

Prácticas de SDP-1

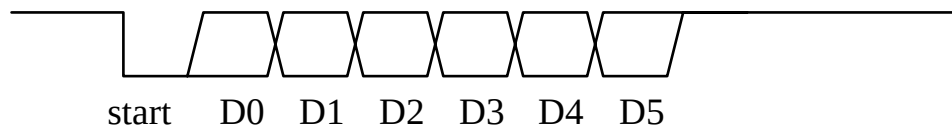
Boletín de ejercicios sobre microcontroladores

Autor: Alfredo del Río Vázquez

Ejercicio 1.-

Un medidor de presión genera cada 10 milisegundos una trama de 7 bits con el formato que se indica:

trama



El estado de reposo corresponde al nivel alto. El primer bit es siempre 0 (bit de start) y los 6 restantes (D0 a D5) indican el valor de la presión en binario, empezando por el bit de menor peso. El valor se expresa en décimas de kg/cm^2 (máximo $6,3 \text{ kg/cm}^2$). La duración de cada bit se puede suponer de 1 ms exacto.

Se desea diseñar un sistema basado en el 8051 que, conectado a la señal *trama*, mantenga actualizado un visualizador de 2 cifras decimales (tipo LED) con el valor de la presión. Además, mientras la presión sea inferior al valor $1,5 \text{ kg/cm}^2$ deberá mantener un led verde encendido. Si excede de $5,8 \text{ kg/cm}^2$ deberá encender un led rojo.

No se permite utilizar el puerto serie para interpretar la trama, porque se prevé emplearlo para la comunicación con un PC en futuras versiones de este sistema.

Solución.-

Antes de comenzar el diseño, se debe adquirir una visión de conjunto de las especificaciones y asignar los recursos del micro.

DISEÑO DEL HARDWARE

Parece razonable llevar la señal *trama* a una entrada de interrupción para detectar rápidamente la llegada del bit *start*.

Los visualizadores de tipo LED no se pueden conectar directamente a un 8051 normal porque necesitan más corriente de la que es capaz de suministrar el micro. Además, la conexión puede realizarse de varias formas:

- 👉 Multiplexada o no multiplexada. Cuando el número de cifras es elevado es razonable multiplexar porque permite ahorrar muchas líneas. En este caso, como se trata de sólo 2 cifras, puede ser razonable una conexión no multiplexada.

- 👉 Activando segmentos individuales o a través de un decodificador BCD a 7 segmentos. Para activar los segmentos se necesitarían 14 buffers (amplificadores de corriente) lo que supone el uso de al menos dos circuitos integrados de este tipo. Con el mismo coste se pueden utilizar decodificadores/drivers BCD a 7 segmentos, lo que simplifica el software. Se elige esta opción, aunque no es la única válida.

Para controlar los decodificadores se necesitan 8 bits (4 para cada cifra). Parece razonable utilizar un puerto completo. Se puede utilizar cualquier puerto completo disponible. Del puerto 3 ya se utiliza una línea y otras podrían ser utilizadas en el futuro (puerto serie), por lo que se descarta. El puerto 0 obligaría a conectar resistencias de pull-up externas, porque sus salidas son del tipo colector abierto. Si en un diseño posterior se precisa utilizar memorias externas, será necesario prescindir del puerto 2. Por lo tanto se elige el puerto 1. Otras soluciones pueden ser válidas según el caso.

El puerto 1 se conecta a dos decodificadores/drivers de BCD a 7 segmentos, que a su vez controlan dos visualizadores de 7 segmentos (los visualizadores hex del simulador ya incluyen el decodificador). Los 4 bits P7-P4 se conectan al dígito más significativo y los 4 bits P3-P0 al dígito menos significativo. De esta forma, puede escribirse directamente en P1 el valor de la presión en formato BCD empaquetado.

En un diseño real se procedería al diseño de un prototipo. Para utilizar el simulador UVI51, se escribe un archivo CNX.

```
* Archivo CNX para el sistema de visualización de presión
* Definición de un micro que ejecutará presion.src
Upresion.src 12.0 16 1024
* Conexión de puertos: trama en P3.2, displays en P1
P32=2
P10=3
P11=4
P12=5
P13=6
P14=7
P15=8
P16=9
P17=10
* Definición de los visualizadores (1º el de mayor peso)
H 7 8 9 10 0 3
H 3 4 5 6 0 4
* Definición de un micro adicional que genere tramas
* para verificar el funcionamiento
Utramas.src 12.0 16 1024
* Conexión del pin P3.2 (salida de tramas) al pin P3.2
* del micro principal
P32=2
* Conexión de P1 a un conjunto de conmutadores para definir
* el valor de la presión que será enviado (cada 1 ms)
P10=11
Cpres0 11 0 1 14
P11=12
Cpres1 12 0 1 13
P12=13
Cpres2 13 0 1 12
P13=14
Cpres3 14 0 1 11
P14=15
```

```

Cpres4 15 0 1 10
P15=16
Cpres5 16 0 1 9
* Se define un analizador para verificar el sistema
Atrama
* Se definen buses y se asignan nombres a nodos
N2 trama
N17 muestra
Bpres 6 11 12 13 14 15 16
Bdis1 4 7 8 9 10
Bdis0 4 3 4 5 6
N3 D0
N4 D1
N5 D2
N6 D3
N7 D4
N8 D5
N9 D6
N10 D7
N11 P0
N12 P1
N13 P2
N14 P3
N15 P4
N16 P5

```

En primer lugar se escribe el programa del micro auxiliar (el que genera la trama, es decir el que simula al medidor de presión), porque el funcionamiento de este micro se puede verificar sin necesidad del principal.

