contenido de A
Fin:
        sjmp Fin        ;el control se queda dando vueltas aqui.

;Empieza la subrutina
hex_a_dec:
        mov B,#100      ;muevo el valor 100 a B
        div AB          ;division de A entre B y el resultado se guarda en A
        mov centena,A   ;Se guarda A en 30h
        mov A,B         ;el resto se guarda en el acumulador
        mov B,#10       ;muevo 10 a B
        div AB          ;divido A entre B y el resultado se guarda en A
        mov decena,A    ;se guarda A en 31h
        mov 32h,B       ;se guarda B en 32h
        ret
        END
```

En el EDSIM51:

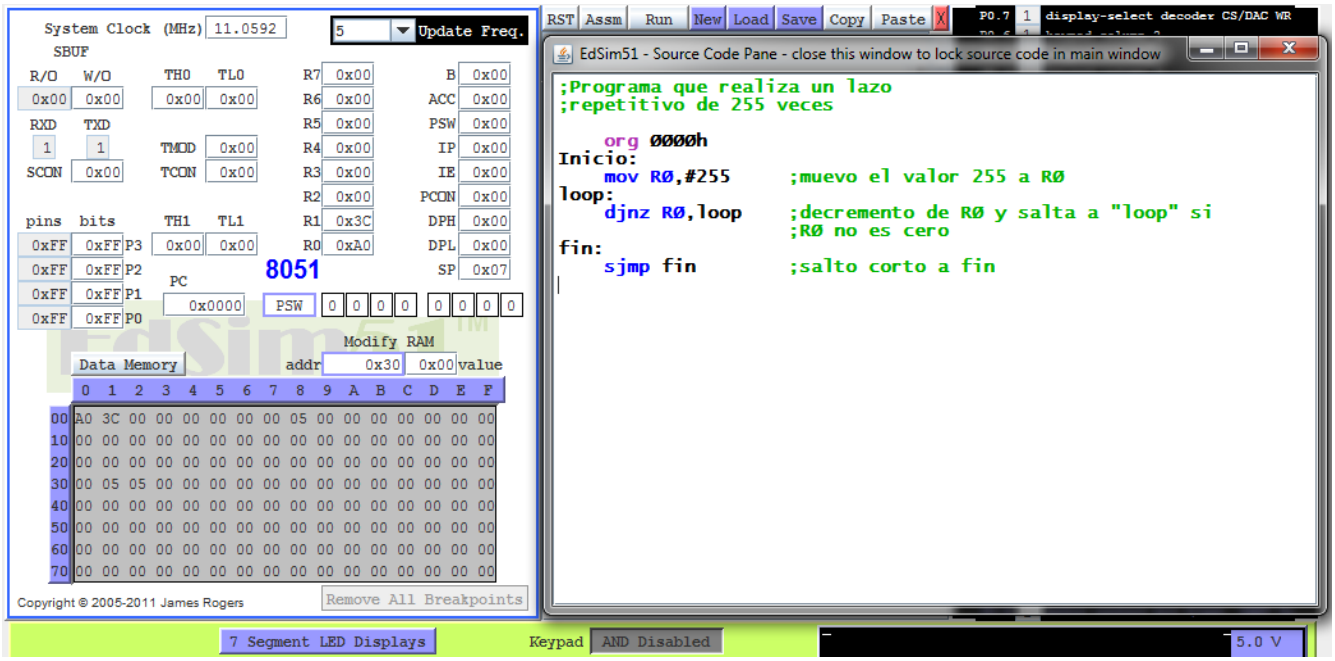


Programa #6:
Programa que realiza un bucle repetitivo de 255 veces. Esto se considera un tiempo de retardo para el microcontrolador.

El listado es el siguiente:

```
;Programa que realiza un lazo
;repetitivo de 255 veces

        org 0000h
Inicio:
        mov R0,#255     ;muevo el valor 255 a R0
loop:
        djnz R0,loop    ;decremento de R0 y salta a "loop" si
                        ;R0 no es cero
fin:
        sjmp fin        ;salto corto a fin
        END
```

En el EDSIM51:

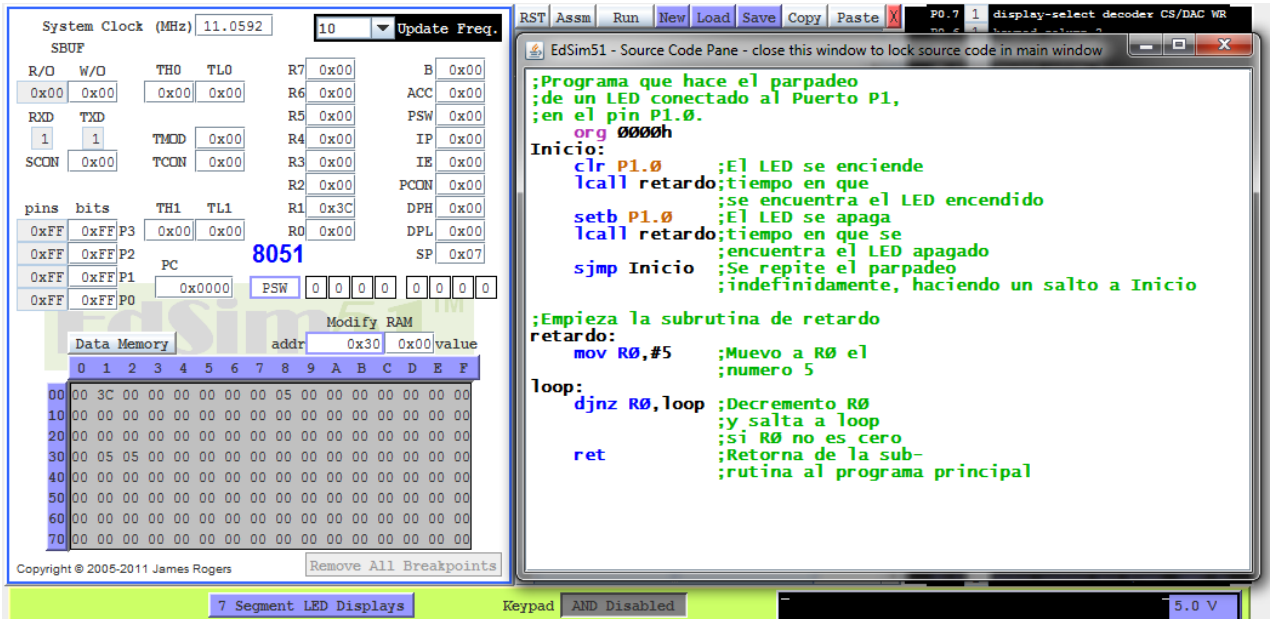


Ejercicio #5:
Realice un programa que ejecute un bucle repetitivo de 400 veces.
Sugerencia: Utilice lazos anidados.
Programa #7:
Programa que realiza el parpadeo de un LED conectado al puerto P1 en el pin P1.0.

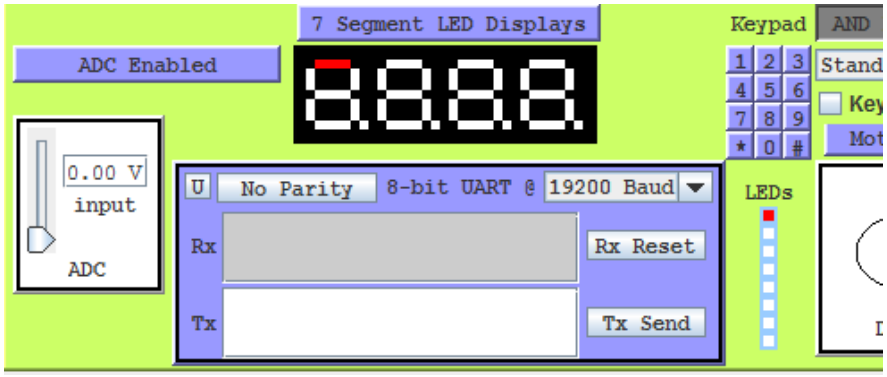
El listado del programa es el siguiente:

```
;Programa que hace el parpadeo
;de un LED conectado al Puerto P1,
;en el pin P1.0.
        org 0000h
Inicio:
        clr P1.0        ;El LED se enciende
        lcall retardo   ;tiempo en que
                        ;se encuentra el LED encendido
        setb P1.0       ;El LED se apaga
        lcall retardo   ;tiempo en que se
                        ;encuentra el LED apagado
        sjmp Inicio     ;Se repite el parpadeo
                        ;indefinidamente, haciendo un salto a Inicio
;Empieza la subrutina de "retardo"
retardo:
        mov R0,#5       ;Muevo a R0 el número 5
loop:
        djnz R0,loop    ;Decrementa R0 y salta a loop si R0 no es cero
        ret             ;Retorna de la subrutina al programa principal
        END
```

En el EDSIM51:



Ejecución del programa:
Cuando el led está encendido.



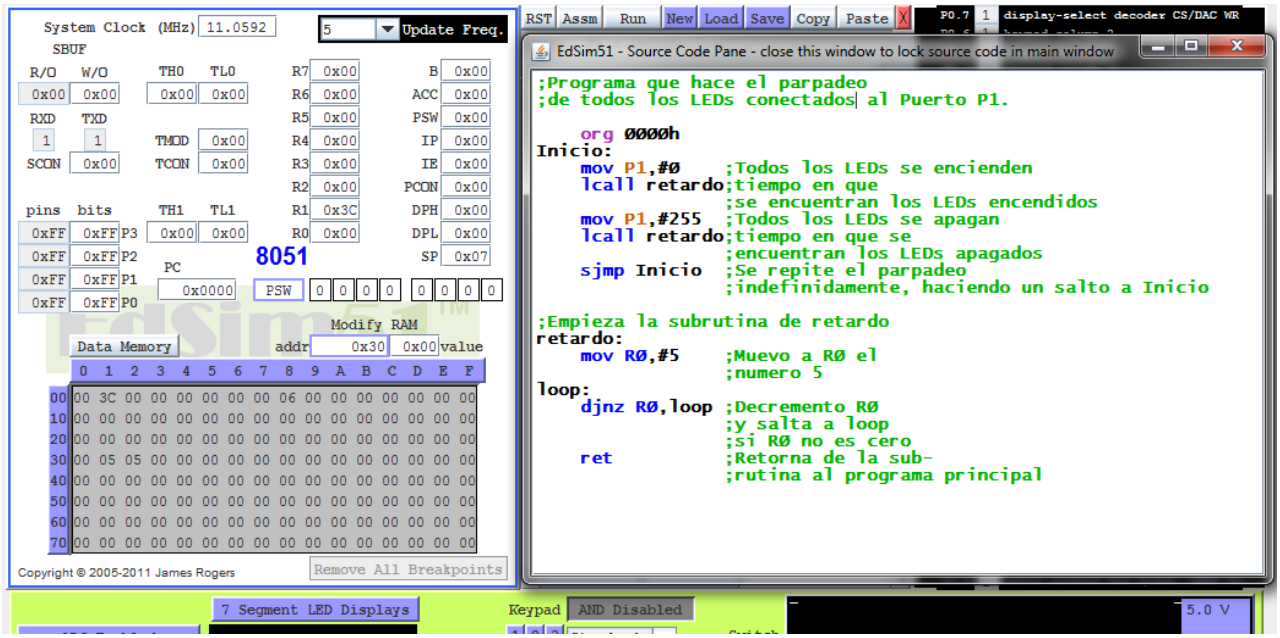
Cuando el led está apagado, se visualiza el led sin el color rojo.

Programa #8:
Programa que realiza el parpadeo de todos los leds conectados al Puerto P1.

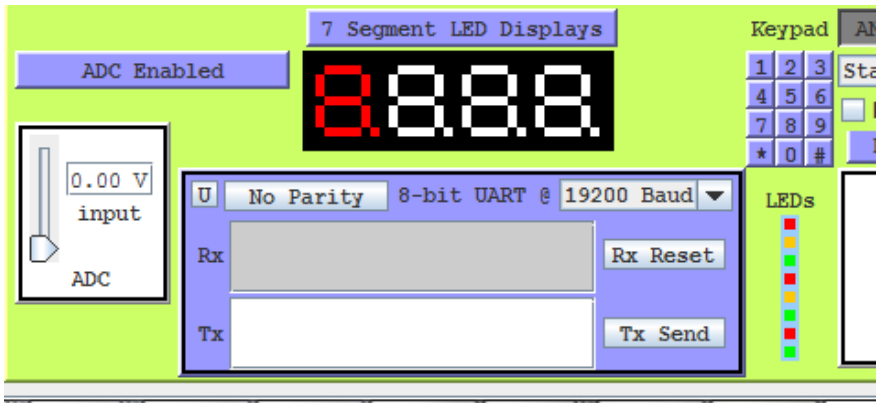
El listado del programa es el siguiente:

```
;Programa que hace el parpadeo de todos los LEDs conectados al Puerto P1.

        org 0000h
Inicio:
        mov P1,#0       ;Todos los LEDs se encienden
        lcall retardo   ;tiempo en que
                        ;se encuentran los LEDs encendidos
        mov P1,#255     ;Todos los LEDs se apagan
        lcall retardo   ;tiempo en que se encuentran los LEDs apagados
        sjmp Inicio     ;Se repite el parpadeo
                        ;indefinidamente, haciendo un salto a Inicio

;Empieza la subrutina de retardo
retardo:
        mov R0,#5       ;Muevo a R0 el número 5
loop:
        djnz R0,loop    ;Decremento R0 y salta a loop
                        ;si R0 no es cero
        ret             ;Retorna de la subrutina al programa principal
        END
```

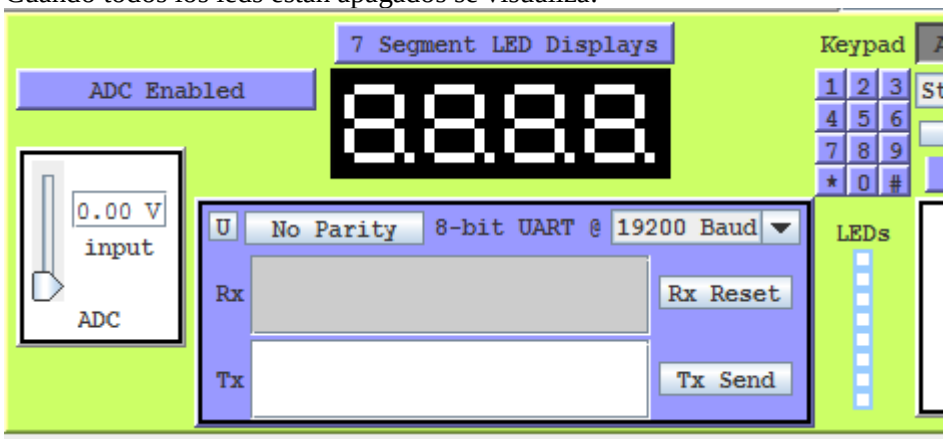
En el EDSIM51:



Ejecución:
Cuando todos los leds están encendidos se visualiza:



Cuando todos los leds están apagados se visualiza:

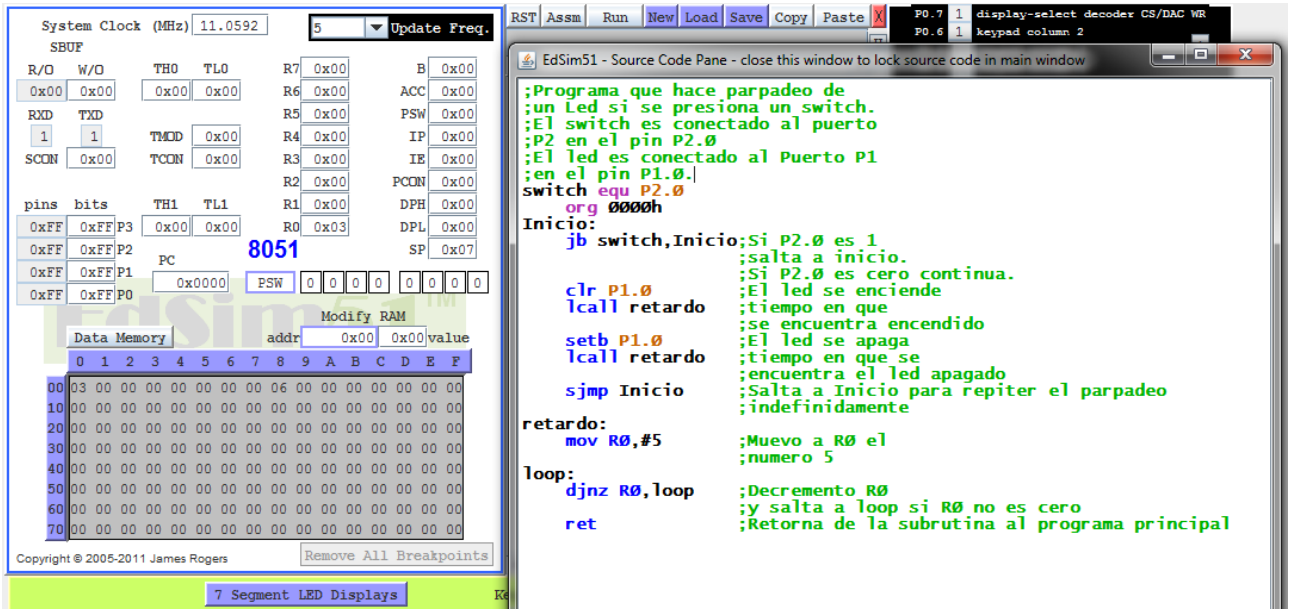


Ejercicio #6:
Realice un programa que realice el parpadeo de tres leds conectados a los pines P1.0, P1.1 y P1.2.

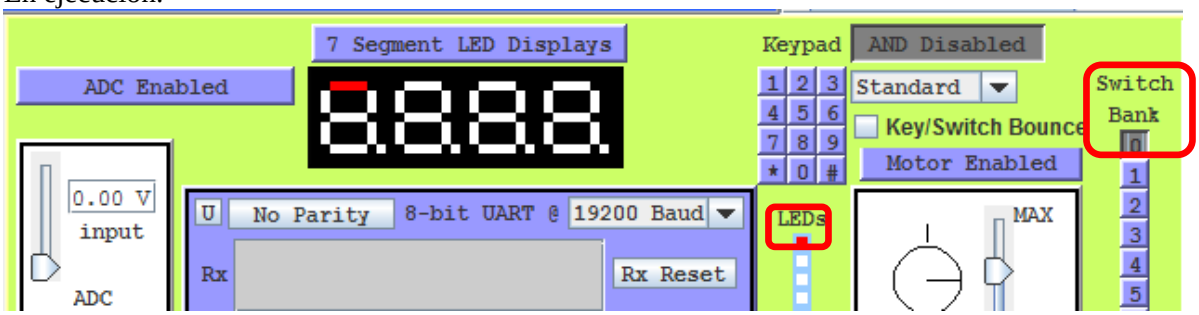
Programa #9:
Programa que hace el parpadeo de un led en caso se presiona un switch para pasar a una posición. Si se cambia a otra posición se detiene el parpadeo.
El listado del programa es el siguiente:

