TERCER CURSO. INFORMÁTICA INDUSTRIAL II

Escuela Politécnica Superior
Universidad de Huelva

Tema 2
Modelo de Programación

Manuel Sánchez Raya
Versión 0.8
24 de Noviembre de 2001

ÍNDICE

BIBLIOGRAFÍA:

Apuntes de Ingeniería Técnica Industrial. Universidad de Málaga.

Microcontroladores MCS-51 y MCS-251. José Matas Alcalá, Rafael Ramón Ramos Lara

Modelo de Programación.

El modelo de programación no es más que el empleo combinado de las instrucciones y del conocimiento de la arquitectura interna del microcontrolador para diseñar rutinas que den solución a problemas comunes en la mayor parte de los diseños, como la transferencia de bloques de datos, realización y consulta de tablas de datos, conversión de datos entre distintos formatos, rutinas aritméticas, etc.

1.- Creación y consulta de tablas.

En la programación de microcontroladores es frecuente utilizar **tablas de datos** en ciertos casos, como en la conversión entre distintos formatos de datos, como de hexadecimal a ASCII, de binario a 7 segmentos, etc. En estos casos se genera una tabla específica, donde cada elemento de la tabla esté relacionado con el dato de entrada que se quiere convertir, de forma que el mismo dato de entrada se emplea como índice para obtener el resultado de la conversión.

Mediante tablas se puede también obtener números primos según un índice o evitar el cálculo de funciones matemáticas no lineales como las funciones trigonométricas o de cualquier otro tipo.

Como ejemplo se muestra la rutina de conversión hexadecimal (00H-0FH) al formato ASCII; para ello la tabla siguiente muestra la correspondencia entre ambos formatos.

Binario	00H	01H	02H	03H	04H	05H	06H	07H	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
ASCII	30H	31H	32H	33H	34H	35H	36H	37H	38H	39H	41H	42H	43H	44H	45H	46H

```
;************************************************
; Subrutina Bin_ASCII
; Convierte un número hexadecimal en ASCII
; Entrada y Salida:     A
;************************************************
BIN_ASCII:  INC   A
            MOVC  A,@A+PC
            RET
            DB    30H,31H,32H,33H,34H,35H,36H,37H
            DB    38H,39H,41H,42H,43H,44H,45H,46H
```

El núcleo de esta subrutina es **la instrucción MOVC A,@A+PC**. Cuando se ejecuta esta instrucción el contador de programa se actualiza para apuntar a la instrucción siguiente, RET, pues la tabla de datos está situada a continuación de la instrucción RET. El acumulador es un índice que apunta al dato en concreto dentro de la tabla. Si A vale 0H la subrutina retornará el dato 30H, puesto que en primer lugar se ha incrementado en una unidad (instrucción INC A) y, por tanto, la suma del puntero indexado, A+PC, apunta al primer dato de la tabla; si A vale 01H la subrutina retorna 31H, si vale 02H retorna 32H, etc.

2.- Transferencia de bloques de datos.

Resulta frecuente tener que **trasladar bloques de datos entre distintas zonas de la memoria**. De estos bloques se suele tener su dirección base y su tamaño, aunque la dirección de destino puede ser fija o variable, dependiendo del tipo de aplicación.

La transferencia del bloque de datos se realiza en la memoria externa de datos, y tiene un único puntero de 16 bits que consiste en el registro @DPTR. Con este puntero la solución del problema resulta difícil, puesto que para trasladar un bloque de datos se necesitan al menos **dos punteros de 16 bits**, uno de origen y otro de destino. Para salvar este obstáculo, se puede tratar de recurrir al uso de cualquiera de los dos punteros restantes de 8 bits, @R0 y @Rl, que se pueden emplear con la instrucción MOVX. Estos punteros sitúan su valor en el puerto P0, byte

bajo del bus de direcciones, cuando se ejecutan las instrucciones de lectura y de escritura en la memoria de datos, (MOVX A,@Ri y MOVX @Ri,A).