Para poder simular el sistema, es necesario que exista también un programa, aunque sea elemental, para el micro principal. Basta con una instrucción `jmp $`.

El programa del micro auxiliar puede ser como el que se indica a continuación.

```

; Programa del micro auxiliar: Genera tramas para verificación
;
trama bit P3.2
;
divis data 40h
pulso data 41h ; 0= start, 6= último, 7 8 y 9 relleno.
presion data 42h
;
org 0
    jmp inicio
org 0bh
    jmp inttim0
;
org 100h
inicio:
    mov tmod,#2
    mov th0,#-250
    mov tl0,#-250
    mov divis,#4
    mov ie,#82h

```

```

    call espera    ; para no enviar tramas hasta que el otro
;                micro esté preparado
; carga inicial presión
    mov a,p1
    clr c
    rlc a
    mov presion,a    ; actualiza presión
;
    setb tr0
    jmp $
;
inttim0:
    djnz divis,salir
    mov divis,#4
; Entra aquí cada milisegundo
    mov a,pulso
    cjne a,#7,void
void:
    jnc reposo
; trama activa
    xch a,presion
    rrc a
    mov trama,c
    xch a,presion
    jmp comun
;
reposo:
    setb trama
;
comun:
    inc pulso
    mov a,pulso
    cjne a,#10,salir
    mov pulso,#0
    mov a,p1
    clr c
    rlc a
    mov presion,a    ; actualiza presión
;
salir:
    reti
;
espera:
    mov b,#0
    djnz b,$
    ret
;
end

```

Se recomienda comprobar este programa, actuando sobre los conmutadores para generar diferentes tramas. Las tramas se deben verificar con el analizador lógico.

Antes de escribir el código para el micro principal, que es el que realmente se pide en este ejercicio, es conveniente definir su diagrama de flujo.

A continuación se muestra un diagrama de flujo a grandes rasgos.


```

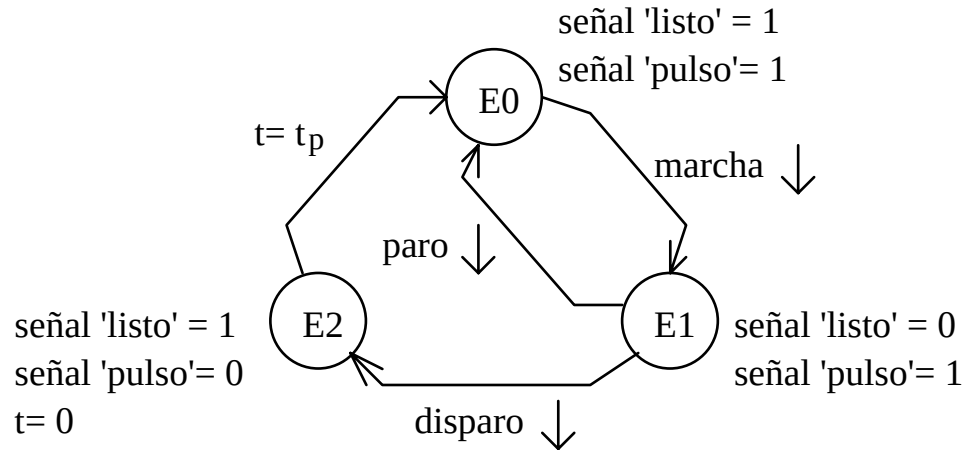
    jmp inicio
org 03h
    jmp intext0
org 0bh
    jmp inttim0
;
inicio:
    mov presion,#0
    mov tmod,#1
    setb it0
    mov ie,#83h    ; habilita interr. extena 0 y temp. 0
;
principal:
    mov a,presion
    call bin2bcd
    mov p1,a
    jmp principal
;
intext0:
    mov tl0,#LOW(-1500)    ; prepara primer retardo 1,5 ms
    mov th0,#HIGH(-1500)
    mov regdesp,#0
    mov numbit,#6
    clr ex0
    setb tr0
    reti
;
inttim0:
    push acc
    push psw
    mov tl0,#LOW(-1000)    ; siguientes retardos 1 ms
    mov th0,#HIGH(-1000)
    mov a,regdesp
; clr marca y setb marca se utilizan solamente para facilitar
; la verificac. de los instantes de muestreo en el analiz.
    clr marca
    mov c,trama
    setb marca
    rrc a
    mov regdesp,a
    djnz numbit,salir
    clr tr0
    clr ie0    ; borra peticiones pendientes
    setb ex0
    rr a
    rr a
    mov presion,a
salir:
    pop psw
    pop acc
    reti
;
; recibe binario en A y devuelve bcd en A
bin2bcd:
    push psw
    push b
    mov b,#10
    div ab