```
;Programa que hace parpadeo de
;un Led si se presiona un switch.
;El switch es conectado al puerto
;P2 en el pin P2.0
;El led es conectado al Puerto P1en el pin P1.0.
switch equ P2.0
        org 0000h
Inicio:
        jb switch,Inicio      ;Si P2.0 es 1 salta a inicio.
                              ;Si P2.0 es cero continua.
        clr P1.0              ;El led se enciende
        lcall retardo         ;tiempo en que se encuentra encendido
        setb P1.0             ;El led se apaga
        lcall retardo         ;tiempo en que se
                              ;encuentra el led apagado
        sjmp Inicio           ;Salta a Inicio para repiter el parpadeo
                              ;indefinidamente
retardo:
        mov R0,#5             ;Muevo a R0 el numero 5
loop:
        djnz R0,loop          ;Decremento R0 y salta a loop si R0 no es cero
        ret                   ;Retorna de la subrutina al programa principal
        END
```

En el EDSIM51:



En ejecución:

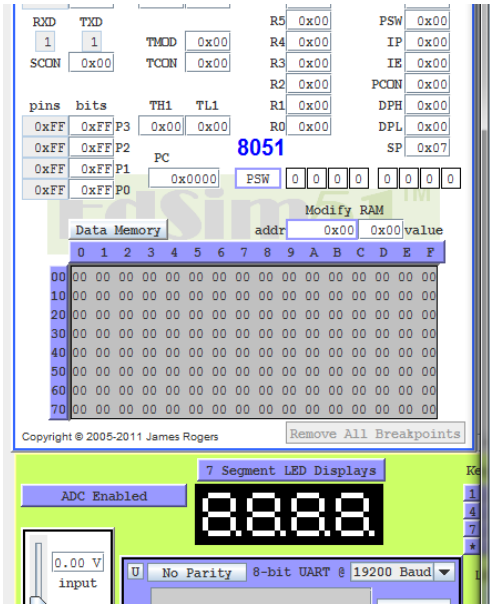


Programa #10:

Programa que a través de 2 switches se selecciona la frecuencia de parpadeo de un led. El listado es:

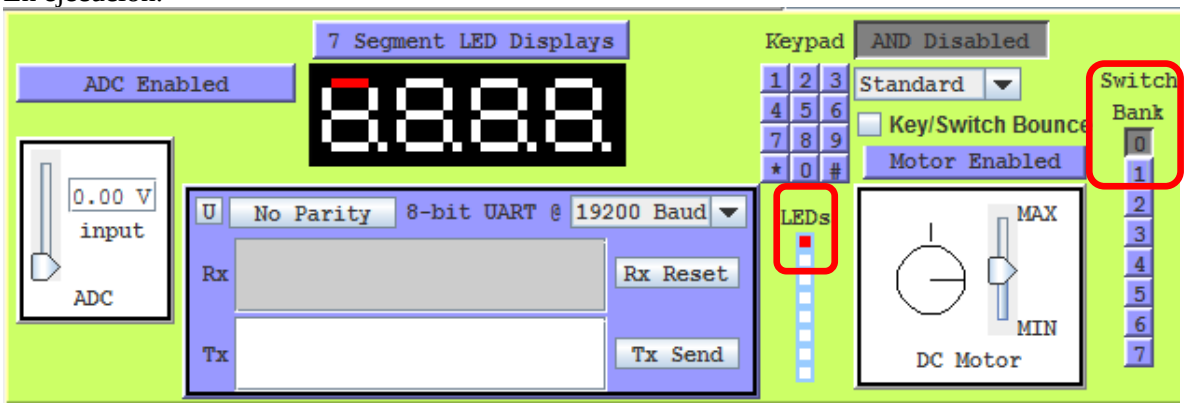
```
;Programa que utilizando dos switches
;selecciona la frecuencia de parpadeo de un Led.
;El primer switch es conectado al puerto P2 en el pin P2.0.
;El segundo switch es conectado al puerto P2 en el pin P2.1.
;El led va conectado al Puerto P1 en el pin P1.0
;Si ambos switches estan en 1 no selecciona nada, tampoco si ambos están en 0s.
        org 0000h
Inicio:
        jnb P2.0,prifrec        ;Si P2.0 es 0 salta a prifrec(primera frecuencia). Si es 1 continua.
        jnb P2.1,segfrec        ;Si es 0 va a segfrec(segunda frecuencia)
        sjmp Inicio             ;Si ambos son 1s va a Inicio
prifrec:
        jnb P2.1,Inicio         ;Regresa a Inicio si P2.1 es cero
        clr P1.0                ;el led se enciende
        lcall retardo           ;Tiempo en que se encuentra encendido
        setb P1.0               ;el led se apaga
        lcall retardo           ;Tiempo en que se encuentra el led apagado
        sjmp Inicio             ;Se repite el parpadeo indefinidamente
segfrec:
        jnb P2.0,Inicio         ;Regresa a Inicio si P2.0 es cero
        clr P1.0                ;el led se enciende
        lcall retardo2          ;Tiempo en que se encuentra encendido
        setb P1.0               ;el led se apaga
        lcall retardo2          ;Tiempo en que se encuentra el led apagado
        sjmp inicio             ;Se repite el parpadeo indefinidamente
retardo:
        mov R0,#6               ;Muevo a R0 el numero 6
loop1:
        djnz R0,loop1           ;Decremento R0 y salta a loop si R0 no es cero
        ret                     ;Retorna de la subrutina al programa principal
retardo2:
        mov R0,#50              ;Muevo a R0 el numero 50
loop:
        djnz R0,loop            ;Decremento R0 y salta a loop si R0 no es cero
        ret                     ;Retorna de la subrutina al programa principal
        END
```

En el EDSIM51:

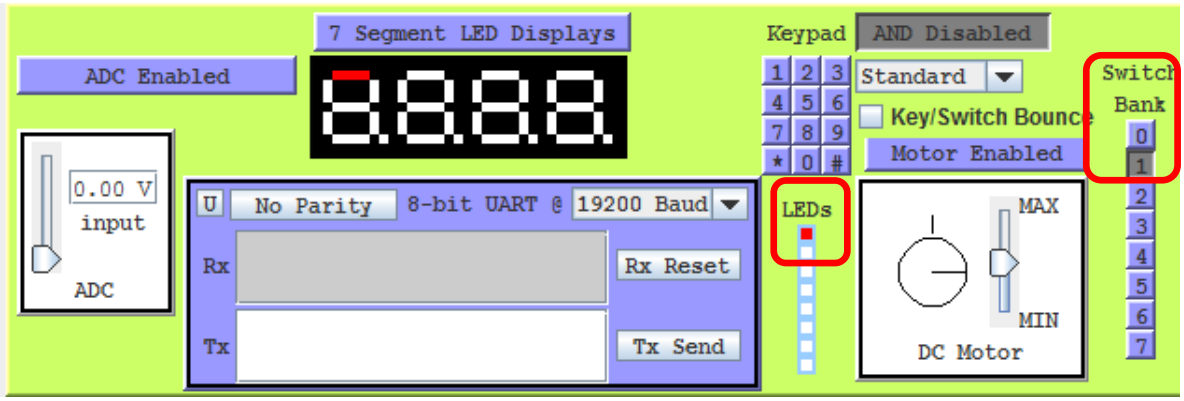



```
;El primer switch es conectado al puerto P2 en el pin P2.0.
;El segundo switch es conectado al puerto P2 en el pin P2.1.
;El led va conectado al Puerto P1 en el pin P1.0
;Si ambos switches estan en 1 no selecciona nada, tampoco si ambos están en 0s.
    org 0000h
Inicio:
    jnb P2.0,prifrec;Si P2.0 es 0 salta a prifrec(primera frecuencia)
                    ;Si es 1 continua.
    jnb P2.1,segfrec;Si es 0 va a segfrec(segunda frecuencia)
    sjmp Inicio     ;Si ambos son 1s va a Inicio
;Empieza la primera frecuencia
prifrec:
    jnb P2.1,Inicio ;Regresa a Inicio si P2.1 es cero
    clr P1.0        ;el led se enciende
    lcall retardo   ;Tiempo en que se encuentra encendido
    setb P1.0       ;el led se apaga
    lcall retardo   ;Tiempo en que se encuentra el led apagado
    sjmp Inicio     ;Se repite el parpadeo indefinidamente
;Empieza la segunda frecuencia
segfrec:
    jnb P2.0,Inicio ;Regresa a Inicio si P2.0 es cero
    clr P1.0        ;el led se enciende
    lcall retardo2  ;Tiempo en que se encuentra encendido
    setb P1.0       ;el led se apaga
    lcall retardo2  ;Tiempo en que se encuentra el led apagado
    sjmp inicio     ;Re repite el parpadeo indefinidamente
;tiempo que se da para la primera frecuencia
retardo:
    mov R0,#6       ;Muevo a R0 el numero 6
loop1:
    djnz R0,loop1   ;Decremento R0 y salta a loop si R0 no es cero
    ret             ;Retorna de la subrutina al programa principal
;tiempo que se da para la segunda frecuencia
retardo2:
    mov R0,#50      ;Muevo a R0 el numero 50
loop:
    djnz R0,loop    ;Decremento R0 y salta a loop si R0 no es cero
    ret             ;Retorna de la subrutina al programa principal
```

En ejecución:



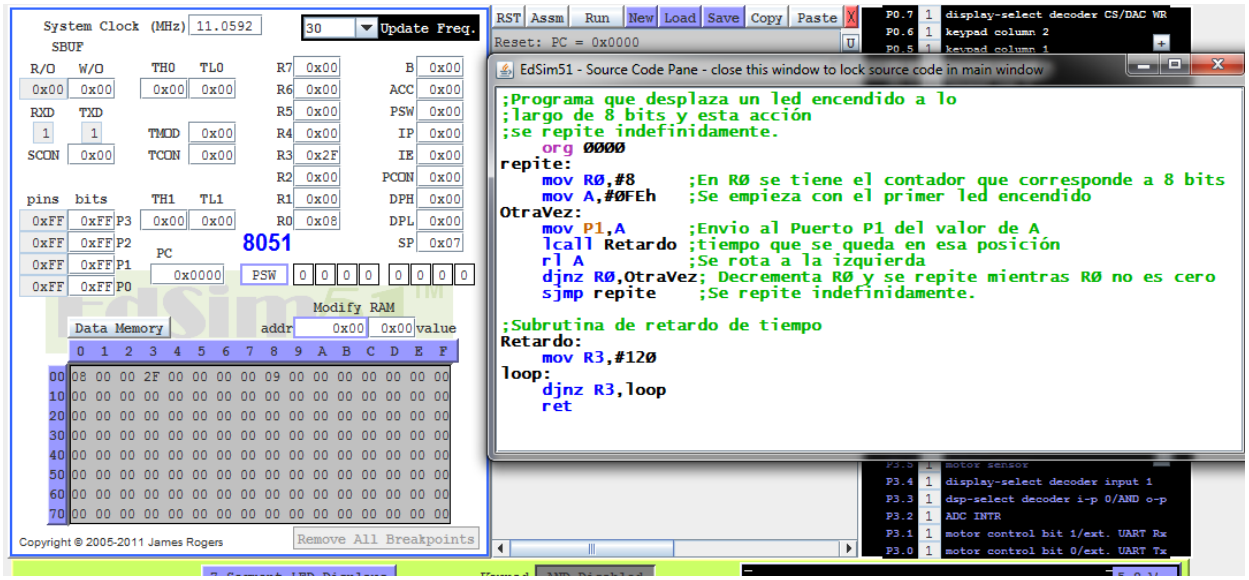
Selección de otra frecuencia:



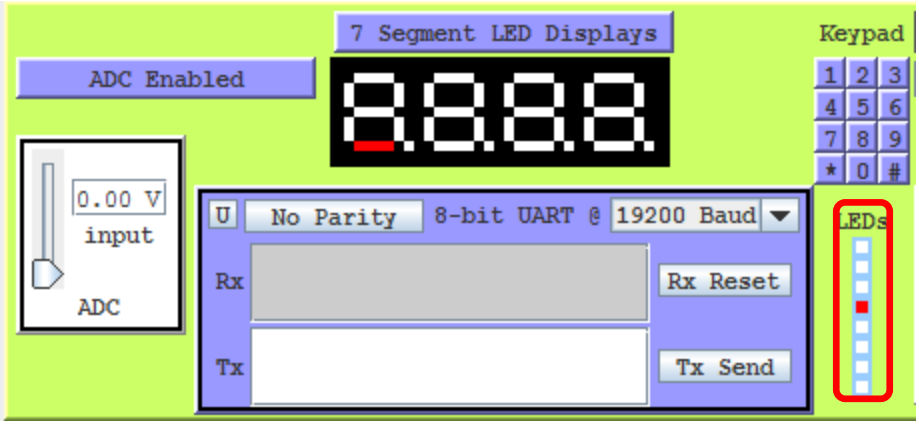
Programa #11:
Programa que realiza el desplazamiento de un led encendido de derecha a izquierda.
El listado del programa es:

```
;Programa que desplaza un led encendido a lo
;largo de 8 bits y esta acción se repite indefinidamente.
        org 0000h
repite:
        mov R0,#8           ;En R0 se tiene el contador que corresponde a 8 bits
        mov A,#0FEh         ;Se empieza con el primer led encendido
OtraVez:
        mov P1,A            ;Envío al Puerto P1 del valor de A
        lcall Retardo       ;tiempo que se queda en esa posición
        rl A                ;Se rota a la izquierda
        djnz R0,OtraVez     ;Decremento R0 y se repite mientras R0 no es cero
        sjmp repite         ;Se repite indefinidamente.