Mientras tanto, en el puerto P2, byte alto del bus de direcciones, se sitúa el contenido del registro P2 del área de SFR. Por tanto, basta con colocar el byte alto de la dirección de destino del bloque de memoria en el registro P2 para que salga por el puerto P2. A continuación se muestra una rutina que transfiere un bloque de 50 bytes desde la dirección origen 1200H a la dirección destino 3500H.

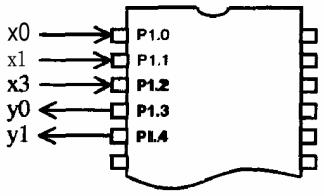
```
;************************************************
;Rutina TRANS (transferencia de datos)
;Punteros: DPTR, R0
;Tamaño: R7
;Modifica: DPTR, R0, R7, P2, A
;************************************************
TRANS:      MOV   DPTR,#1200H       ;Dirección inicio
            MOV   R0,#00H           ;Dirección destino
            MOV   P2,#35H
            MOV   R7,#50
B_TR:       MOVX  A,@DPTR           ;Lee en origen
            MOVX  @R0,A             ;Escribe en destino
            INC   DPTR              ;Incrementa punteros
            INC   R0
            DJNZ  R7,B_TR           ;Bucle R7 veces
```

La transferencia de datos, según el ejemplo mostrado, está **limitada a una misma página**, es decir, a todas aquellas direcciones de 16 bits que tienen el mismo valor del byte alto de dirección, pues en el puerto P2 se mantiene este valor mientras se ejecuta la rutina. Si en esta rutina el primer byte a transferir del origen tiene como byte bajo de dirección el valor 00H, entonces el tamaño máximo de datos que se pueden transferir es de 256 bytes; en caso contrario el tamaño máximo que se puede transferir vendrá delimitado por el inicio de la página siguiente.

3.- Funciones booleanas

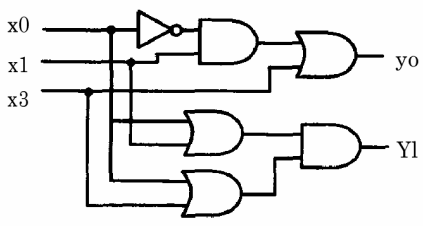
Los microcontroladores de la familia MCS-51 están capacitados para realizar **funciones booleanas**, tanto a nivel de bit como a nivel de byte. A nivel de bit, el microcontrolador puede realizar funciones lógicas y realizar otras tareas como leer el estado de un teclado, activar un relé, etc.

A continuación se expone un ejemplo de aplicación de las instrucciones booleanas que consiste en la evaluación de las funciones lógicas Y0 e Y1, con tres variables de entrada cada una, X0, X1 y X2. Las entradas se han conectado a los terminales P1.0, P1.l y P1.2, y las funciones de salida se han conectado a P1.3 y P1.4.




```
;*********************************
;Rutina     F_BOL
;Entradas:  P1.0, P1.1, P1.2
;salidas:   P1.3, P1.4
;Modifica:  C, B.0
;*********************************
            EQU   P1.0  ;entradas
            EQU   P1.l
            EQU   P1.2
            EQU   P1.3  ;salidas
            EQU   P1.4
F_BOL:      MOV   C,X0
```

Funciones lógicas



```
ANL   C,Xl
ORL   C,X0
MOV   Y0,C  ;obtiene Y0
MOV   C,X0
ORL   C,Xl
MOV   B.0,C
MOV   C,X0
ORL   C,X2
ANL   C,B.0
MOV   Y1,C  ;obtiene Y 1
SJMP  F_BOL
```

4.- Retardos de tiempo.

Para realizar retardos de manera precisa se pueden emplear los recursos hardware del microcontrolador, como los temporizadores Timer0, Timerl y Timer2; en ese caso se deben configurar y utilizar interrupciones de manera adecuada. Por otro lado, también se pueden realizar retardos mediante **rutinas de software,** para aquellos casos en los que no se requiere de precisión en los tiempos generados; pueden entonces destinarse los temporizadores a otras tareas más relevantes. De todas formas, el uso de retardos mediante software implica una programación poco eficiente del microcontrolador, puesto que éste pasa una buena parte de su tiempo realizando el retardo.