```

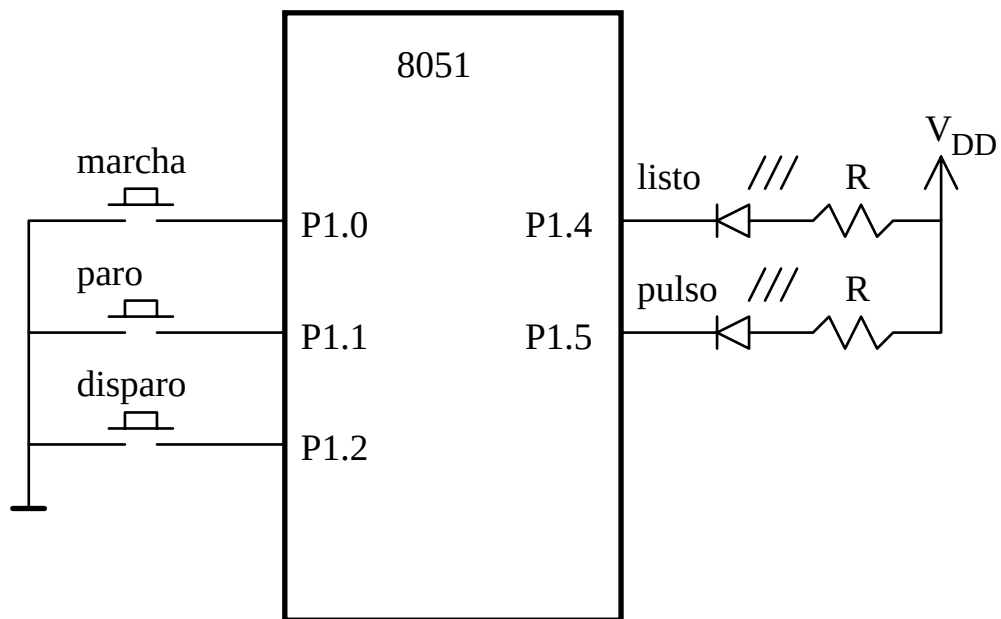
```
    swap a
    orl a,b
    pop b
    pop psw
    ret
;
end
```

Ejercicio 2.-

Se desea realizar mediante un 8051 el diagrama de estados que se indica a continuación.



El sistema dispone de 3 pulsadores denominados marcha, paro y disparo conectados a las entradas P1.0, P1.1 y P1.2 respectivamente, según se muestra en la figura. Las salidas P1.4 y P1.5 generan las señales listo y pulso respectivamente.



El circuito real intercala amplificadores de corriente para el control de los LEDs, pero su funcionamiento es equivalente al del circuito mostrado, a nivel lógico. Es decir, cada LED se enciende con el nivel bajo de la señal correspondiente.

Los cambios de estado se producen al accionar los pulsadores indicados. El cambio del estado E2 al E0 se produce de forma automática después de permanecer un tiempo t_p en el estado E2. Este tiempo debe poder ajustarse con una modificación sencilla del programa a cualquier valor entre 1 y 255 ms, con un error máximo de 1,5 ms. Probar con 20 ms.

Solución.-

En primer lugar, se define un archivo CNX para poder realizar la simulación del sistema.

```
* Sistema definido mediante un diagrama de estados (ejercicio
*   2 boletín)
* Se define un analizador lógico
Amaquest
* El micro ejecutará el programa maquest.src
Umaquest.src 12 256 1024
* Pulsadores
P10=2
TMAR 2 0 1
P11=3
TPAR 3 0 2
P12=4
TDIS 4 0 4
* LEDs
P14=5
LON 1 5 9 V
P15=6
LPULS 1 6 12 A
* Etiquetas de nodos
N2 marcha
N3 paro
N4 disparo
N5 listo
N6 pulso
```

La solución propuesta se basa en la siguiente estructura:

Programa principal que realiza inicializaciones del HW y de variables para entrar después en un bucle infinito en el que se llama a las subrutinas ent_sal y maq_est.

- 👉 Subrutina ent_sal que actualiza las salidas y lee las entradas.
- 👉 Subrutina maq_est que realiza la secuencia de estados.
- 👉 Subrutina temp_soft que arranca un temporizador software descendente.
- 👉 Subrutina de interrupción del temporizador 0 que actualiza el temporizador software y activa el indicador alarma cuando llega a 0.

Para realizar la máquina de estados se utiliza una variable (byte) *estado* que indica el estado actual. Además, se emplea el indicador (bit) *cambio* para marcar los cambios de estado. La rutina maq_est determina el estado actual mediante una cadena de CJNE. Si se acaba de entrar en ese estado realiza varias acciones

- 👉 Borrar marca cambio
- 👉 Actualizar copia interna de las salidas

En todo caso, comprueba si debe cambiar de estado (capacidades de transición). Si es así, actualiza la variable *estado* y activa el indicador *cambio*. Para determinar las capacidades de transición emplea las copias internas de las entradas. Para el paso de E2 a E0 comprueba el bit temp_alarma.