;Subrutina de retardo de tiempo
Retardo:
        mov R3,#120
loop:
        djnz R3,loop
        ret
        END
```

En el EDSIM51:



En ejecución:



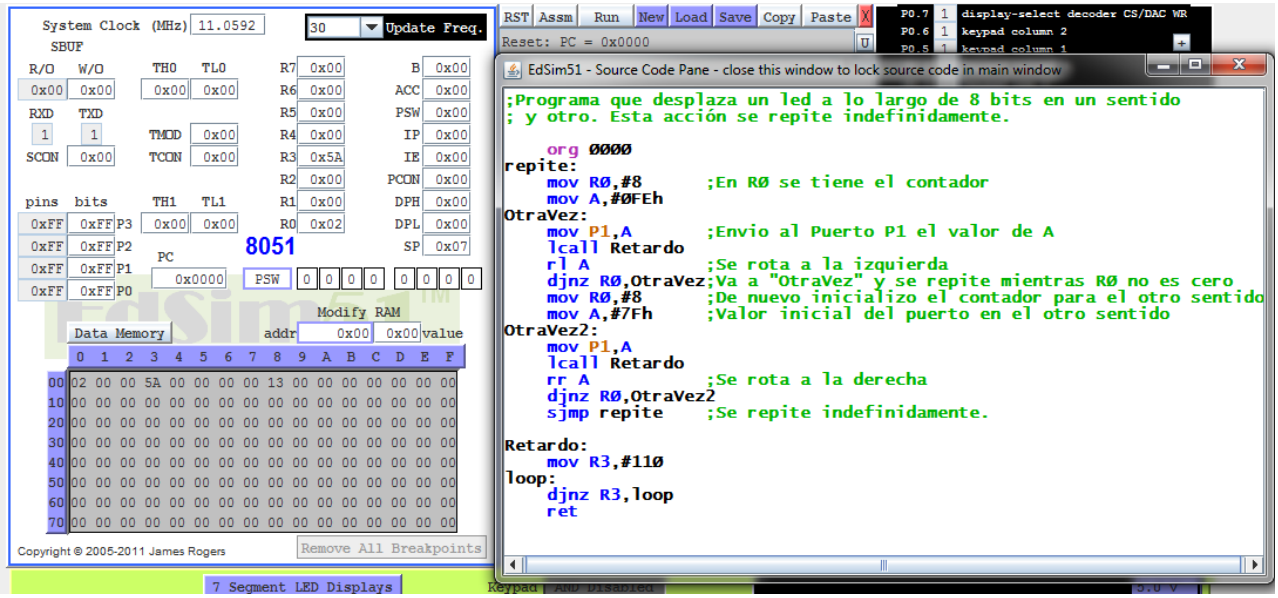
Ejercicio #7:
Desarrolle un programa que realice el desplazamiento de un led encendido de izquierda a derecha.
Sugerencia: Utilice rr A.

Ejercicio #8:
Desarrolle un programa que realice el desplazamiento de un led apagado de derecha a izquierda.

Programa #12:
Programa que desplaza un led encendido de un lado a otro en forma indefinida.
El listado del programa es:

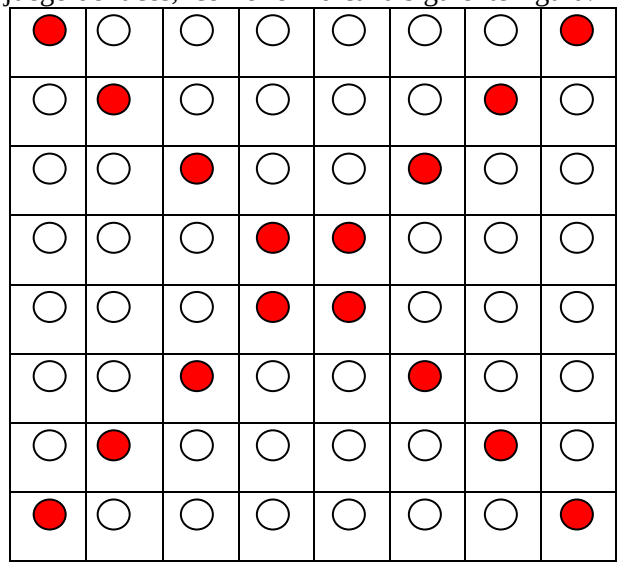
```
; Programa que desplaza un led a lo largo de 8 bits en un sentido y otro. Esta acción se repite.
        org 0000h
repite:
        mov R0,#8           ;En R0 se tiene el contador
        mov A,#0FEh
OtraVez:
        mov P1,A            ;envío al Puerto P1 el valor de A
        lcall Retardo
        rl A                ;Se rota a la izquierda
        djnz R0,OtraVez     ;Va a "OtraVez" y se repite mientras R0 no es cero
        mov R0,#8           ;De nuevo inicializo el contador para el otro sentido
        mov A,#7Fh          ;Valor inicial del puerto en el otro sentido
OtraVez2:
        mov P1,A
        lcall Retardo
        rr A                ;Se rota a la derecha
        djnz R0,OtraVez2
        sjmp repite         ;Se repite indefinidamente.
Retardo:
        mov R3,#110
loop:
        djnz R3,loop
        ret
        END
```

En el EDSIM51:



Ejercicio #9:
Desarrolle un programa que realice el desplazamiento de un led apagado de un sentido a otro.

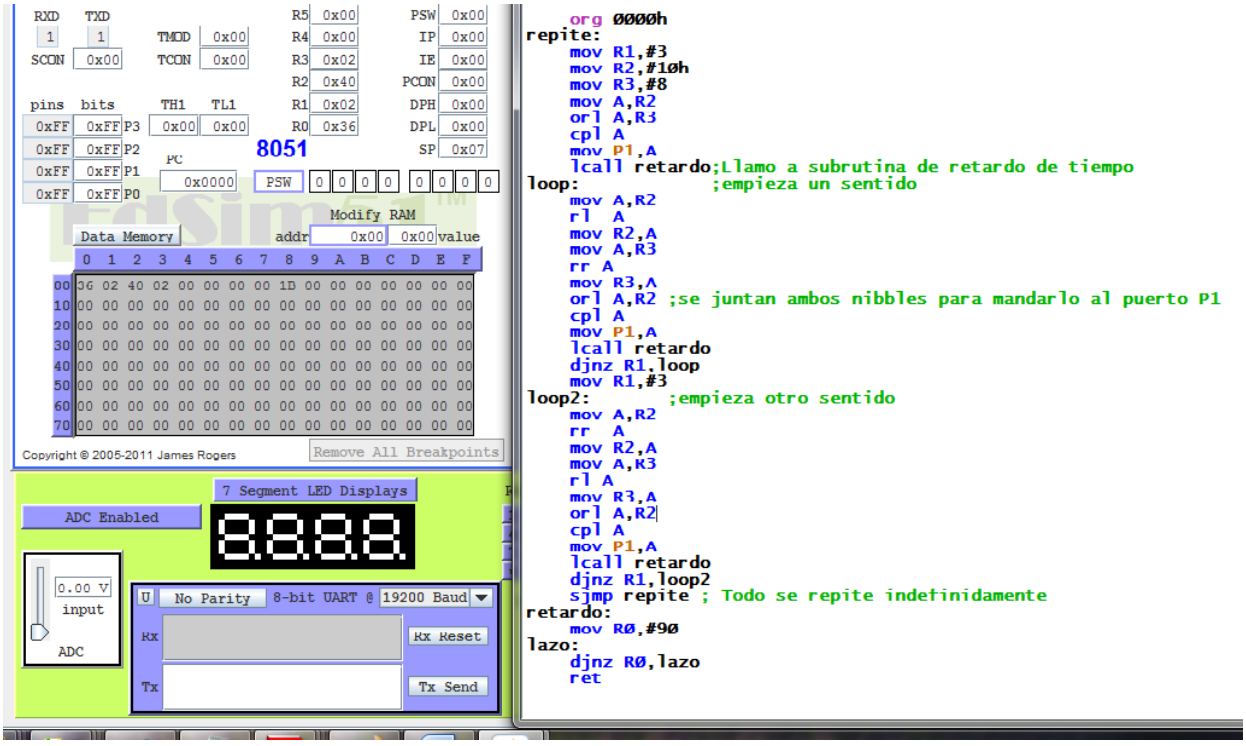
Programa #13:
Programa que realiza un juego de luces, como lo indica la siguiente figura:



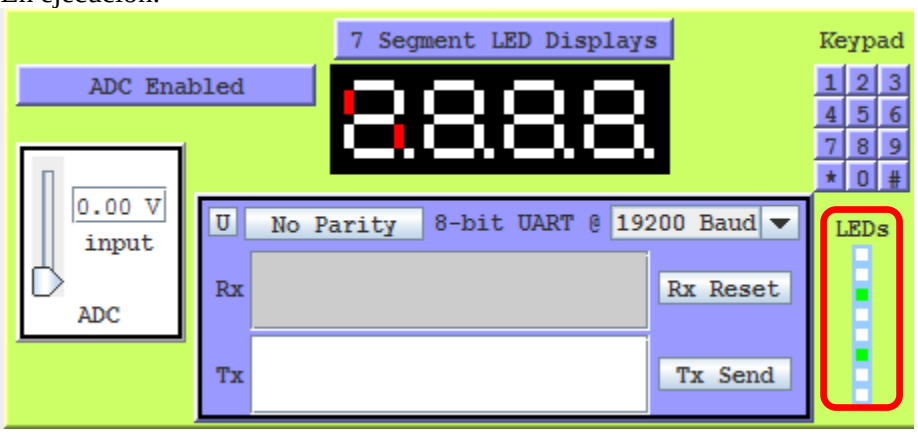
El listado del programa es:

```
;Programa de juego de luces
;Se usan las instrucciones de rotación a la izquierda
;y rotación a la derecha
;R1 es el contador, es decir el número de veces que se repite el loop.
;Se utilizan R2 o R3 para envío de los números a mostrar en los leds.

        org 0000h
repite:
        mov R1,#3
        mov R2,#10h
        mov R3,#8
        mov A,R2
        orl A,R3
        cpl A
        mov P1,A
        lcall retardo           ;Llamo a subrutina de retardo de tiempo
loop:                           ;Empieza un sentido
        mov A,R2
        rl      A
        mov R2,A
        mov A,R3
        rr A
        mov R3,A
        orl A,R2                ;Se juntan ambos nibbles para mandarlo al puerto P1
        cpl A
        mov P1,A
        lcall retardo
        djnz R1,loop
        mov R1,#3
loop2:                          ;Empieza otro sentido
        mov A,R2
        rr      A
        mov R2,A
        mov A,R3
        rl A
        mov R3,A
        orl A,R2
        cpl A
        mov P1,A
        lcall retardo
        djnz R1,loop2
        sjmp repite             ; Todo se repite indefinidamente
retardo:
        mov R0,#90
lazo:
        djnz R0,lazo
        ret
        END
```

En el EDSIM51:



En ejecución:



Ejercicio #10:
Hacer un programa que cuando presione un switch en P2.0 un led encendido se desplace de izquierda a derecha. Y cuando P2.1 es presionado un led encendido se desplace de derecha a izquierda.

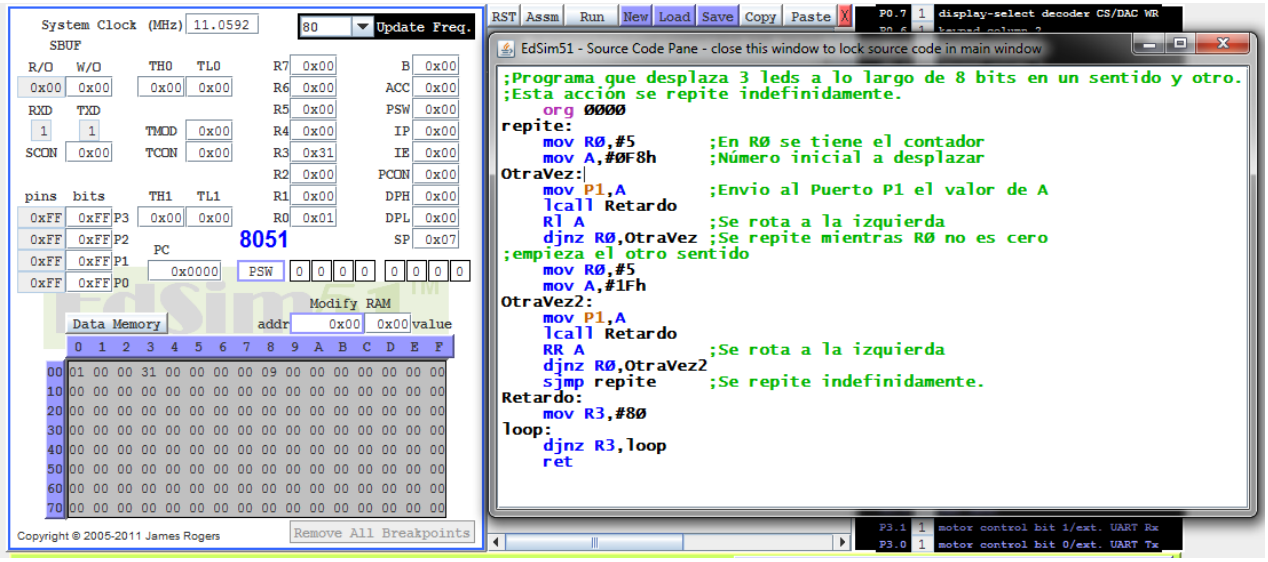
Ejercicio #11:
Hacer un programa que realice una barra de progreso de leds.

Programa #14:
Programa que desplaza 3 leds al mismo tiempo a lo largo del puerto P1 en un sentido y otro.

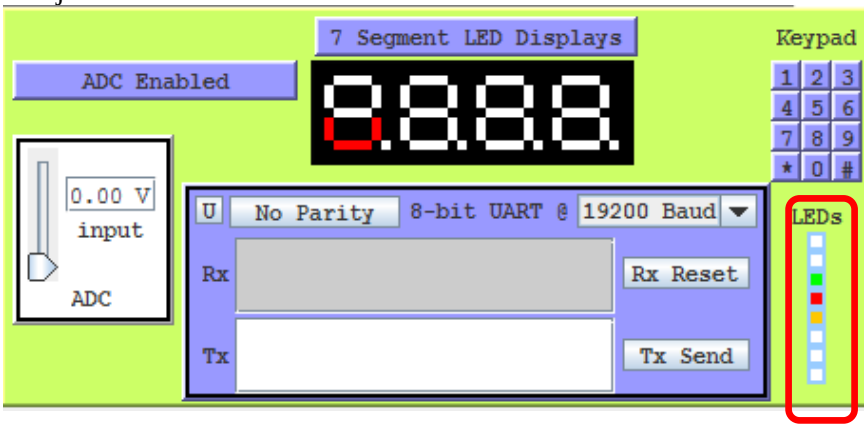
El listado del programa es:

```
; Programa que desplaza 3 leds a lo largo de 8 bits en un sentido y otro.
;Esta acción se repite indefinidamente.
        org 0000h
repite:
        mov R0,#5           ;En R0 se tiene el contador
        mov A,#0F8h         ;Número inicial a desplazar
OtraVez:
        mov P1,A            ;Envío al Puerto P1 el valor de A
        lcall Retardo
        Rl A                ;Se rota a la izquierda
        djnz R0,OtraVez     ;Se repite mientras R0 no es cero
;empieza el otro sentido
   mov R0,#5
        mov A,#1Fh
OtraVez2:
        mov P1,A
        lcall Retardo
        RR A                ;Se rota a la izquierda
        djnz R0,OtraVez2
        sjmp repite         ;Se repite indefinidamente todo el proceso.
Retardo:
        mov R3,#80
loop:
        djnz R3,loop
        ret
        END
```

En el EDSIM51:



En ejecución:



Ejercicio #12:

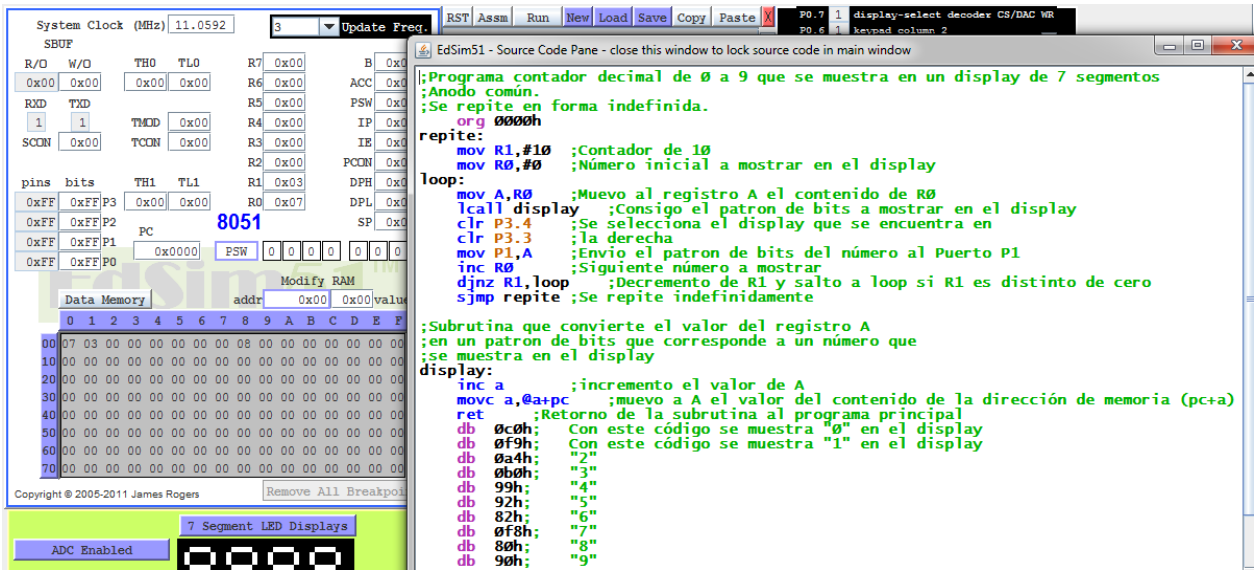
Desarrolle un programa que mediante 2 switches conectados a P2.0 y P2.1 seleccione el tipo de "juego de luces" a visualizar en los leds. Los juegos de luces a seleccionar son el programa #13 y el programa #14. Seleccione "100" veces la frecuencia del reloj en el simulador EDSIM51.

Programa #15:
Programa que muestra en un display de 7 segmentos un contador creciente de 0 a 9.
El listado del programa es:

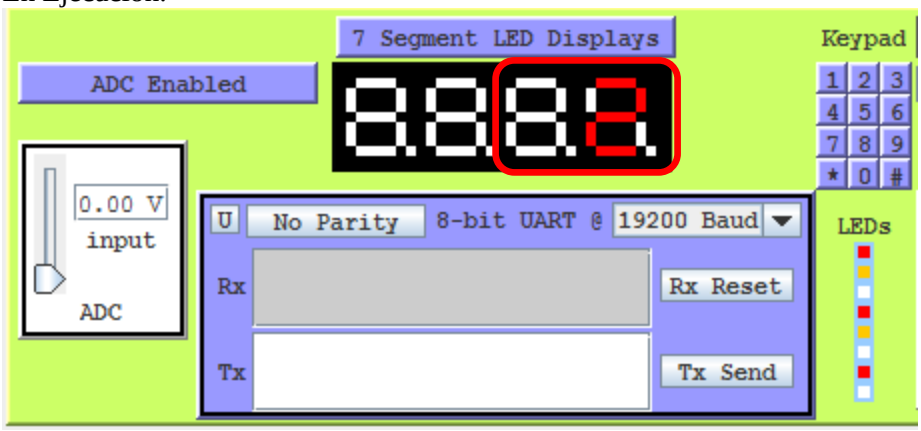
```
; Programa contador decimal de 0 a 9 que se muestra en un display de 7 segmentos ánodo común.
; Se repite en forma indefinida.
        org 0000h
repite:
        mov R1,#10      ;Contador de 10
        mov R0,#0       ;Número inicial a mostrar en el display
loop:
        mov A,R0        ;Muevo al registro A el contenido de R0
        lcall display   ;Consigo el patrón de bits a mostrar en el display
        clr P3.4        ;Se selecciona el display que se encuentra en
        clr P3.3        ;la derecha
        mov P1,A        ;Envío el patrón de bits del número al Puerto P1
        inc R0          ;Siguiente número a mostrar
        djnz R1,loop    ;Decremento de R1 y salto a loop si R1 es distinto de cero
        sjmp repite     ;Se repite indefinidamente

;Subrutina que convierte el valor del registro A
;en un patron de bits que corresponde a un número que
;se muestra en el display
display:
        inc a           ;incremento el valor de A
        movc a,@a+pc    ;muevo a A el valor del contenido de la dirección de memoria (pc+a)
        ret             ;Retorno de la subrutina al programa principal
        db      0c0h    ;Con este código se muestra "0" en el display
        db      0f9h    ;Con este código se muestra "1" en el display
        db      0a4h;   "2"
        db      0b0h;   "3"
        db      99h;    "4"
        db      92h;    "5"
        db      82h;    "6"
        db      0f8h;   "7"
        db      80h;    "8"
        db      90h;    "9"
        END
```

En el EDSIM51:

En el EDSIM51:



En Ejecución:

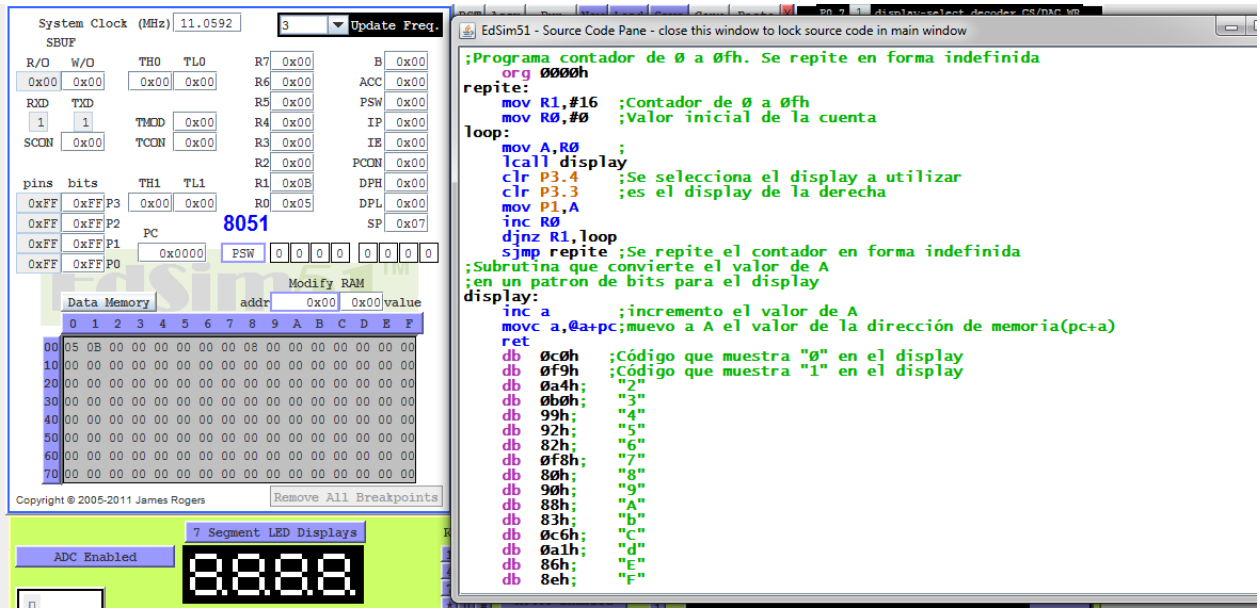


Ejercicio #13:
Desarrolle un programa de un contador decreciente de 9 a 0 y lo muestre en un display de 7 segmentos.

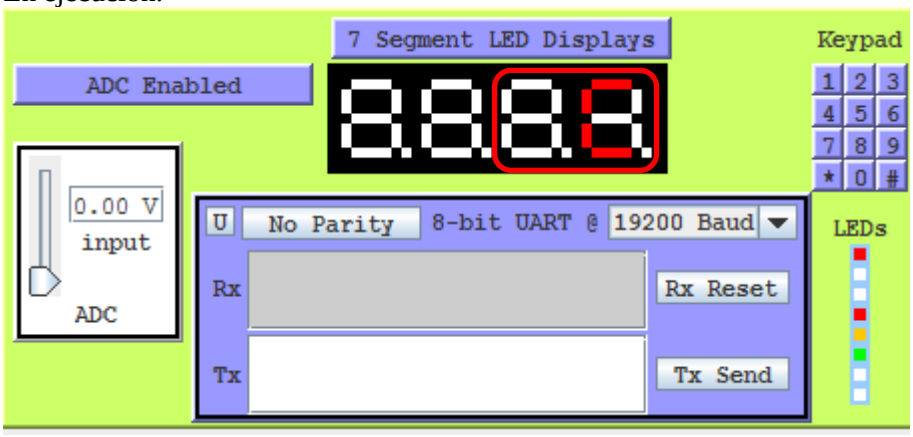
Programa #16:
Programa que realiza un contador de 0 a f y lo muestra en un display de 7 segmentos.
El listado del programa es:

```
; Programa contador de 0 a 0fh. Se repite en forma indefinida
        org 0000h
repite:
        mov R1,#16      ;Contador de 0 a 0fh
        mov R0,#0       ;Valor inicial de la cuenta
loop:
        mov A,R0        ;
        lcall display   ;Consigo el patron de bits a mover al Puerto P1
        clr P3.4        ;Se selecciona el display a utilizar
        clr P3.3        ;es el display de la derecha
        mov P1,A
        inc R0
        djnz R1,loop
        sjmp repite     ;Se repite el contador en forma indefinida
;Subrutina que convierte el valor de A
;en un patron de bits para el display
display:
        inc a           ;incremento el valor de A
        movc a,@a+pc    ;muevo a A el valor de la dirección de memoria (pc+a)
        ret
        db      0c0h    ;Código que muestra "0" en el display
        db      0f9h    ;Código que muestra "1" en el display
        db      0a4h;   "2"
        db      0b0h;   "3"
        db      99h;    "4"
        db      92h;    "5"
        db      82h;    "6"
        db      0f8h;   "7"
        db      80h;    "8"
        db      90h;    "9"
        db      88h;    "A"
        db      83h;    "b"
        db      0c6h;   "C"
        db      0a1h;   "d"
        db      86h;    "E"
        db      8eh;    "F"
        END
```

En el EDSIM51:



En ejecución:



Ejercicio #14:
Desarrolle un programa que al presionar el switch conectado al pin P2.0, empiece a contar el contador de 0 a f. Cuando se vuelve a presionar el switch que se detenga la cuenta.

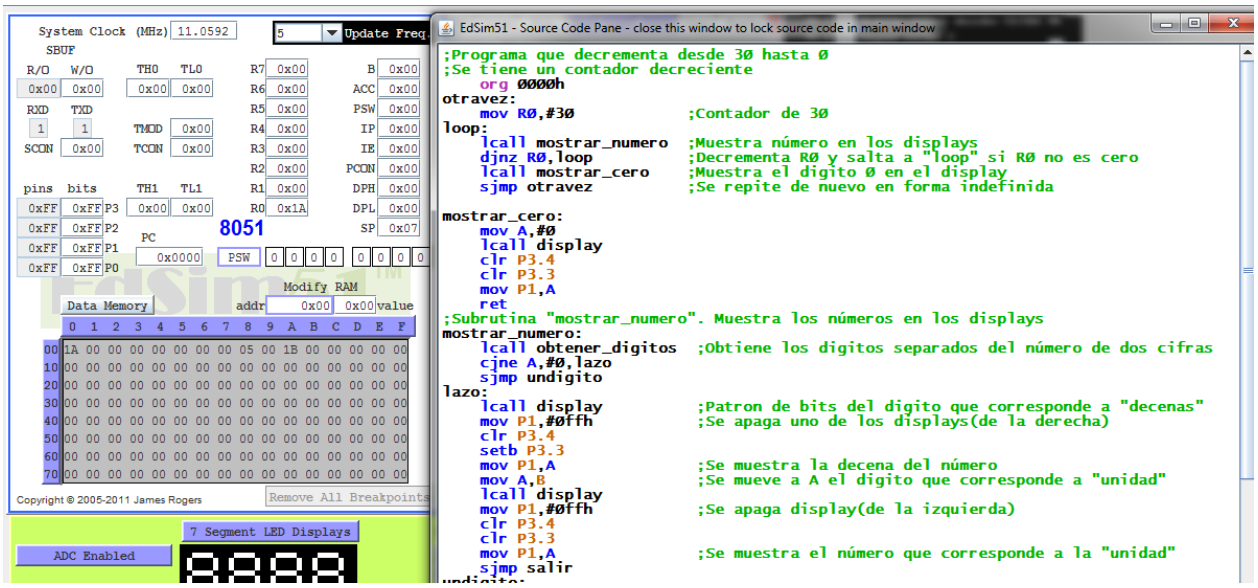
Ejercicio #15:
Programa que luego de cerrado un switch empiece el conteo descendente desde el número 7 hasta 0. Cuando llegue a 0 empiece el parpadeo de todos los leds del puerto P1.

Programa #17:
Programa de un contador decreciente desde 30 hasta 0, utilizando 2 displays de 7 segmentos.
El listado del programa es:

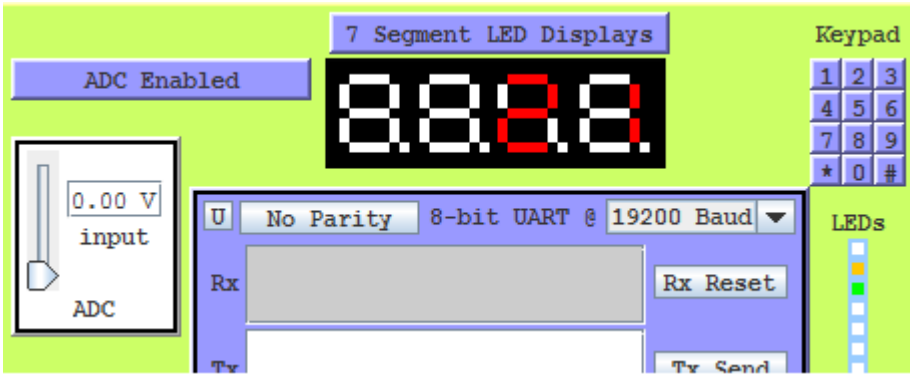
```
; Programa que decrementa desde 30 hasta 0. Se tiene un contador decreciente
        org 0000h
otravez:
        mov R0,#30              ;Contador de 30
loop:
        lcall mostrar_numero    ;Muestra número en los displays
        djnz R0,loop            ;Decrementa R0 y salta a "loop" si R0 no es cero
        lcall mostrar_cero      ;Muestra el dígito 0 en el display
        sjmp otravez            ;Se repite de nuevo en forma indefinida

mostrar_cero:
        mov A,#0
        lcall display
        clr P3.4
        clr P3.3
        mov P1,A
        ret
;Subrutina "mostrar_numero". Muestra los números en los displays
mostrar_numero:
        lcall obtener_digitos   ;Obtiene los dígitos separados del número de dos cifras
        cjne A,#0,lazo
        sjmp undigito
lazo:
        lcall display           ;Patrón de bits del dígito que corresponde a "decenas"
        mov P1,#0ffh            ;Se apaga uno de los displays(de la derecha)
        clr P3.4
        setb P3.3
        mov P1,A                ;Se muestra la decena del número
        mov A,B                 ;Se mueve a A el dígito que corresponde a "unidad"
        lcall display
        mov P1,#0ffh            ;Se apaga display(de la izquierda)
        clr P3.4
        clr P3.3
        mov P1,A                ;Se muestra el número que corresponde a la "unidad"
        sjmp salir
undigito:
        mov A,B                 ;Se visualiza en el display sólo el dígito de "unidad"
        lcall display
        clr P3.4
        clr P3.3
        mov P1,A
salir:
        ret
```

```
; Subrutina “obtener_digitos”. Obtiene los dígitos del número de 2 cifras
;En A se guarda la "decena" y en B se guarda la "unidad"
obtener_digitos:
        mov A,R0
        mov B,#10
        div AB
        ret