Un retardo por software se hace básicamente a través de uno o varios **bucles anidados**, donde el número de anidamientos depende del tiempo de retardo requerido. Como ejemplo se propone una rutina de retardo con dos anidamientos, donde el número de veces que se ejecuta cada bucle depende de los registros R7 y R6.

```
;***********************************************
;Subrutina RETARDO
;Registros afectados: R7, R6
;***********************************************
RETARDO:    MOV   R7,#10
Ll:         MOV   R6,#100
L2:         DJNZ  R6,L2       ;Bucle interno
            DJNZ  R7,Ll       ;Bucle externo
            RET
```

El bucle interno formado por el registro R6 esta dentro del bucle externo formado por el registro R7; a cada vuelta del bucle externo le corresponden 100 ejecuciones del bucle interno. Sabiendo que el tiempo de ejecución de las instrucciones MOV y DJNZ es de uno y dos ciclos maquina, respectivamente, y que un ciclo máquina equivale a 1 us con una frecuencia de reloj de 12 MHz, se puede determinar el tiempo exacto que tardará en ejecutarse la subrutina de retardo. La figura siguiente muestra el diagrama de tiempos de la subrutina e indica también el tiempo resultante.

$$t_R = 1 + \left[1 + 2 \cdot R6 + 2\right] \cdot R7 + 2 = 2.033 \mu s$$

$$t_R = \frac{1}{3}\mu s + \left[\frac{1}{3}\mu s + (R6-1)\mu s + \frac{1}{2}\mu s\right] \cdot R7 + \\ + (R7-1)\mu s + \frac{1}{2}\mu s + 1\mu s = 1.009{,}16\mu s$$

El tiempo máximo que se puede obtener con una rutina de dos bucles anidados es de 130.818us, que equivale a unos 0.13 segundos, poniendo a FFH los registros R6 y R7. Si se desean obtener tiempos mayores es necesario realizar más anidamientos. Una alternativa a la rutina anterior consiste en la siguiente rutina:

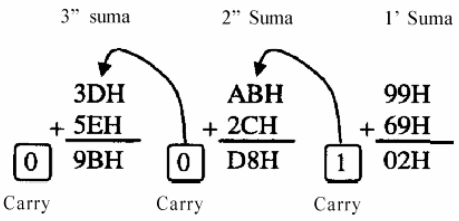
```
;***********************************************
;Subrutina RETARDO2
;Registros afectados: R7, R6
;***********************************************

RETARDO2:   MOV   R7,#0
            MOV   R6,#0
Ll:         DJNZ  R6,Ll       ;Bucle interno
            DJNZ  R7,Ll       ;Bucle externo
            RET
```

A primera vista da la impresión de que los bucles de esta rutina dan cero vueltas, pues R6 y R7 valen cero. Pero si se observa en detalle la ejecución de la instrucción DJNZ, resulta que esta instrucción **primero decrementa el registro indicado, y luego examina la condición de salto**. Por tanto, si el registro vale cero, al decrementarse pasa a valer FFH, (00H-0lH=FFH), y se ejecuta el bucle FFH+1 o 256 veces. Realizando el diagrama de tiempos para esta rutina, el tiempo que se obtiene es de 13.1588us para una frecuencia de reloj de 12MHz.

5.- Suma y resta de datos.