A continuación se indica la solución propuesta.

```
; Programa del generador de impulsos definido mediante una
; máquina de estados (boletín de ejercicios de SDP1)
;
; Constantes
RECARGA equ 6
DUR_PULSO equ 20          ; 20 ms (error -> aprox. de 19 a 20 ms)
; Estados
REPOSO equ 0
ACTIVO equ 1
GENPULSO equ 2
;
; Copia interna de las E/S
marcha bit 0
paro bit 1
disparo bit 2
listo bit 3
pulso bit 4
;
; Para la máquina de estados
estado data 40h
cambio bit 5
;
; Para el temporizador software (decrementa cada 1 ms,
;                                     tmax= 0,255 s)
divisor data 41h
temp_cents data 42h
temp_run bit 8
temp_era0 bit 9
temp_alarma bit 10
;
tiempo data 43h
;
org 0
jmp inicio
org 0bh
jmp inttim0
;
org 100h
inicio:
; temp. 0 en modo 2
mov tmod,#2
mov th0,#RECARGA
mov tl0,#RECARGA
mov divisor,#4
mov temp_cents,#0
setb temp_era0
clr temp_run
clr temp_alarma
;
mov estado,#REPOSO
```

```

clr cambio
setb listo
setb pulso
;
mov ie,#82h
setb tr0
;
bucle:
call ent_sal
call maq_est
jmp bucle
;
; P10=MAR, P11=PAR, P12=DISP, P14=ON, P15=PULSO
ent_sal:
mov a,p1
mov c,listo
mov acc.4,c
mov c,pulso
mov acc.5,c
orl a,#0cfh          ; deja a '1' los que no son salidas
mov p1,a
mov a,p1
mov c,acc.0
mov marcha,c
mov c,acc.1
mov paro,c
mov c,acc.2
mov disparo,c
ret
;
maq_est:
mov a,estado
cjne a,#REPOS0,noreposo
jnb cambio,reposo2
; sólo al entrar en este estado
clr cambio
setb pulso
setb listo
; siempre que se encuentre en este estado
reposo2:
jb marcha,salmaq
mov estado,#ACTIVO
setb cambio
jmp salmaq
;
noreposo:
cjne a,#ACTIVO,noactivo
jnb cambio,activo2
; sólo al entrar en este estado
clr cambio
setb pulso          ; redundante
clr listo
; siempre que se encuentre en este estado
activo2:
jb paro,activo3
mov estado,#REPOS0
setb cambio
jmp salmaq

```

```

activo3:
jb disparo,salmaq
mov estado,#GENPULSO
setb cambio
jmp salmaq
;
noactivo:
cjne a,#GENPULSO,error
jnb cambio,genpulso2
; sólo al entrar en este estado
clr cambio
clr pulso
clr listo ; redundante
mov tiempo,#DUR_PULSO ; arranca temporizador software
call tempsoft

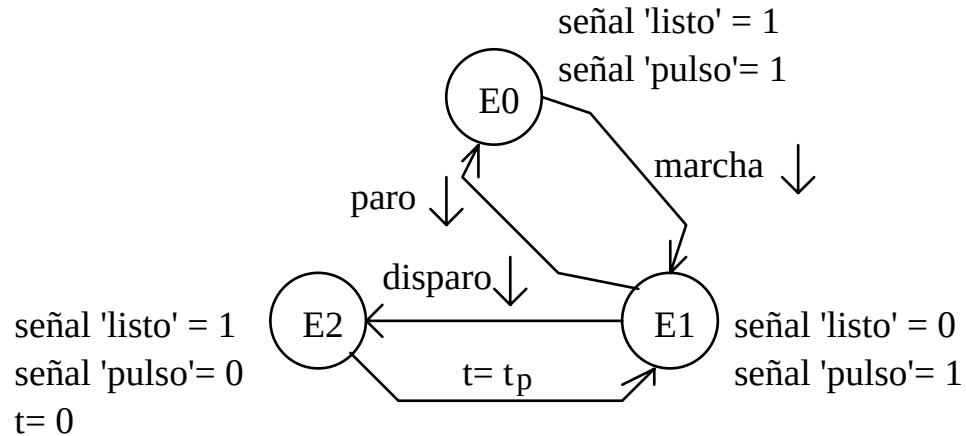
mov temp_cents,#DUR_PULSO ; arranca temporizador software
clr temp_alarma
setb temp_run
; siempre que se encuentre en este estado
genpulso2:
jnb temp_alarma,salmaq
mov estado,#REPOSO
setb cambio
jmp salmaq
;
error:
mov estado,#REPOSO
setb cambio
salmaq:
ret
;
tempsoft:
mov temp_cents,tiempo ; arranca temporizador software
clr temp_alarma
setb temp_run
ret
;
; Esta interrupción se produce cada 250 microsegundos
; el divisor es asíncrono a temp_ (error máx. 1 ms)
inttim0:
djnz divisor,salir
mov divisor,#4
jnb temp_run,salir
djnz temp_cents,nocero
jb temp_era0,salir
setb temp_alarma
setb temp_era0
clr temp_run
salir:
reti
nocero:
clr temp_era0
reti
;
end

```

Es importante que el bucle principal se ejecute en un tiempo breve (menos de 0,5 ms), con objeto de no introducir un error importante en la duración del impulso.

El analizador lógico permite medir la duración de los impulsos generados.

Aunque existen soluciones mucho más simples, la propuesta tiene la virtud de ser muy estructurada, lo que hace mucho más fácil su mantenimiento. Para comprobarlo se propone modificar el diagrama de estados cambiando la transición de E2 a E0 por una transición de E2 a E1 con la misma condición (capacidad de transición).



Además, el sistema empleado para definir el temporizador software puede ser ampliado fácilmente para mantener varios temporizadores software de forma simultánea. El número máximo de temporizadores está limitado por la duración de la rutina de interrupción, que no debe acercarse a 250 μ s como media ni a 500 μ s en el peor caso.