;Subrutina que convierte el valor de A
;en un patrón de bits para enviar al display
display:
        inc a               ;Incremento el valor de A
        movc a,@a+pc        ;muevo a A el valor de la dirección de memoria(pc+a)
        ret
        db      0c0h        ;Con este código se muestra "0" en el display
        db      0f9h        ;Con este código se muestra "1" en el display
        db      0a4h        ;se muestra "2"
        db      0b0h        ;se muestra "3"
        db      99h         ;"4"
        db      92h;        "5"
        db      82h;        "6"
        db      0f8h;       "7"
        db      80h;        "8"
        db      90h;        "9"
        END
```

En el EDSIM51:



En ejecución:



Ejercicio #16:
Desarrolle un contador decreciente de tres dígitos. Que decremente desde 126 hasta 0.

Programa #18:
Programa que genera la secuencia de Fibonacci para (n=12). Esta secuencia se visualiza uno por uno en los displays.
Se aplicará para el desarrollo del programa el siguiente algoritmo:

INICIO
si(n<= 2) retornar 1;
a = 1, b = 1, c, i;
para(i = 3; i < n ; i++)
{
c = a + b;
a = b; b = c;
}
retornar c;
FIN

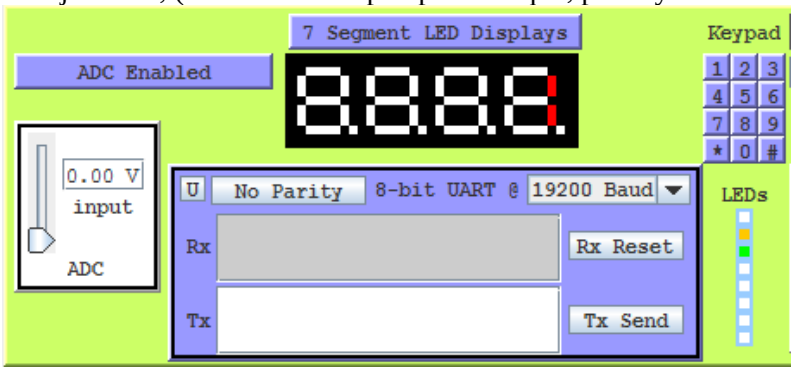
El listado del programa es:

```
;Programa que visualiza en el display
;la serie de fibonacci: 1,1,2,3,5,8,13,21,34,55,89,144
;para (n=12).
centena equ 30h
decena  equ 31h
unidad  equ 32h

        org 0000h
        lcall serie             ;Se encarga de generar la serie para n=12
otravez:
        mov R2,#12
        mov R0,#50h             ;la serie se almacenó a partir de la dirección de memoria 50h
loop:
        mov A,@R0               ;empiezo a mover a A los valores de la serie generada
        lcall obtiene_digitos   ;se obtiene los dígitos(centena, decena y unidad) a visualizar en los
displays
        cjne A,#0,muestradig
        sjmp undigito           ;muestra sólo un display(el que está mas a la derecha)
```

```
        ;Se muestra los digitos en los displays
muestradig:
        mov A,centena
        lcall display
   mov P1,#0FFh
        setb P3.4                   ;a partir de aqui se selecciona el display que va mostrar la "centena"
        clr P3.3
        mov P1,A
        mov A,decena
        lcall display
   mov P1,#0FFh
        clr P3.4                    ;a partir de aqui se selecciona el display que va mostrar la "decena"
        setb P3.3
        mov P1,A
undigito:
        mov A,unidad
        lcall display
        mov P1,#0FFh
        clr P3.4                    ;a partir de aqui se selecciona el display que va mostrar la "unidad"
        clr P3.3
        mov P1,A
        inc R0
        lcall retardo               ;tiempo que demora en cambiar el digito
        djnz R2,loop
        sjmp otravez                ;se repite indefinidamente

retardo:
        mov R1,#11
        djnz R1,$
        ret
;Subrutina que genera la serie de Fibonacci
serie:
        mov R0,#10      ;Contador, que indica el número de veces que se repite el bucle.
        mov R1,#52h     ;dirección de memoria que contiene el tercer número de fibonacci
        mov R2,#3       ;Se empieza con n=3, ya que para n=1 y n=2 se conocen los números
        mov R4,#1       ;R4 va ser "a" de acuerdo con el algoritmo
        mov R5,#1       ;R5 va ser "b" de acuerdo con el algoritmo
        mov 50h,R4
        mov 51h,R5
repite:
        mov A,R4
        add A,R5        ;Se suma a + b = c
        mov @R1,A       ;el valor de la suma "c" se guarda en la dirección de memoria
        mov A,R5        ;como no se puede hacer mov R4,R5 entonces uso el registro "A" como intermediario
        mov R4,A        ;el nuevo valor de "a" es ahora "b"
        mov A,@R1
        mov R5,A        ;el nuevo valor de "b" es ahora "c"
        inc R2          ;un nuevo valor de "n",
        inc R1          ;una nueva posición de memoria
```

```
        djnz R0,repite
        ret

;Subrutina que obtiene los digitos a mostrar en el display
obtiene_digitos:
        mov B,#100          ;Muevo el valor 100 a B
        div AB                 ;División de A entre B y el resultado se guarda en A
        mov centena,A       ;Muevo el valor de A a 30h
        mov A,B                ;el residuo se guarda en el acumulador
        mov B,#10          ;muevo 10 a B
        div AB                 ;divido A entre B y el resultado se guarda en A
        mov decena,A        ;muevo el valor de A a 31h
        mov unidad,B        ;se guarda el valor de B en 32h
        ret

;Subrutina que obtiene el patrón de bits a mostrar en el display
Display:
        inc A
        movc A,@A+PC
        ret
        db      0c0h;    se muestra "0" en el display
        db      0f9h;    se muestra "1" en el display
        db      0a4h;    "2"
        db      0b0h;    "3"
        db      99h;     "4"
        db      92h;     "5"
        db      82h;     "6"
        db      0f8h;    "7"
        db      80h;     "8"
        db      90h;     "9"
   END
```

En Ejecución, (hacer en el campo update freq: 6, para System clock: 12)



.....
.....
.....

Programa #19:
Programa que calcula el factorial de un número y lo muestra en los displays. Por ejemplo 5!.
Se aplica el siguiente algoritmo:

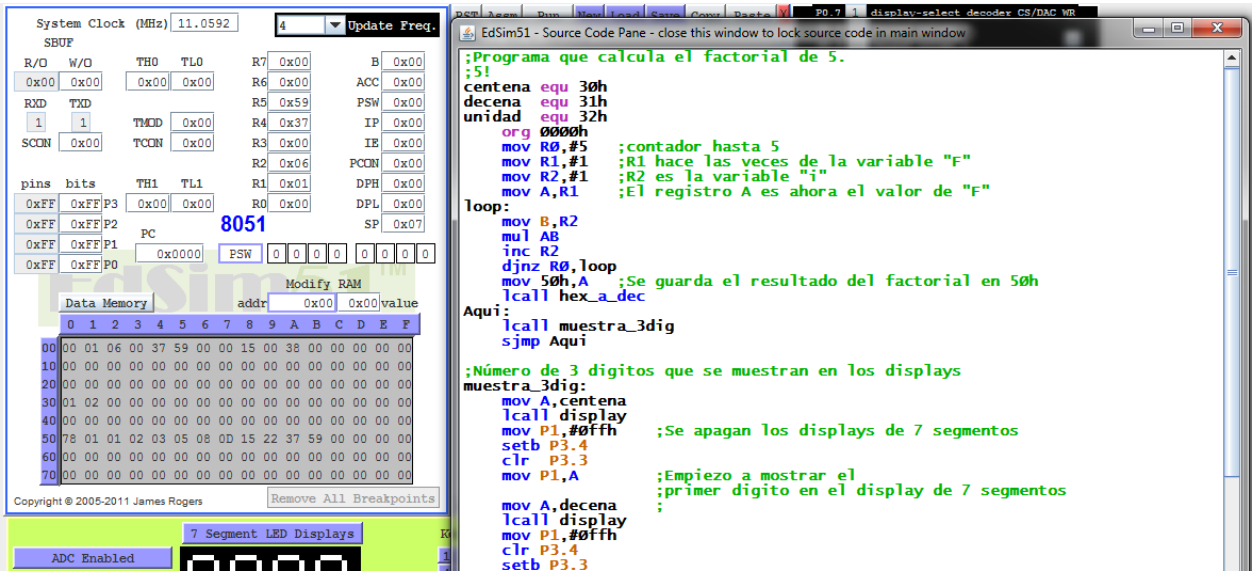
INICIO
F=1, i=1
Mientras(i<=n)
{
F=Fxi
i=i+1
}
return F
FIN

El listado del programa es:

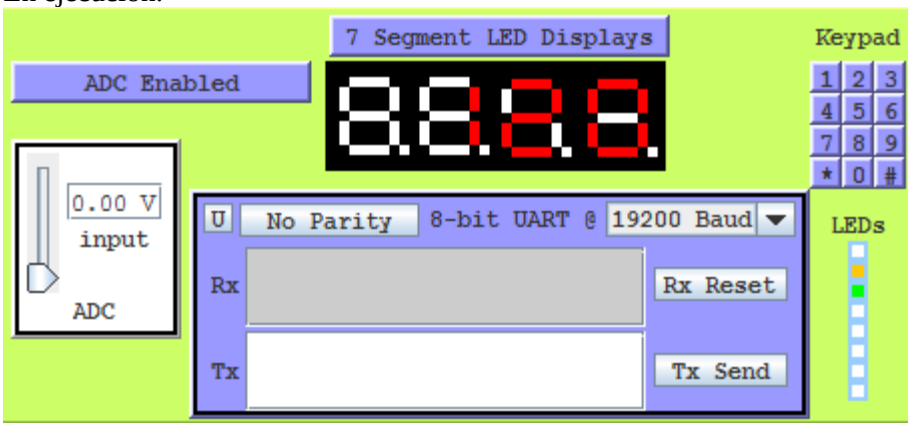
```
;Programa que calcula el factorial de 5.
;5!
centena equ 30h
decena  equ 31h
unidad  equ 32h
        org 0000h
        mov R0,#5          ;Contador hasta 5
        mov R1,#1          ;R1 hace las veces de la variable "F"
        mov R2,#1          ;R2 es la variable "i"
        mov A,R1           ;El registro A es ahora el valor de "F"
loop:
        mov B,R2
        mul AB
        inc R2
        djnz R0,loop
        mov 50h,A           ;Se guarda el resultado del factorial en 50h
        lcall hex_a_dec
Aqui:
        lcall muestra_3dig  ;Subrutina que muestra 3 dígitos en los displays
        sjmp Aqui

;Número de 3 dígitos que se muestran en los displays
muestra_3dig:
        mov A,centena
        lcall display
        mov P1,#0ffh       ;Se apagan los displays de 7 segmentos
        setb P3.4
        clr  P3.3
        mov P1,A           ;Empiezo a mostrar el
                           ;primer dígito en el display de 7 segmentos
```

```
        mov A,decena    ;
        lcall display
        mov P1,#0ffh
        clr P3.4
        setb P3.3
        mov P1,A          ;Muestro el segundo dígito
        mov A,unidad
        lcall display
        mov P1,#0ffh      ;
        clr P3.4
        clr P3.3
        mov P1,A          ;Muestro el tercer dígito en el display de 7 segmentos
        ret

;Subrutina que obtiene tres dígitos en decimal
hex_a_dec:
        mov B,#100        ;Muevo el valor 100 a B
        div AB            ;División de A entre B y el resultado se guarda en A
        mov centena,A     ;Muevo el valor de A a 30h
        mov A,B           ;el residuo se guarda en el acumulador
        mov B,#10         ;muevo 10 a B
        div AB            ;divido A entre B y el resultado se guarda en A
        mov decena,A      ;muevo el valor de A a 31h
        mov unidad,B      ;se guarda el valor de B en 32h
        ret
Display:
        inc A
        movc A,@A+PC
        ret
        db      0C0h    ;Con este patrón se muestra "0" en el display de 7 segmentos
        db      0f9h    ;se muestra "1"
        db      0a4h;   se muestra "2"
        db      0b0h;   "3"
        db      99h;    "4"
        db      92h;    "5"
        db      82h;    "6"
        db      0f8h;   "7"
        db      80h;    "8"
        db      90h;    "9"
        END
```

En el EDSIM51:



En ejecución:

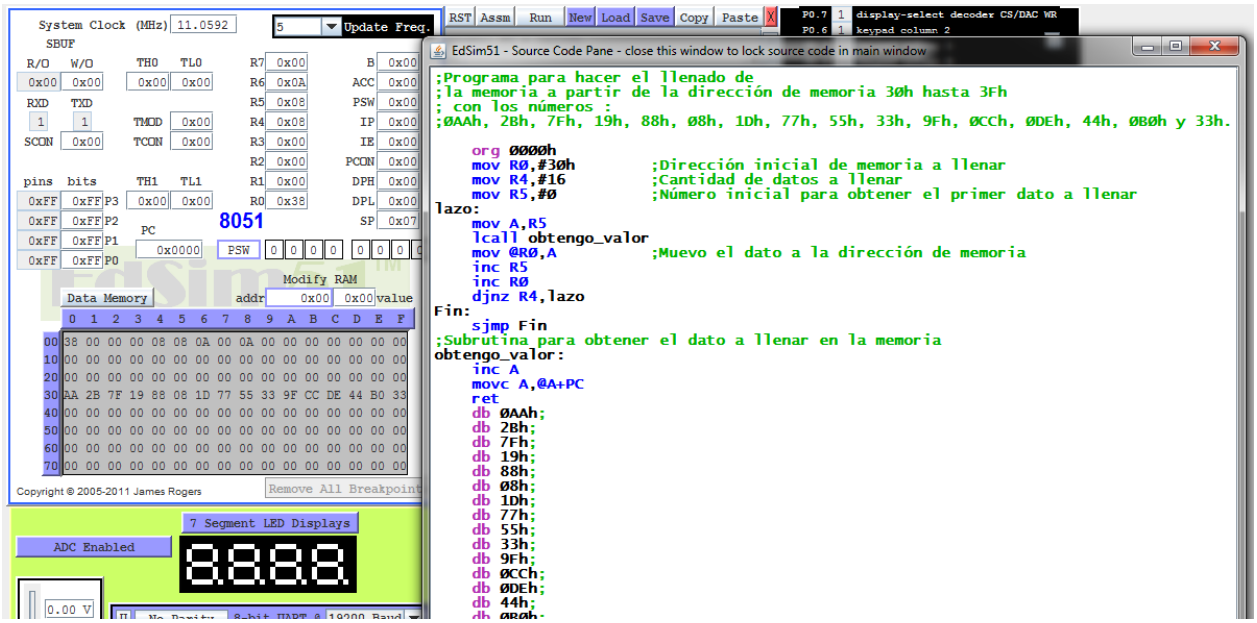


Nota: En el simulador se observa que se enciende un dígito a la vez del número de tres dígitos, entonces se encienden en forma secuencial. Aquí se muestran los tres dígitos encendidos a la vez sólo por ilustración. En el mundo real los tres dígitos se visualizaran como si estuvieran todos encendidos a la vez.

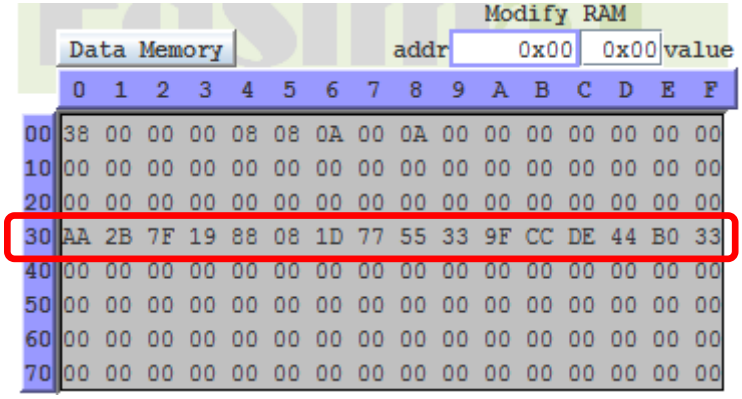
Programa #20:
Programa para llenar a partir de 30h hasta 3fh con un listado de números por ejemplo: 0AAh, 2Bh, 7Fh, 19h, 88h, 08h, 1Dh, 77h, 55h, 33h, 9Fh, 0CCh, 0DEh, 44h, 0B0h y 33h.
El listado del programa es:

```
;Programa para hacer el llenado de
;la memoria a partir de la dirección de memoria 30h hasta 3Fh
; con los números :
;0AAh, 2Bh, 7Fh, 19h, 88h, 08h, 1Dh, 77h, 55h, 33h, 9Fh, 0CCh, 0DEh, 44h, 0B0h y 33h.