La suma y resta de datos es sencilla de hacer mediante las instrucciones ADD, ADDC y SUBB, cuando los datos que intervienen son de 1 byte, pero resulta más complicada cuando el tamaño de los datos es mayor de 1 byte, es decir, cuando se tienen que sumar datos de 2,3 ó 4 bytes de tamaño.

```
;***********************************************
; Subrutina SUMA_N (Suma de números de N bytes)
; Entrada Punteros: R0 a X, Rl a Y. R7: Tamaño
; Salida: El resultado sustituye al operando X
; Modifica: A, R0, Rl, R7, C
;***********************************************

SUMA_N:     CLR   C           ;Borra acarreo
BUC_SUM:    MOV   A,@R0       ;Lee operando X
            ADDC  A,@Rl       ;Suma X con Y
            MOV   @R0,A       ;Resultado sustituye a X
            INC   R0          ;Incrementa punteros
            INC   Rl
            DJNZ  R7,BUC_SUM  ;Repite 7 veces
            RET
```



Para resolver este problema se realiza una subrutina que es capaz de sumar datos de cualquier tamaño en bytes, aunque, para este caso, será suficiente con que sume datos de 3 bytes de tamaño. Los datos que se deberán sumar son X = 2DAB99H e Y = 5E2C69H en formato hexadecimal y están almacenados a partir de las posiciones de la memoria interna 60H y 70H, respectivamente; es decir, el byte de menor peso del dato X (99H) estará ubicado en la dirección 60H, su byte intermedio (ABH) en la 61H y su byte más significativo (2DH) en la 62H; lo mismo se hace con el dato Y a partir de la dirección 70H de la memoria interna.

Para ejecutar esta subrutina se tienen primero que poner las direcciones 60H y 70H en los registros R0 y Rl; ello se debe efectuar antes de llamar a la subrutina, por ejemplo:

```
            MOV   R0,#60H     ;Pone dirección de X en RO
            MOV   R1,#70H     ;Pone dirección de Y en R1
            MOV   R7,#3       ;Pone tamaño del dato en R7
            LCALL SUMA_N      ;Llama a la subrutina
```

La manera de operar de la subrutina se muestra claramente en la figura anterior. La subrutina hace tres sumas consecutivas y obtiene los datos en cada una de ellas a través de los punteros @R0 y @Rl, que se actualizan con la instrucción INC. Esta subrutina es capaz de sumar datos de 4, 8 ó 16 bytes, con la condición previa de situar los datos en la memoria interna y poner en R0, Rl y R7 los valores adecuados.

La razón de que el resultado generado en las operaciones sustituya a uno de los operandos, es porque el 8051 tan sólo se dispone de los punteros @R0 y @R1; se carece de un tercer puntero necesario para dejar el resultado en otras direcciones. El resto de los registros no pueden ser empleados como punteros, pero sí como variables intermedias, pasando su contenido a R0 o R1, y conseguir, así, que hagan de punteros por medio de estos registros.

Si el formato de los números que se quiere sumar es BCD, basta con añadir la instrucción de ajuste decimal, DA A, tras la instrucción de suma ADDC, para obtener el resultado también en formato BCD, lo que se muestra en la siguiente subrutina. En este caso, si X=1 969H y Y=1050H dará por resultado Z=0919H.

```
;***********************************************
; Subrutina SUMA_N (Para datos en formato BCD)
; Entrada Punteros: R0 a X, Rl a Y. R7: Tamaño
; Salida: El resultado sustituye al operando X
; Modifica: A, R0, RI, R7, C
;***********************************************

SUMA_N:     CLR   C           ;Borra Carry
BUC_SN:     MOV   A,@R0       ;Lee dato X
            ADDC  A,@Rl       ;Suma X con Y
            DA A
            MOV   @R0,A       ;Resultado sustituye a X
            INC   R0          ;Incrementa punteros
            INC   R1
            DJNZ  R7,BUC_SN   ;Repite R7 veces
            RET
```

En cuanto a la resta de datos de longitud mayor que un byte, se puede realizar una subrutina con este propósito simplemente reemplazando la instrucción de suma por la instrucción de resta, SUBB, en las subrutinas anteriores.