Ejercicio 3.-

Se desea realizar un autómata elemental mediante un microcontrolador 8051, con las siguientes características:

- 1) Debe disponer de 4 terminales de entrada y 4 de salida con niveles compatibles TTL.
- 2) Dispondrá también de 8 *puntos* (memorias de bit) internos.
- 3) El repertorio de instrucciones será el indicado, y empleará la asignación dada de códigos de operación y campo de dirección:

Instrucción		Código	
LD	bit	0000nnnn	(nnnn= bit utilizado de 0 a 15)
OR	bit	0001nnnn	
AND	bit	0010nnnn	
OUT	bit	0011nnnn	
LDNOT	bit	0100nnnn	
ORNOT	bit	0101nnnn	
ANDNOT	bit	0110nnnn	
OUTNOT	bit	0111nnnn	
ORLD		10000000	
ANDLD		10010000	
END		11111111	

- 4) La asignación de valores nnnn del bit utilizado (campo de dirección) es la siguiente:

Entradas: 0000 a 0011 (0 a 3)
Salidas: 0100 a 0111 (4 a 7)
Internos: 1000 a 1111 (8 a 15)

- 5) El programa de aplicación (el que ejecuta el "autómata") se ubicará en la memoria ram externa a partir de la posición 0, y no ocupará más de 256 bytes (00 a FFh). En un autómata real este programa debería cargarse mediante una consola o instrumento similar. Sin embargo, para simplificar el ejercicio, se cargará un programa de prueba de forma directa mediante una subrutina adecuada.

- 6) El autómata tendrá una entrada de control ONOFF que actuará de la siguiente forma:

ONOFF= 1 El autómata se detiene una vez finalizado el ciclo actual.
ONOFF= 0 El autómata reanuda la ejecución.

Solución.-

Estudio previo.-

La codificación reúne algunas características que facilitan el reconocimiento de la instrucción. Las que llevan NOT, tienen el mismo código básico de operación que sus equivalentes sin NOT, pero con el bit 6 a uno. El campo de dirección de bit corresponde a una entrada si sus bits 3 y 2 son '0', a una salida si son '0' y '1' respectivamente, y a un punto interno (memoria) si su bit 3 es '1'.

Para ejecutar las instrucciones del autómata, se utiliza una pila de 8 bits, en la que se cargan los valores de bit mediante LD y LDNOT, sobre la que se ejecutan las instrucciones ORLD y ANDLD y de donde se toma el valor de bit para OUT y OUTNOT. Esta pila se maneja como un registro de desplazamiento.

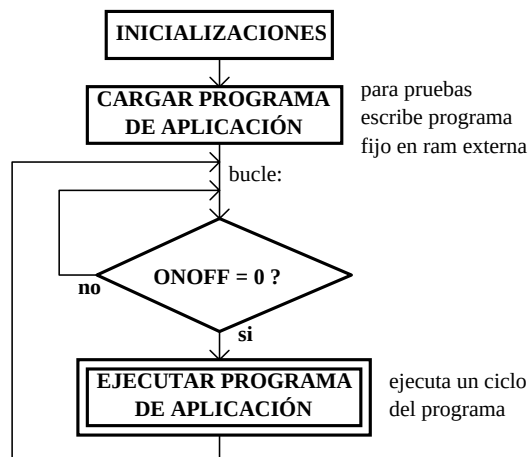
Instrucción	Operaciones a realizar
LD bit	$(pila) \leftarrow bit$
LDNOT bit	$(pila) \leftarrow /bit$
OR bit	$(pila) \leftarrow bit \text{ OR } (pila)$
ORNOT bit	$(pila) \leftarrow /bit \text{ OR } (pila)$
AND bit	$(pila) \leftarrow bit \text{ AND } (pila)$
ANDNOT bit	$(pila) \leftarrow /bit \text{ AND } (pila)$
OUT bit	$bit \leftarrow (pila)$
OUTNOT bit	$bit \leftarrow /(pila)$
ORLD	$(pila) \leftarrow (pila) \text{ OR } (pila+1)$
ANDLD	$(pila) \leftarrow (pila) \text{ AND } (pila+1)$

Asignación de recursos.-

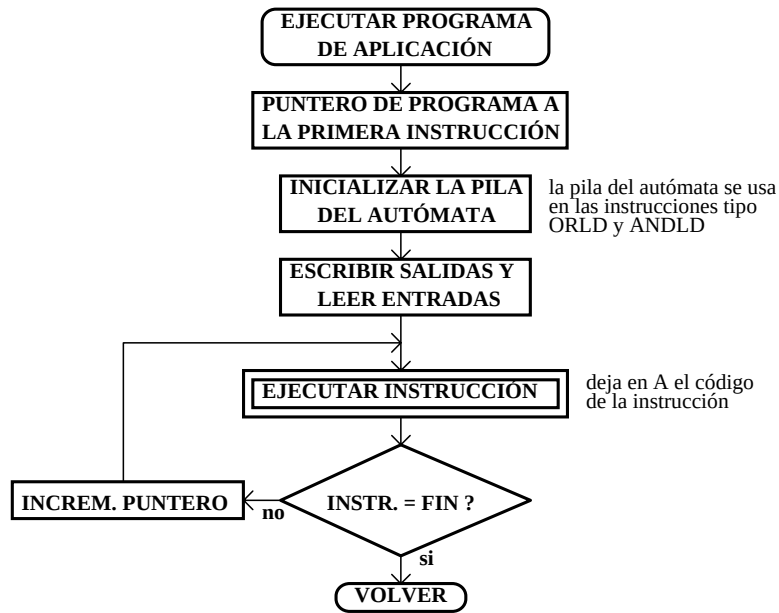
Este sistema no emplea ninguno de los periféricos incorporados en el 8051 a excepción de los puertos de entrada salida.