        org 0000h
        mov R0,#30h         ;Dirección inicial de memoria a llenar
        mov R4,#16          ;Cantidad de datos a llenar
        mov R5,#0           ;Número inicial para obtener el primer dato a llenar
lazo:
        mov A,R5
        lcall obtengo_valor
        mov @R0,A           ;Muevo el dato a la dirección de memoria
        inc R5
        inc R0
        djnz R4,lazo
Fin:
        sjmp Fin
;Subrutina para obtener el dato a llenar en la memoria
obtengo_valor:
        inc A
        movc A,@A+PC
        ret
        db 0AAh;
        db 2Bh;
        db 7Fh;
        db 19h;
        db 88h;
        db 08h;
        db 1Dh;
        db 77h;
        db 55h;
        db 33h;
        db 9Fh;
        db 0CCh;
        db 0DEh;
        db 44h;
        db 0B0h;
        db 33h;
         END
```

En el EDSIM51:


EdSim51 - Source Code Pane - close this window to lock source code in main window
;Programa para hacer el llenado de
;la memoria a partir de la dirección de memoria 30h hasta 3Fh
; con los números :
;ØAAh, 2Bh, 7Fh, 19h, 88h, Ø8h, 1Dh, 77h, 55h, 33h, 9Fh, ØCCh, ØDEh, 44h, ØB0h y 33h.
org ØØØØh
mov RØ,#3Øh ;Dirección inicial de memoria a llenar
mov R4,#16 ;Cantidad de datos a llenar
mov R5,#Ø ;Número inicial para obtener el primer dato a llenar
lazo:
mov A,R5
lcall obtengo_valor
mov @RØ,A ;Muevo el dato a la dirección de memoria
inc R5
inc RØ
djnz R4,lazo
Fin:
sjmp Fin
;Subrutina para obtener el dato a llenar en la memoria
obtengo_valor:
inc A
movc A,@A+PC
ret
db ØAAh;
db 2Bh;
db 7Fh;
db 19h;
db 88h;
db Ø8h;
db 1Dh;
db 77h;
db 55h;
db 33h;
db 9Fh;
db ØCCh;
db ØDEh;
db 44h;
8051
Modify RAM
Data Memory
addr 0x00 0x00 value
7 Segment LED Displays
ADC Enabled

Resultado:


Modify RAM
Data Memory
addr 0x00 0x00 value
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 38 00 00 00 08 08 0A 00 0A 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 AA 2B 7F 19 88 08 1D 77 55 33 9F CC DE 44 B0 33
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Programa #21:
Programa que encuentra el mayor número de un listado de números. Se usará el listado de números del programa #20. Para "N=16".
Se aplicará el algoritmo siguiente:

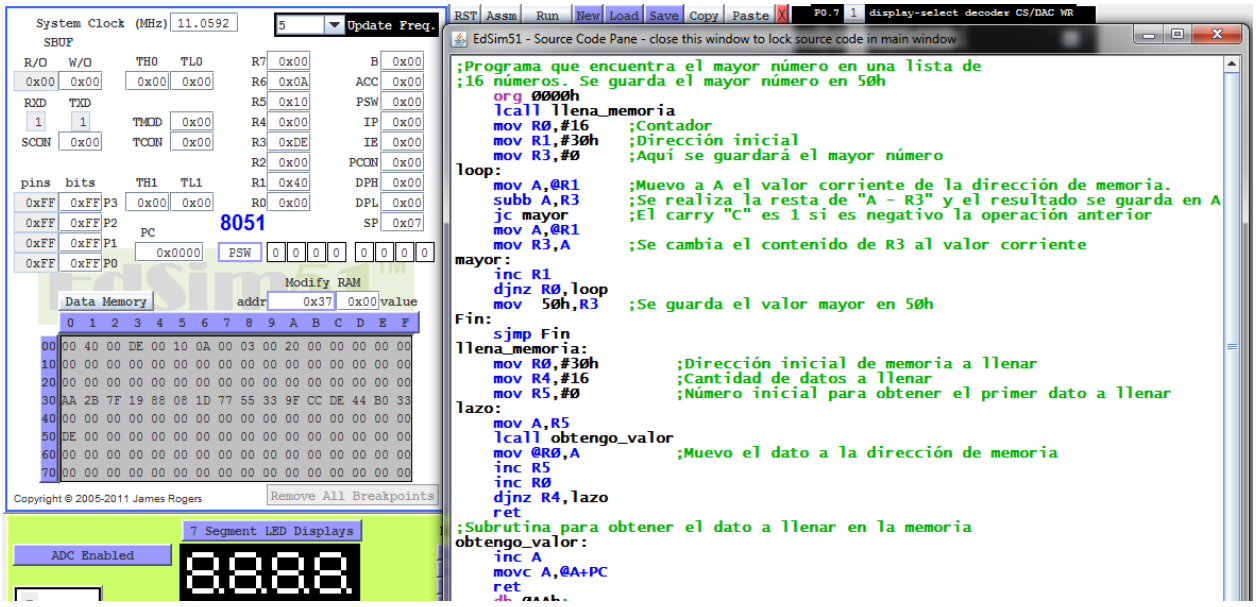
INICIO
Enteros Mayor, Contador, ElementoCorriente, N, listado(N)
Mayor ← -∞
Contador ← 0
MIENTRAS (Contador < = N)
{
ElementoCorriente ← listado(Contador)
SI (ElementoCorriente > Mayor)
{
Mayor ← ElementoCorriente
}
Contador ← Contador + 1
}
Retorna Mayor
FIN

El listado del programa es:

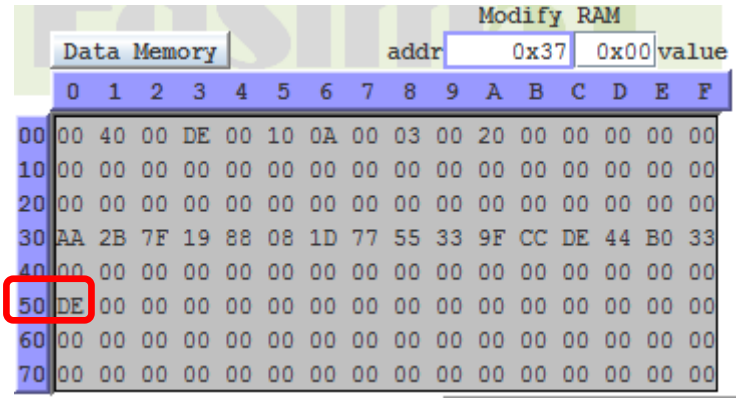
```
;Programa que encuentra el mayor número en una lista de
;16 números. Se guarda el mayor número en 50h
        org 0000h
        lcall llena_memoria
        mov R0,#16      ;En este caso "R0" va a tomar como valor inicial "N=16"
        mov R1,#30h     ;Dirección inicial de memoria en donde empieza el listado
        mov R3,#0       ;R3 hará las veces de "Mayor" y guardará el mayor número
loop:
        mov A,@R1       ;Muevo a "A" el elemento corriente
        subb A,R3       ;Se realiza la resta de "A - R3" y el resultado se guarda en A
        jc mayor        ;El carry "C" es 1 si es negativo la operación anterior
        mov A,@R1       ;Muevo a "A" el elemento corriente
        mov R3,A        ;Se cambia el contenido de R3 con el  elemento corriente
mayor:
        inc R1          ;Siguiente posición de memoria
        djnz R0,loop
        mov  50h,R3     ;Se guarda el mayor número en 50h
Fin:
        sjmp Fin
llena_memoria:
        mov R0,#30h     ;Dirección inicial de memoria a llenar
        mov R4,#16      ;Cantidad de datos a llenar
        mov R5,#0       ;Número inicial para obtener el primer dato a llenar
lazo:
        mov A,R5
        lcall obtengo_valor
        mov @R0,A       ;Muevo el dato a la dirección de memoria
```

```
        inc R5
        inc R0
        djnz R4,lazo
        ret
;Subrutina para obtener el dato a llenar en la memoria
obtengo_valor:
        inc A
        movc A,@A+PC
        ret
        db 0AAh;
        db 2Bh;
        db 7Fh;
        db 19h;
        db 88h;
        db 08h;
        db 1Dh;
        db 77h;
        db 55h;
        db 33h;
        db 9Fh;
        db 0CCh;
        db 0DEh;
        db 44h;
        db 0B0h;
        db 33h;
        END
```

En el EDSIM51:



Resultado:



Ejercicio #17:
Desarrolle un programa que encuentra el menor número de un listado de números. Se deberá utilizar el listado de números del programa #20.
Aplicar el siguiente algoritmo:

INICIO
Enteros Menor, Contador, ElementoCorriente, N, listado(N)
Menor ← ∞
Contador ← 0
MIENTRAS (Contador < = N)
{
ElementoCorriente ← listado(Contador)
SI (ElementoCorriente < Menor)
{
Menor ← ElementoCorriente
}
Contador ← Contador + 1
}
Retorna Menor
FIN

Programa #22:
Programa que calcula la suma de un listado de 13 números.
Se aplicará el siguiente algoritmo:

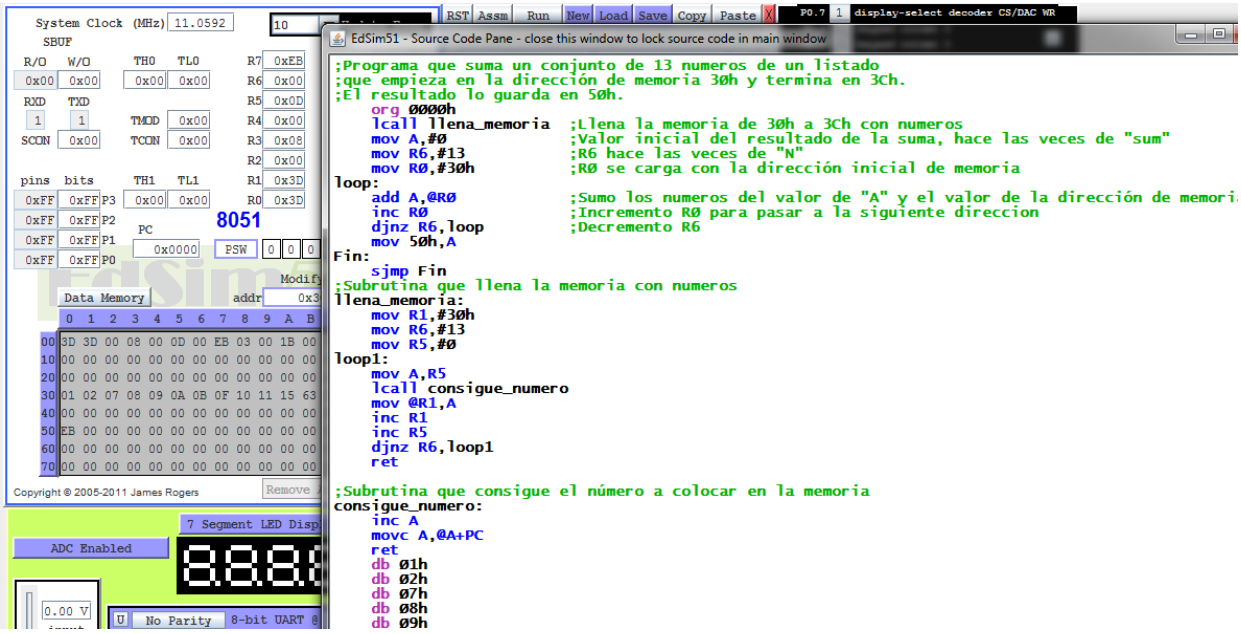
INICIO
Enteros sum, N, ElementoCorriente, i, listado(N)
sum ← 0
i ← 0
Mientras (i<=N)
{
ElementoCorriente ← listado(i)
sum ← sum + ElementoCorriente
i ← i +1
}
Retorna sum
FIN

El listado del programa es:

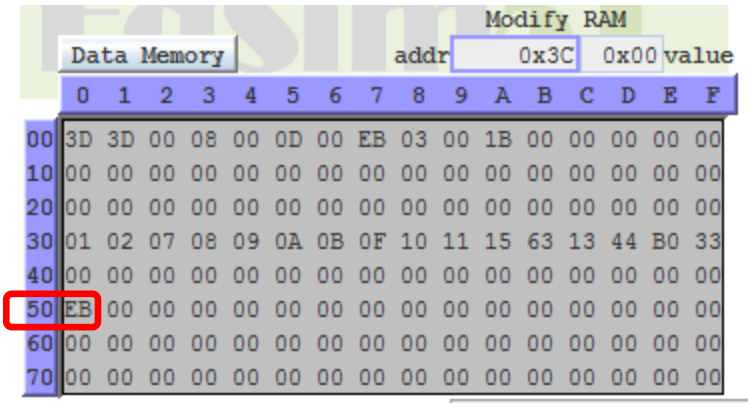
```
;Programa que suma un conjunto de 13 números de un listado
;que empieza en la dirección de memoria 30h y termina en 3Ch.
;El resultado lo guarda en 50h.
        org 0000h
        lcall llena_memoria  ;Llena la memoria de 30h a 3Ch con números
        mov A,#0             ;Valor inicial del resultado de la suma, hace las veces de "sum"
        mov R6,#13           ;R6 hace las veces de "N"
        mov R0,#30h          ;R0 se carga con la dirección inicial de memoria
loop:
        add A,@R0            ;Sumo los números del valor de "A" y el valor de la dirección de memoria
        inc R0               ;Incremento R0 para pasar a la siguiente dirección
        djnz R6,loop         ;Decremento R6
        mov 50h,A
Fin:
        sjmp Fin
;Subrutina que llena la memoria con números
llena_memoria:
        mov R1,#30h
        mov R6,#13
        mov R5,#0
loop1:
        mov A,R5
        lcall consigue_numero
        mov @R1,A
        inc R1
        inc R5
        djnz R6,loop1
        ret
```

```
;Subrutina que consigue el número a colocar en la memoria
consigue_numero:
        inc A
        movc A,@A+PC
        ret
        db 01h
        db 02h
        db 07h
        db 08h
        db 09h
        db 0Ah
        db 0Bh
        db 0Fh
        db 10h
        db 11h
        db 15h
        db 63h
        db 13h
        END
```

En el EDSIM51:



Resultado:



Programa #23:
Programa que calcula el producto de un listado de 8 números.
Se aplicará el algoritmo siguiente:

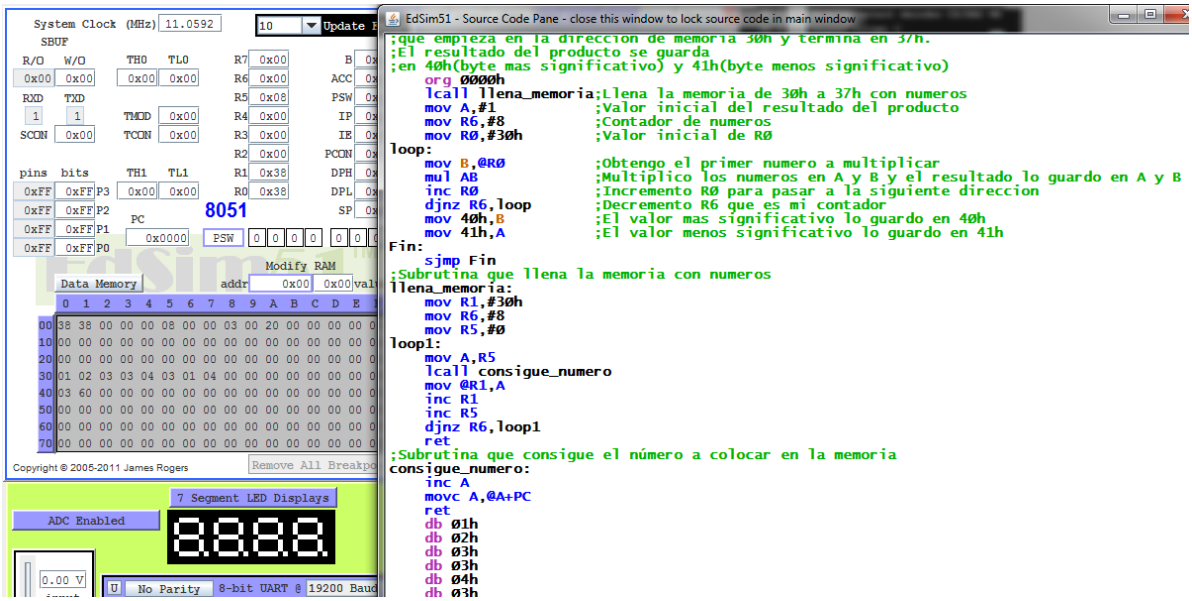
INICIO
Enteros product, ElementoCorriente, i, N, listado(N)
product ← 1
i ← 0
Mientras(i<=N)
{
ElementoCorriente ← listado(i)
product ← product x ElementoCorriente
i ← i + 1
}
Retorna product
FIN

El listado del programa es:

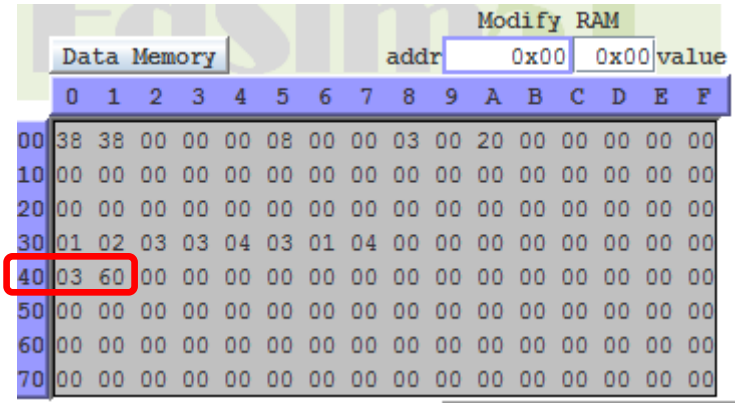
```
;Programa que multiplica un conjunto de 8 numeros de un listado
;que empieza en la dirección de memoria 30h y termina en 37h.
;El resultado del producto se guarda
;en 40h(byte mas significativo) y 41h(byte menos significativo)
        org 0000h
        lcall llena_memoria     ;Llena la memoria de 30h a 37h con números
        mov A,#1                ;Valor inicial del resultado del producto
        mov R6,#8               ;Contador de números
        mov R0,#30h             ;Valor inicial de R0
loop:
        mov B,@R0               ;Obtengo el primer numero a multiplicar
        mul AB                  ;Multiplico los numeros en A y B y el resultado lo guardo en A y B
        inc R0                  ;Incremento R0 para pasar a la siguiente direccion
        djnz R6,loop            ;Decremento R6 que es mi contador
        mov 40h,B               ;El valor mas significativo lo guardo en 40h
        mov 41h,A               ;El valor menos significativo lo guardo en 41h
```