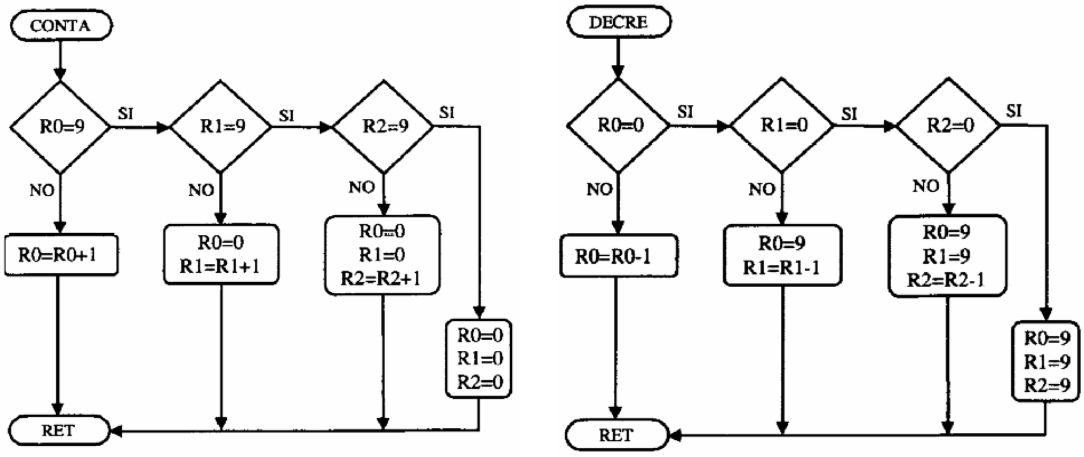
6.- Contador en BCD.

Es habitual en muchas aplicaciones realizar rutinas que hagan la función de **contador** en formato binario o BCD. Estas rutinas tienen por objetivo contabilizar objetos, eventos, pulsos de una señal cuadrada, etc. Los contadores en formato binario son sencillos de realizar, ya que hay instrucciones específicas para ello como INC, DEC, ADD, ADDC y SUBB; por el contrario, los contadores en formato BCD no son tan evidentes de hacer, pues la ALU del microcontrolador contiene un sumador binario, pero no un sumador en base decimal. Por otro lado, debe tenerse en cuenta que en aquellas aplicaciones donde el resultado de la cuenta se muestra a un operario, éste **debe aparecer en formato decimal**, para que se pueda interpretar de manera conveniente.

El siguiente ejemplo muestra una subrutina para realizar un contador en BCD de cuatro cifras, es decir, que el máximo valor que puede alcanzar es 999. Cada cifra del contador esta contenida en un registro: las unidades están en el registro R0, las decenas en R1 y las centenas en R2. De esta forma, para visualizar posteriormente el valor del contador, sólo se ha de volcar cada uno de los registros en el visualizador' numérico correspondiente, ya sea de tipo LED o LCD.

```
;***********************************************
```

```
; Subrutina CONTA
; Registros afectados: R0, R1, R2
;*********************************************

CONTA:      CJNE  R0,#9,UNIDAD      ;Compara unidad
            CJNE  R1,#9,DECENA      ;Compara decena
            CJNE  R2,#9,CENTENA     ;Compara centena
            MOV   R0,#0             ;Al ser 999 pone a 000
            MOV   R1,#0
            MOV   R2,#0
            RET
UNIDAD:     INC   R0                ;Incrementa unidades
            RET
DECENA:     MOV   R0,#0             ;Pone a cero unidades
            INC   R1
            RET
CENTENA:    MOV   R0,#0
            MOV   R1,#0
            INC   R2
            RET
```



En el flujograma de la subrutina CONTA, se observa cómo primero se comprueba si las unidades han llegado a 9; de no ser así se incrementa R0. En caso afirmativo, se incrementan las decenas y las unidades se ponen a cero. Procediendo de esta manera con el resto de dígitos, se completa la subrutina del contador, que se puede extender fácilmente a un mayor número de dígitos.