Diagramas de flujo.-

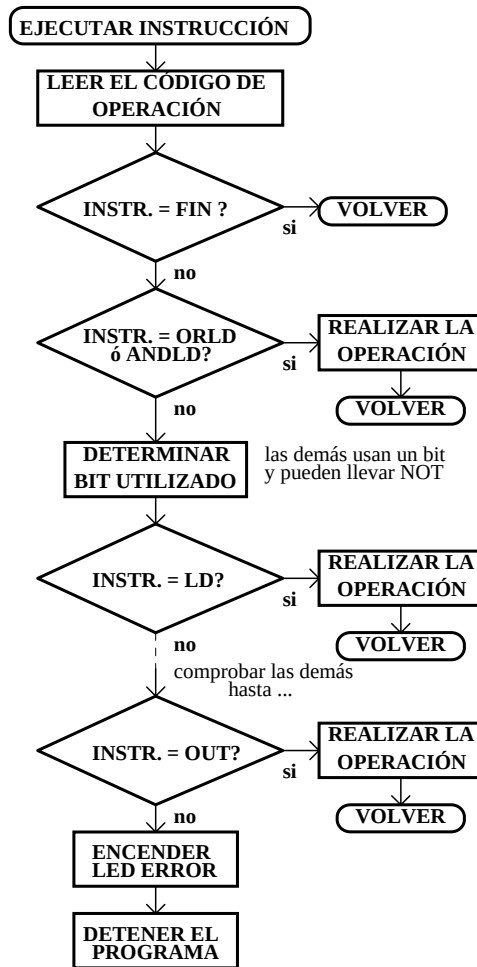
El diagrama de flujo principal es el siguiente:



Cada ciclo de programa (del autómata) se ejecuta según el diagrama de flujo siguiente:



Para ejecutar cada instrucción se sigue el diagrama siguiente:



El archivo CNX puede ser:

```

*      Archivo de configuración del ejemplo del AUTOMATA      *
*
*  Se supone que los circuitos de adaptación de E/S actúan  *
*  como inversores, de forma que una entrada del autómeta a 1*
*  hace que el terminal del uC se ponga a 0. Asimismo, se  *
*  supone que un terminal del uC a 0, pone la salida del  *
*  autómeta a 1      *
*****
Uautomata.src 12.0 256 2048
* Entradas
P10=2
P11=3
P12=4
P13=5
* Salidas
P14=6
P15=7
P16=8
P17=9
* OnOff
P32=10
* Fallo
P33=11
*
CM/P 10 1 0 1
CE0 2 0 1 3
CE1 3 0 1 11
CE2 4 0 1 19
CE3 5 0 1 27
*
LS4 6 0 4 A
LS5 7 0 12 A
LS6 8 0 20 A
LS7 9 0 28 A
*
LERR 1 11 6 R
*
N0 GND
N1 Vdd

```

A continuación se propone un programa como posible solución.

```

; *****
; * Programa que convierte el uC en un autómeta elemental *
; *****
onoff bit p3.2      ; interruptor de control
avisoerr bit p3.3   ; indicador de error
entsal data p1      ; puerto e/s P1
copiaes data 20h     ; copia de P1 en RAM (direccionable
;                   a nivel de bit)
interno data 21h     ; puntos internos en ram
pila data 22h        ; pila de 8 bits para instrucciones
del tipo ORLD
pospila data 23h     ; altura de la pila del autómeta
nbit data 24h        ; variable que contiene un nº de bit
inv bit 26h.0        ; variable de bit usada en leerbit
;
y escribit

```

```

programa equ 0      ; origen del programa de aplicación en xram
; códigos de operación básicos
_ld equ 0
_or equ 1
_and equ 2
_out equ 3
_orld equ 8
_andld equ 9
_fin equ 0fh
;
; Entrada de RESET
org 0
jmp inicio
;
org 100h
inicio:
mov sp,#60h
call rxprog
; inicializaciones
mov entsal,#0fh      ; salidas reales a cero ('1' en las
                     ; entradas)
mov copiaes,#0       ; copia interna de entradas y salidas a cero
bucle:
jb onoff,$
call run
jmp bucle
run:
; la comprobación de la pila se hace en poner y quitar
mov pila,#0
mov pospila,#0
mov dptr,#programa
; actualiza ent/sal
call actentsal
run2:
call ejecutar
cjne a,#_fin,run3
ret
run3:
inc dptr
jmp run2
;
actentsal:
; escribir salidas
mov a,copiaes
orl a,#0fh           ; en las entradas escribe unos
mov entsal,a
; leer entradas
mov copiaes,entsal ; las salidas no cambian
ret
;
; version con programa fijo para pruebas
rxprog:
mov dptr,#programa
mov a,#0             ; LD 0
movx @dptr,a
inc dptr
mov a,#11h           ; OR 1
movx @dptr,a