```
Fin:
        sjmp Fin
;Subrutina que llena la memoria con números
llena_memoria:
        mov R1,#30h
        mov R6,#8
        mov R5,#0
loop1:
        mov A,R5
        lcall consigue_numero
        mov @R1,A
        inc R1
        inc R5
        djnz R6,loop1
        ret
;Subrutina que consigue el número a colocar en la memoria
consigue_numero:
        inc A
        movc A,@A+PC
        ret
        db 01h
        db 02h
        db 03h
        db 03h
        db 04h
        db 03h
        db 01h
        db 04h
        END
```

En el EDSIM51:



Resultado:



Programa #24:
Programa que realiza el ordenamiento por "selección" de un listado de números.
En el programa se limpia inicialmente la memoria y luego se llena con el listado de (N=16) números.
Se muestra la letra "d" cuando se termina de llenar con la lista desordenada y luego se muestra la letra "o" cuando el programa finaliza de ordenar el listado.

Se aplicará el algoritmo siguiente:

```
INICIO
Enteros listado(N), N, i, minimo, j
i ← 0
Listado(minimo) ← ∞
Mientras(i <= N)
{
        j ←  i
        Mientras(j <= N)
           {
                Si (Listado(j) <  Listado(minimo))
                 {
                        Listado(minimo) ← Listado(j)
                   }
                 j ← j+1
             }
        Intercambio (Listado(i), Listado(minimo))
        i  ← i+1
}
FIN
```

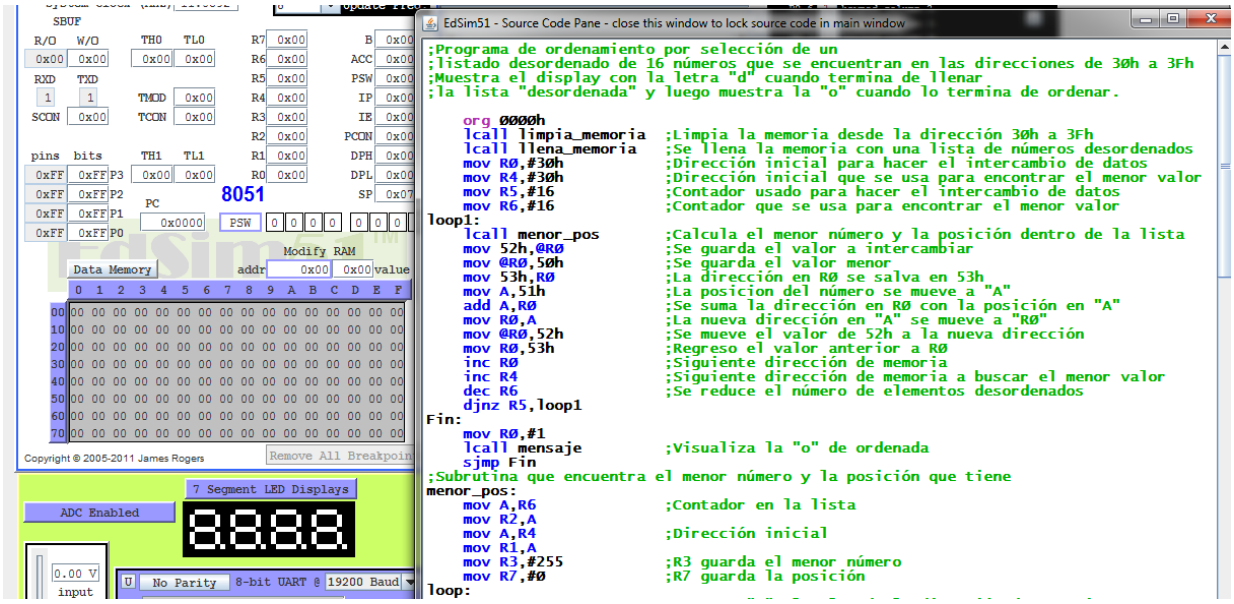
El listado del programa es:

```
;Programa de ordenamiento por selección de un
;listado desordenado de 16 números que se encuentran en las direcciones de 30h a 3Fh
;Muestra el display la letra "d" cuando termina de llenar
;la lista "desordenada" y luego muestra la "o" cuando lo termina de ordenar.

        org 0000h
        lcall limpia_memoria    ;Limpia la memoria desde 30h a 3Fh
        lcall llena_memoria     ;Se llena la memoria con una lista de números desordenados
        mov R0,#30h             ;Dirección inicial para hacer el intercambio de datos
        mov R4,#30h             ;Dirección inicial que se usa para encontrar el menor valor
        mov R5,#16              ;Contador usado para hacer el intercambio de datos
        mov R6,#16              ;Contador que se usa para encontrar el menor valor
loop1:
        lcall menor_pos         ;Calcula el menor número y la posición dentro de la lista
        mov 52h,@R0             ;Se guarda el valor a intercambiar
        mov @R0,50h             ;Se guarda el valor menor
        mov 53h,R0              ;La dirección en R0 se salva en 53h
        mov A,51h               ;La posicion del número se mueve a "A"
        add A,R0                ;Se suma la dirección en R0 con la posición en "A"
        mov R0,A                ;La nueva dirección en "A" se mueve a "R0"
        mov @R0,52h             ;Se mueve el valor de 52h a la nueva dirección
        mov R0,53h              ;Regreso el valor anterior a R0
        inc R0                  ;Siguiente dirección de memoria
        inc R4                  ;Siguiente dirección de memoria a buscar el menor valor
        dec R6                  ;Se reduce el número de elementos desordenados
        djnz R5,loop1
Fin:
        mov R0,#1
        lcall mensaje           ;Muestra la letra "o" indicando un listado “ordenado”
        sjmp Fin                ;Termina el programa
;Subrutina que encuentra el menor número y la posición que tiene
menor_pos:
        mov A,R6                ;Contador en la lista
        mov R2,A
        mov A,R4                ;Dirección inicial
        mov R1,A
        mov R3,#255             ;R3 guarda el menor número
        mov R7,#0               ;R7 guarda la posición
loop:
        mov A,@R1               ;Muevo a "A" el valor de la dirección de memoria
        subb A,R3;
        jnc mayor               ;El carry "C" es 1 si es negativo la operación "A - R3"
        mov A,@R1               ;El valor en @R1 es menor
        mov R3,A                ;Se guarda el valor menor en R3
        mov 51h,R7              ;La posición se guarda en 51h
mayor:
        inc R1
        inc R7
        djnz R2,loop
        mov  50h,R3             ;El menor número se guarda en 50h
        ret
```

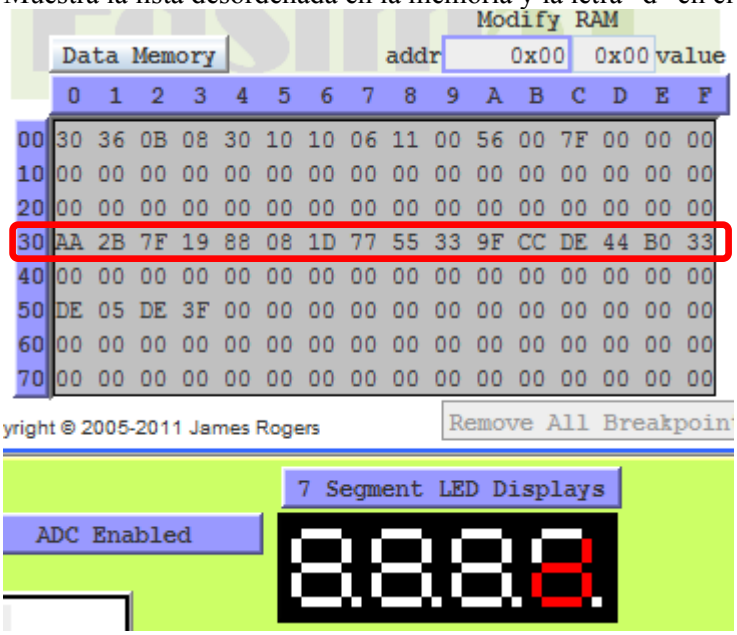
```
;Llena la memoria con números desordenados
llena_memoria:
        mov R0,#30h
        mov R1,#16
        mov R2,#0
lazo:
        mov A,R2
        lcall obtiene_numero
        mov @R0,A
        inc R2
        inc R0
        djnz R1, lazo
        mov R0,#0
        lcall mensaje
        ret

;Limpia la memoria de 30h a 3fh colocando 0s.
limpia_memoria:
        mov R0,#30h
        mov R1,#16
lazo1:
        mov @R0,#0
        inc R0
        djnz R1,lazo1
        ret

;Números a colocar en las posiciones de memoria
obtiene_numero:
        inc A
        movc A,@A+PC
        ret
        db 0AAh;
        db 2Bh;
        db 7Fh;
        db 19h;
        db 88h;
        db 08h;
        db 1Dh;
        db 77h;
        db 55h;
        db 33h;
        db 9Fh;
        db 0CCh;
        db 0DEh;
        db 44h;
        db 0B0h;
        db 33h;
```

```
;Muestra en el display la letra "d" o "o" según sea el caso
mensaje:
        mov A,#0
        cjne R0,#0,muestra_o
        sjmp muestra_d
muestra_o:
        mov A,R0
muestra_d:
        lcall valores
        clr P3.4
        clr P3.3
        mov P1,A
        ret
;Patrones de bits de las letras "d" y "o"
valores:
        inc A
        movc A,@A+PC
        ret
        db 0A1h; "d"
        db 0A3h; "o"
        END
```

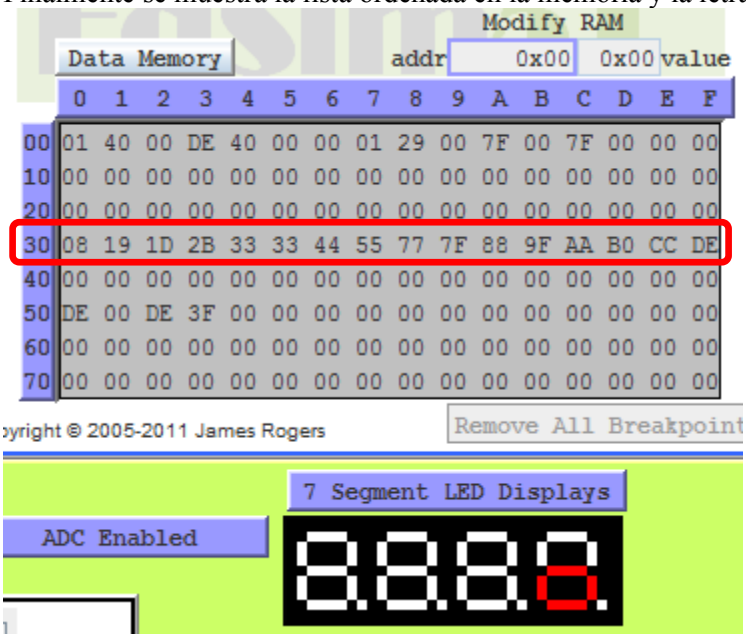
En el EDSIM51:



En ejecución:
Muestra la lista desordenada en la memoria y la letra “d” en el display



Finalmente se muestra la lista ordenada en la memoria y la letra “o” en el display



Programa #25:
Programa que utilizando el algoritmo de burbuja realiza el ordenamiento de un listado de números. Para N=16.
Se empleará el algoritmo siguiente:

```
INICIO
Enteros X, Z, LISTADO(N)
X←0
MIENTRAS(X < N)
 { Z ← N
        MIENTRAS (Z>=0)
        {
                SI (LISTADO (Z) < LISTADO (Z-1))
                {
                        INTERCAMBIO (LISTADO (Z), LISTADO (Z-1))
                }
               Z← Z - 1
        }
     X ← X + 1
    }
FIN
```

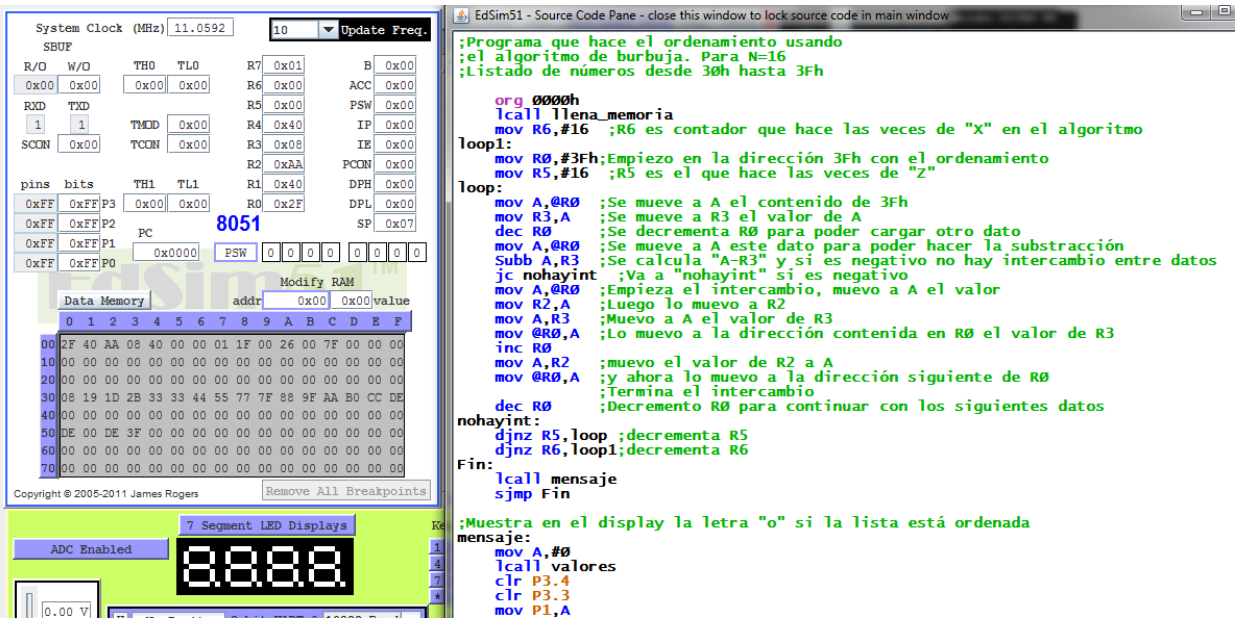
El listado del programa es:

```
;Programa que hace el ordenamiento usando
;el algoritmo de burbuja
;Listado de números desde 30h hasta 3fh

        org 0000h
        lcall llena_memoria
        mov R6,#16          ;R6 es contador que hace las veces de "X" en el algoritmo
loop1:
        mov R0,#3Fh         ;Empiezo en la dirección 3Fh con el ordenamiento
        mov R5,#16          ;R5 es el que hace las veces de "Z"
loop:
        mov A,@R0           ;Se mueve a A el contenido de 3Fh
        mov R3,A            ;Se mueve a R3 el valor de A
        dec R0              ;Se decrementa R0 para poder cargar otro dato
        mov A,@R0           ;Se mueve a A este dato para poder hacer la substracción
        Subb A,R3           ;Se calcula "A-R3" y si es negativo no hay intercambio entre datos
        jc nohayint         ;Va a "nohayint" si es negativo
        mov A,@R0           ;Empieza el intercambio, muevo a A el valor
        mov R2,A            ;Luego lo muevo a R2
        mov A,R3            ;Muevo a A el valor de R3
        mov @R0,A           ;Lo muevo a la dirección contenida en R0 el valor de R3
        inc R0
        mov A,R2            ;muevo el valor de R2 a A
        mov @R0,A           ;y ahora lo muevo a la dirección siguiente de R0
                            ;Termina el intercambio
        dec R0              ;Decremento R0 para continuar con los siguientes datos
```

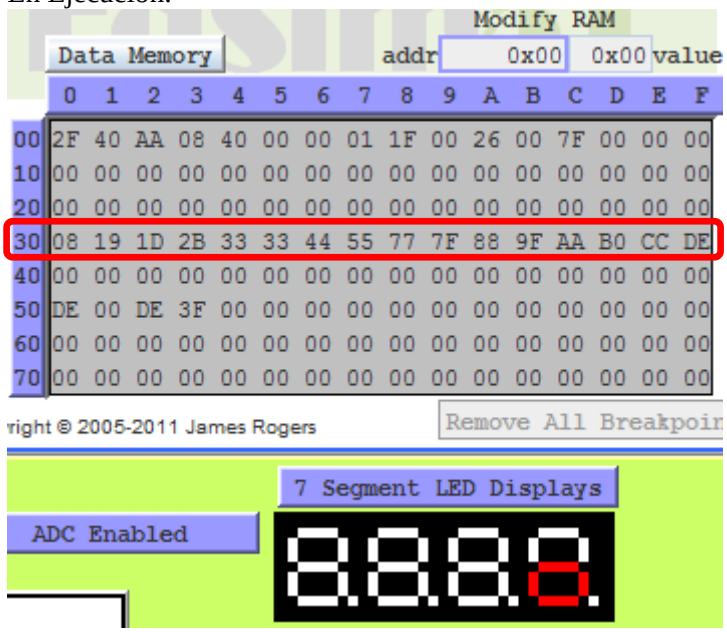
```
nohayint:
        djnz R5,loop            ;decrementa R5 y va a loop si no es cero
        djnz R6,loop1           ;decrementa R6 y va a loop1 si no es cero
Fin:
        lcall mensaje
        sjmp Fin

;Muestra en el display la letra "o" si la lista está ordenada
mensaje:
        mov A,#0
        lcall valores
        clr P3.4
        clr P3.3
        mov P1,A
        ret

;Subrutina para llenar la memoria de datos desde 30h hasta 3Fh
llena_memoria:
        mov R0,#0
        mov R1,#30h
        mov R2,#16
loop2:
        mov A,R0
        lcall consigue_numero
        mov @R1,A
        inc R0
        inc R1
        djnz R2,loop2
        ret
consigue_numero:
        inc A
        movc A,@A+PC
        ret
        db 0AAh;
        db 2Bh;
        db 7Fh;
        db 19h;
        db 88h;
        db 08h;
        db 1Dh;
        db 77h;
        db 55h;
        db 33h;
        db 9Fh;
        db 0CCh;
        db 0DEh;
        db 44h;
        db 0B0h;
        db 33h;
```

```
        ;Patrones de bits de la letra "o"
valores:
        inc A
        movc A,@A+PC
        ret
        db 0A3h; "o"
        END
```

En el EDSIM51:



En Ejecución:



Ejercicio #18:
Desarrolle un programa que averigua si un número es primo o no.
Si es primo que muestre “P” si no que muestre “0”.
Aplicar el siguiente algoritmo:

```
INICIO
Enteros X, N
X <-- 2
 Mientras(X es diferente a N)
{
        ¿Es entero(N/X)?
        {
                Si es entero
                Retorna mensaje "0" indica que no es primo
                Termina
        }
        {
                No es entero
                X <-- X +1
        }
}
Retorna mensaje "P" indica que es primo
FIN
```

Programa #26:
Conversión de un número decimal de 8 bits a cualquier base.
Aplicar el siguiente algoritmo:

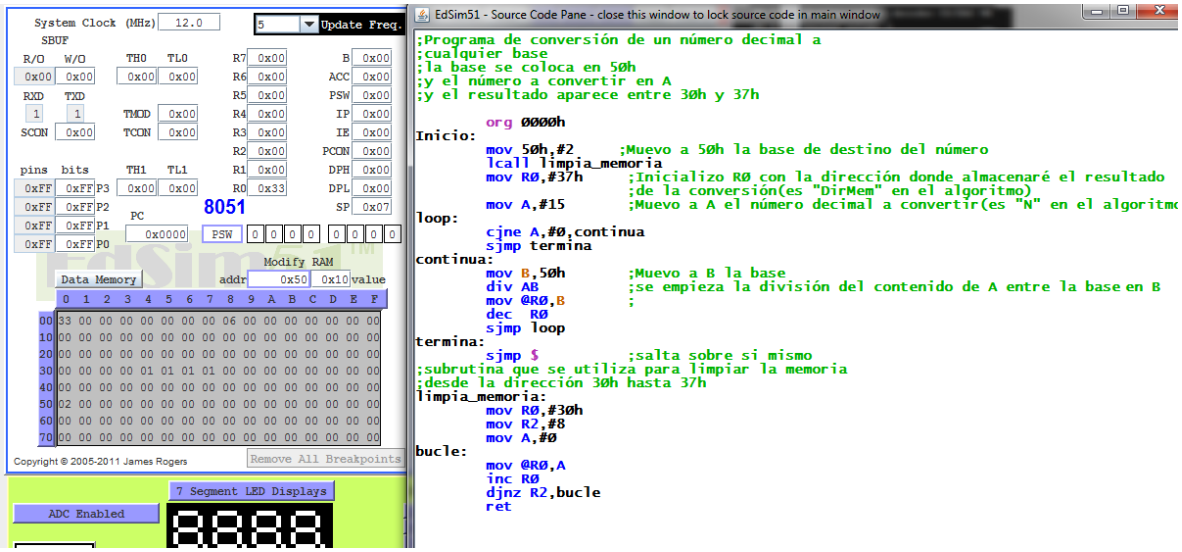
```
INICIO
Enteros Num, base, DirMem
DirMem ← 37h
Mientras (Num es distinto a 0)
{
        DirMem ← Resto(Num/base)  ;Almaceno el resto de la división en la
                                  ;dirección de memoria(DirMem)
        Num ← Num/base            ; El nuevo "Num" es el cociente de "Num"  entre la "base"
        DirMem ← DirMem-1         ;Decremento DirMem
}
FIN
```

El listado del programa es:

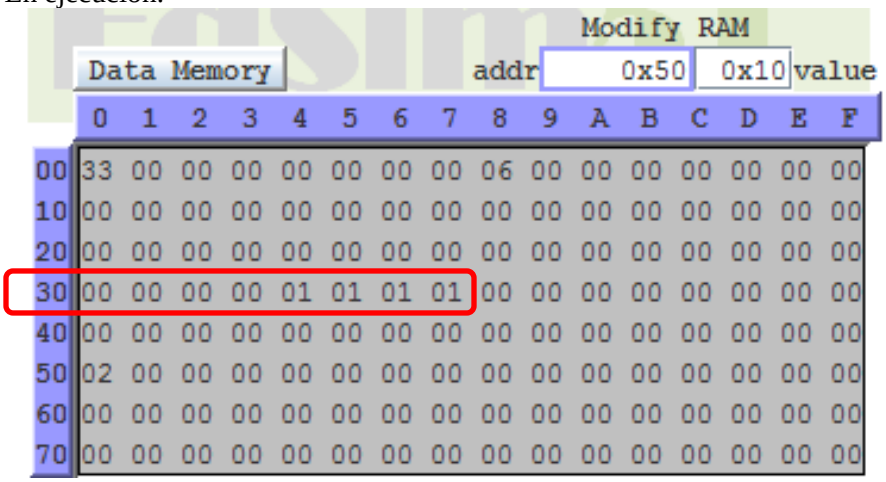
```
;Programa de conversión de un número decimal a
;cualquier base
;la base se coloca en 50h
;y el número a convertir en A
;y el resultado aparece entre 30h y 37h

                org 0000h
Inicio:
                mov 50h,#2              ;Muevo a 50h la base de destino del número
                lcall limpia_memoria
                mov R0,#37h             ;Inicializo R0 con la dirección donde almacenaré el resultado
                                        ;de la conversión(es "DirMem" en el algoritmo)
                mov A,#15               ;Muevo a A el número decimal a convertir(es "Num" en el
algoritmo)
loop:
                cjne A,#0,continua
                sjmp termina
continua:
                mov B,50h               ;muevo a B la base
                div AB                  ;se empieza la división del contenido de A entre la base en B
                mov @R0,B               ;
                dec  R0
                sjmp loop
termina:
                sjmp $                  ;salta sobre si mismo
```

```
;subrutina que se utiliza para limpiar la memoria
;desde la dirección 30h hasta 37h
limpia_memoria:
                mov R0,#30h
                mov R2,#8
                 mov A,#0
bucle:
                mov @R0,A
                inc R0
                djnz R2,bucle
                ret
```

En el EDSIM51:



En ejecución:



APENDICE 1: Conjunto de Instrucciones de lenguaje ensamblador para el Microcontrolador 8051

Operaciones Aritméticas

Mnemónico	Descripción	Bytes	Ciclos de Máquina
ADD A,Rn	Suma registro a A y el resultado se guarda en A	1	1
ADD A,direct	Suma byte directo a A y el resultado se guarda en A	2	1
ADD A,@Ri	Suma RAM indirecto a A y el resultado se guarda en A	1	1
ADD A,#data	Suma dato inmediato a A y el resultado se guarda en A	2	1
ADDC A,Rn	Add register to A with Carry	1	1
ADDC A,direct	Add direct byte to A with Carry	2	1
ADDC A,@Ri	Add indirect RAM to A with Carry	1	1
ADDC A,#data	Add immediate data to A with Carry	2	1
SUBB A,Rn	Subtract register from A with Borrow	1	1
SUBB A,direct	Subtract direct byte from A with Borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB A,#data	Subtract immediate data from A with Borrow	2	1
INC A	Incremento A	1	1
INC Rn	Incremento registro	1	1
INC direct	Incremento byte directo	2	1
INC @Ri	Incremento RAM indirecto	1	1
DEC A	Decremento A	1	1
DEC Rn	Decremento registro	1	1
DEC direct	Decremento byte directo	2	1
DEC @Ri	Decremento RAM indirecto	1	1
INC DPTR	Incremento Data Pointer	1	2
MUL AB	Multiplica A y B (A x B => BA)	1	4
DIV AB	Divide A entre B (A/B => A + B)	1	4
DA A	Ajuste decimal de A	1	1

Operaciones Lógicas

Mnemónico	Descripción	Bytes	Ciclos de Máquina
ANL A,Rn	AND register to A	1	1
ANL A,direct	AND direct byte to A	2	1
ANL A,@Ri	AND indirect RAM to A	1	1
ANL A,#data	AND immediate data to A	2	1
ANL direct,A	AND A to direct byte	2	1
ANL direct,#data	AND immediate data to direct byte	3	2
ORL A,Rn	OR register to A	1	1
ORL A,direct	OR direct byte to A	2	1
ORL A,@Ri	OR indirect RAM to A	1	1
ORL A,#data	OR immediate data to A	2	1
ORL direct,A	OR A to direct byte	2	1
ORL direct,#data	OR immediate data to direct byte	3	2
XRL A,Rn	Exclusive-OR register to A	1	1
XRL A,direct	Exclusive-OR direct byte to A	2	1
XRL A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL A,#data	Exclusive-OR immediate data to A	2	1
XRL direct,A	Exclusive-OR A to direct byte	2	1
XRL direct,#data	Exclusive-OR immediate data to direct byte	3	2
CLR A	Clear A	1	1
CPL A	Complement A	1	1
RL A	Rotate A Left	1	1
RLC A	Rotate A Left through Carry	1	1
RR A	Rotate A Right	1	1
RRC A	Rotate A Right through Carry	1	1
SWAP A	Swap nibbles within A	1	1

Operaciones de Transferencia de Datos

Mnemónico	Descripción	Bytes	Ciclos de Máquina
MOV A,Rn	Move register to A	1	1
MOV A,direct	Move direct byte to A	2	1
MOV A,@Ri	Move indirect RAM to A	1	1
MOV A,#data	Move immediate data to A	2	1
MOV Rn,A	Move A to register	1	1
MOV Rn,direct	Move direct byte to register	2	2
MOV Rn,#data	Move immediate data to register	2	1
MOV direct,A	Move A to direct byte	2	1
MOV direct,Rn	Move register to direct byte	2	2
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,#data	Move immediate data to direct byte	3	2
MOV @Ri,A	Move A to indirect RAM	1	1
MOV @Ri,direct	Move direct byte to indirect RAM	2	2
MOV @Ri,#data	Move immediate data to indirect RAM	2	1
MOV DPTR,#data16	Load Data Pointer with 16-bit constant	2	1
MOVC A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX @Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX @DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH direct	Push byte directo hacia la pila	2	2
POP direct	Pop byte directo desde la pila	2	2
XCH A,Rn	Intercambio de registro con A	1	1
XCH A,direct	Intercambio byte directo con A	2	1
XCH A,@Ri	Intercambio RAM indirecto con A	1	1
XCHD A,@Ri	Exchange low-order Digit indirect RAM with A	1	1

Operaciones de único Bit (Variable Booleana)

Mnemónico	Descripción	Bytes	Ciclos de Máquina
CLR C	Pone a “0” el flag Carry	1	1
CLR bit	Pone a “0” bit directo	2	1
SETB C	Pone a uno el flag Carry	1	1
SETB bit	Pone a uno el bit directo	2	1
CPL C	Complementa el flag Carry	1	1
CPL bit	Complementa el bit directo	2	1
ANL C,bit	AND direct bit to Carry flag	2	2
ANL C,/bit	AND complement of direct bit to Carry flag	2	2
ORL C,bit	OR direct bit to Carry flag	2	2
ORL C,/bit	OR complement of direct bit to Carry flag	2	2
MOV C,bit	Mover bit directo a flag de Acarreo	2	1
MOV bit,C	Mover el flag de Acarreo a bit directo	2	2

Control del Flujo de Programa

Mnemónico	Descripción	Bytes	Ciclos de Máquina
ACALL addr11	Llamada Absoluta a subrutina	2	2
LCALL addr16	Llamada Larga a subrutina	3	2
RET	Retorno de la subrutina	1	2
RETI	Retorno de la interrupción	1	2
AJMP addr11	Salto Absoluto	2	2
LJMP addr16	Salto Largo incondicional	3	2
SJMP rel	Salto corto a una dirección relativa	2	2
JMP @A+DPTR	Salto indirecto relativo a DPTR	1	2
JZ rel	Salta si A es cero	2	2
JNZ rel	Salta si A no es cero	2	2
JC rel	Salta si el flag Carry es 1	2	2
JNC rel	Salta si el flag Carry es cero	2	2
JB bit,rel	Salta si el Bit directo es 1	3	2
JNB bit,rel	Salta si el Bit directo no es 1	3	2
JBC bit,rel	Salta si el Bit directo es 1 y pone a 0 el Bit	3	2
CJNE A,direct,rel	Compare direct to A and Jump if Not Equal	3	2
CJNE A,#data,rel	Compare immediate to A and Jump if Not Equal	3	2
CJNE Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	2
CJNE @Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	2
DJNZ Rn,rel	Decrementa registro y Salta si No es cero	2	2
DJNZ direct,rel	Decrementa byte directo y Salta si No es cero	3	2
NOP	No operación	1	1

Notas

A	Registro Acumulador
Rn	Registros de Trabajo de R0-R7
direct	Es cualquier dirección dentro de las 128 posiciones de la RAM interna, cualquier puerto Entrada/Salida, registro de control o estado
@Ri	Posición Indirecta de la RAM interna direccionado por el registro R0 o R1
#data	Constante de 8-bits incluida en la instrucción
#data16	Constante de 16-bits incluida en la instrucción
bit	Es cualquier bit direccionable, cualquier pin de Entrada/Salida, bit de control o estado
addr16	Dirección de destino puede ser cualquiera dentro de espacio de direcciones de programa de 64-kByte
addr11	Dirección de destino estará dentro del espacio de direcciones de programa de 2-kByte
rel	Son 8-bits de desplazamiento relativo entre (+127, -128)