```

```

inc dptr
mov a,#34h ; OUT 4
movx @dptr,a
inc dptr
mov a,#0 ; LD 0
movx @dptr,a
inc dptr
mov a,#21h ; AND 1
movx @dptr,a
inc dptr
mov a,#35h ; OUT 5
movx @dptr,a
inc dptr
mov a,#42h ; LD NOT 2
movx @dptr,a
inc dptr
mov a,#36h ; OUT 6
movx @dptr,a
inc dptr
mov a,#02h ; LD 2
movx @dptr,a
inc dptr

mov a,#03h ; LD 3
movx @dptr,a
inc dptr
mov a,#90h ; AND LD
movx @dptr,a
inc dptr
mov a,#77h ; OUT NOT 7
movx @dptr,a
inc dptr
mov a,#0f0h ; FIN
movx @dptr,a
ret
;
ejecutar:
movx a,@dptr ; busqueda
mov b,a ; guarda instr.
swap a ; ver código operación
anl a,#0fh
cjne a,#_fin,no_fin
ret ; si es fin, volver
no_fin:
cjne a,#_orld,no_orld
call quitar ; desplaza la pila
orl c,pila.0
mov pila.0,c
ret
no_orld:
cjne a,#_andld,no_andld
call quitar ; desplaza la pila
anl c,pila.0
mov pila.0,c
ret
no_andld:
jb acc.3,error ; cod. op. no existe
; las demas usan un punto (marca, bit, etc.)

```

```

push acc      ; salva cod. op.
mov a,b
anl a,#7
jb acc.3,inter
mov r0,#copiaes ; R0 apunta al byte copiaes
jmp seguir
inter:
mov r0,#interno ; R0 apunta al byte interno
seguir:
mov nbit,a      ; nbit indica el nº del bit (0-7)
pop acc         ; reponer cod. op.
mov c,acc.2     ; c=1 indica NOT
mov inv,c       ; prepara inv para que lo usen leerbitac y
escribitac
anl a,#3        ; refunde con y sin NOT en el código básico
cjne a,#_ld,no_ld
call leerbit    ; c= @R0.nbit si inv=0
;               c= not(@R0.nbit) si inv=1
call poner
ret
no_ld:
cjne a,#_or,no_or
call leerbit
orl c,pila.0
mov pila.0,c ; no desplaza la pila
ret
no_or:
cjne a,#_and,no_and
call leerbit
anl c,pila.0
mov pila.0,c ; no desplaza la pila
ret
no_and:
cjne a,#_out,no_out
; out (se verifica que no es (R0=#copiaes y nbit<4))
mov a,R0
cjne a,#copiaes,noesent
jb nbit.2,noesent
jmp error      ; se intentó escribir en una entrada
noesent:
mov c,pila.0 ; trae el bit
call escribit
ret
no_out:
ret
error:
clr avisoerr   ; enciende led error
jmp $          ; y se detiene
;
; *****
; * lectura de bit *
; *****
; lee el bit dado en nbit del byte apuntado por R0
; y lo deja en el bit de acarreo
leerbit:
mov a,nbit
rl a
rl a           ; a = bit * 4

```

```

push acc      ; dir low
mov a,# HIGH tablalec
; con UVIENS51 sustituir por mov a,#[tablalec
push acc      ; dir high
mov a,@r0
ret           ; salto a la dirección dada
lecvolver:
jnb inv,nonegar
cpl c
nonegar:
ret
org 400h      ; origen al comienzo de una página de 256 bytes
tablalec:
mov c,acc.0
ajmp lecvolver
mov c,acc.1
ajmp lecvolver
mov c,acc.2
ajmp lecvolver
mov c,acc.3
ajmp lecvolver
mov c,acc.4
ajmp lecvolver
mov c,acc.5
ajmp lecvolver
mov c,acc.6
ajmp lecvolver
mov c,acc.7
ajmp lecvolver
;
;*****
;
;* escritura de bit *
;*****
; copia el contenido del acarreo C en el bit dado en nbit
; del byte apuntado por R0
escribit:
jnb inv,nonegar2
cpl c
nonegar2:
mov a,nbit
rl a
rl a          ; a = bit * 4
push acc      ; dir low
mov a,# HIGH tablaesc
; con UVIENS51 sustituir por mov a,#[tablaesc
push acc      ; dir high
mov a,@r0
ret           ; salto a la dirección dada
escvolver:
mov @r0,a
ret
;
org 500h      ; origen al comienzo de una página de 256 bytes
tablaesc:
mov acc.0,c
jmp escvolver
mov acc.1,c
jmp escvolver

```

```

mov acc.2,c
jmp escvolver
mov acc.3,c
jmp escvolver
mov acc.4,c
jmp escvolver
mov acc.5,c
jmp escvolver
mov acc.6,c
jmp escvolver
mov acc.7,c
jmp escvolver
;
poner:
mov a,pila
rlc a
mov pila,a
inc pospila
mov a,pospila
anl a,#0f8h          ; <=7
jz bien
jmp error            ; desbordamiento de la pila
; puede hacerse jmp error porque allí se detiene en jmp $
;
quitar:
mov a,pila
rrc a
mov pila,a
dec pospila
mov a,pospila
anl a,#0f8h          ; <=7
jz bien
jmp error            ; desbordamiento de la pila
bien:
ret
;
end

```

Ejercicio 4.-

Se desea escribir un conjunto de subrutinas para controlar una pantalla LCD de 2 líneas por 16 columnas. La pantalla se conecta al microcontrolador mediante un enlace serie síncrono compuesto de 3 señales denominadas CLK (P1.0), DAT (P1.1) y LOAD (P1.2).

La pantalla reconoce los siguientes comandos:

0 d7 d6 d5 d4 d3 d2 d1 d0

Escribe carácter d7...d0 (ASCII ext.) en la posición actual del cursor, en modo normal o de inserción (ver comandos 1100 y 1101).

1 0 0 a4 a3 a2 a1 a0

Desplaza el cursor a la posición a4...a0. (0= inicio, 31= fin)

1 0 1 0

Oculto el cursor

1 0 1 1

Muestra el cursor

1 1 0 0

Activa modo INSERTAR

1 1 0 1

Desactiva modo INSERTAR

1 1 1 0

Borra el carácter apuntado por el cursor y desplaza los que se encuentran a su derecha.

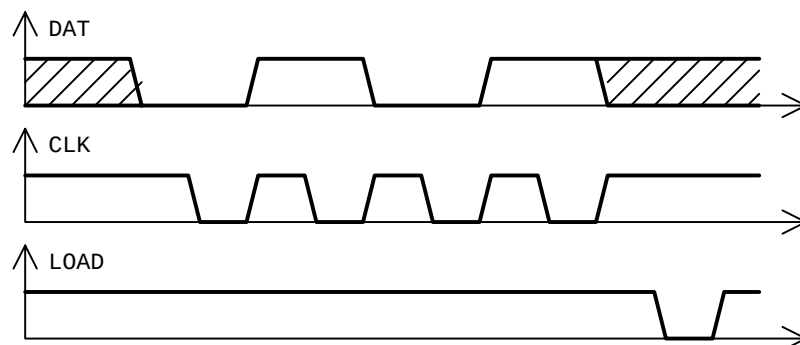
1 1 1 1

Borra toda la pantalla y pone el cursor en la dirección 0 (al inicio).

El envío de todos los comandos debe empezar con el bit menos significativo. Por ejemplo, para ocultar el cursor se envían los bits:

1 0 1 0

empezando por el 0 de la derecha. El siguiente cronograma muestra la evolución de las señales. Los flancos activos de CLK y LOAD son los de bajada.



a) Escribir una subrutina **enviar** que genere las señales DAT, CLK y LOAD a partir del comando que se desee enviar. Se utilizarán las siguientes variables:

tipo Bit más significativo del comando. Es 0 para "escribir carácter" y 1 para todos los demás comandos.

nbits Byte que contiene el número de bits del comando menos 1.

valor Byte que contiene el comando (excepto el bit más significativo) ajustado a la derecha.

La subrutina debe enviar a la pantalla LCD los **nbits** menos significativos de **valor**, empezando por el menos significativo. A continuación debe enviar el bit **tipo** y finalizar con la carga del comando (impulso de la señal LOAD). Por ejemplo, cuando el programador desee enviar el comando "ocultar cursor", cargará en las variables los valores:

```
tipo= 1      (setb tipo)
valor=2      (mov valor,#010b)
```

nbits= 3 (mov nbits,#3)

y llamará a la subrutina ocultar (call ocultar).

b) Escribir una subrutina para cada tipo de comando disponible. Estas subrutinas llamarán a la subrutina **enviar**. En el caso del comando "escribir carácter", se supone que el programador pasa el ASCII en el acumulador. En el caso del comando "desplazar cursor", se supone que el programador pasa la posición del cursor (0 a 31) en una variable denominada **posicion**.

c) Escribir una subrutina que envíe el texto apuntado por DPTR (en ROM) a la posición actual del cursor. El texto termina con el valor 0, que no se debe enviar. Esta rutina llamará a la del comando "escribir carácter" del apartado b.

Solución.-

a) Se indica una posible solución para la subrutina enviar.

```
; Envía los nbits menos significativos de valor y finaliza
; con el bit tipo
;
; Conexiones al LCD
clk bit P1.0
dat bit P1.1
load bit P1.2
;
; Variables empleadas
valor data 40h
nbits data 41h
tipo bit 0
;
; Código
enviar:
push acc
push psw
mov a,valor
send2:
rrc a
mov dat,c
clr clk
setb clk
djnz nbits,enviar2
mov c,tipo
mov dat,c
clr clk
setb clk
clr load
setb load
pop psw
pop acc
ret
```

b) Se indican solamente dos subrutinas; las demás son muy semejantes.

```
ocultarcursor:
mov valor,#02h
mov nb,#3
setb tipo
call send
ret
```

```
; escribe el byte dado en A, en la posición actual del cursor
ponchar:
mov valor,a
mov nbits,#8
clr tipo
call send
ret
```

c) Se indica una posible solución.

```
; Envía la cadena ubicada en ROM, apuntada por dptr
; (terminada en 0)
; a la posición actual del cursor en la pantalla LCD (0 a 31)
poncad:
push acc
clr a
poncad2:
push acc
movc a,@a+dptr
jz poncad3
call ponchar
pop acc
inc a
jmp poncad2
poncad3:
pop acc
pop acc
ret
;
```