A la hora de hacer una subrutina para descontar, el proceso es idéntico al mostrado en la figura, pero cambiando la condición existente por la de comprobar si las unidades, decenas y centenas son iguales a cero; los registros, en lugar de ponerse a cero, se ponen a 9, tal y como se indica en la figura. A continuación se muestra la subrutina para descontar:

```
;*********************************************
; Subrutina DECRE
; Registros afectados: R0, R1, R2
;*********************************************

DECRE:      CJNE  R0,#0,D_UNI ;Compara unidad
            CJNE  R1,#0,D_DECE      ;Compara decena
            CJNE  R2,#0,D_CENT      ;Compara centena
            MOV   R0,#9             ;Al ser 000 pone a 999
            MOV   R1,#9
            MOV   R2,#9
            RET
```

```
D_UNI:      DEC   R0              ;Decrementa anidad
            RET
D_DECE:     MOV   R0,#9           ;Pone a 9 unidad
            DEC   R1              ;Decrementa decena
            RET
D_CENT:     MOV   R0.#9           ;Pone a 9 unidad
            MOV   R1,#9           ;Pone a 9 decena
            DEC   R2              ;Decrementa centena
            RET
```

En el caso de producirse un rebosamiento de los valores máximo y mínimo en las subrutinas CONTA y DECRE, respectivamente, se ha optado por actualizar el valor de todos los registros al valor inicial 000 y 999, respectivamente.

Otra manera de realizar un contador en formato BCD, consiste en emplear de manera hábil la **instrucción de ajuste decimal DA A**. Para ello, supóngase que se desea realizar un contador de cuatro cifras en formato BCD, y que las posiciones de memoria interna 40H y 41H harán de contador; los cuatro bits bajos de 40H serán las unidades y sus cuatro bits altos, las decenas, los cuatro bits bajos de 41H serán las centenas y sus cuatro bits altos la unidad de millar del contador. Con un contador de cuatro cifras en las posiciones 40H y 41H, se puede llegar hasta un valor de 9999, es decir, ambas posiciones contienen el valor 99H. A continuación se muestra la rutina CONTA que hace esta función:

```
;*****************************************************
; Subrutina CONTA
; Registros afectados: A, R0, R7, C, (40H), (41H)
;*****************************************************

CONTA:      MOV   R7,#2       ;R7 controla el bucle
            MOV   R0,#40H     ;Pone dirección en puntero
            SETB  C           ;Pone C=l
BUC_CT:     MOV   A,@R0       ;Lee
            ADDC  A,#0        ;suma con acarreo
            DA    A           ;Ajuste decimal
            MOV   @R0,A       ;Guarda
            INC   R0          ;Incrementa puntero
            DJNZ  R7,BUC_CT   ;Repite R7 veces
            RET
```

Esta rutina incrementa en una unidad el valor de las posiciones 40H y 41H. La rutina, al principio, incrementa 40H de 00 H a 01H, de 01H a 02H, y así sucesivamente hasta llegar a 09H, pues la instrucción de ajuste decimal no modifica el valor del acumulador tras la instrucción de suma. En el siguiente incremento el valor de 40H pasa de 09H a 0AH, por lo que la instrucción DA A transforma este dato en 10H, al ser el nibble bajo mayor que 9. En los siguientes incrementos la instrucción DA A no modificará el acumulador, al menos hasta que llegue al valor de 1AH, momento en el cual lo transformará en 20H. Así, la instrucción de ajuste decimal actúa en los instantes en que el acumulador vale lAH, 2AH, 3AH, ... 9AH; este último caso lo transforma en 00 H y pone el bit de acarreo a uno, C=l, de manera que incrementa en una unidad el contenido de la posición de 41H, pasando el contador a valer 0100H.

La rutina CONTA basada en la instrucción de ajuste decimal tiene un menor tamaño que la anterior y, además, permite extender el máximo valor del contador a cualquier tamaño; basta, para ello, con aumentar el número de bytes del contador, aumentado el número de posiciones de memoria que ocupa. Por el contrario, no es posible realizar del mismo modo una rutina que decremente el valor del contador, debido a que la instrucción de ajuste decimal no puede ir asociada a la instrucción SUBB de resta.

7.- Multiplicación y división de datos de 16 bits

Para llevar a cabo la multiplicación de dos números de 16 bits se deben realizar primero dos multiplicaciones de un número de 8 bits por otro de 16 bits (XL por YH-YL, y XH por YH-YL), que proporcionan dos resultados intermedios de 24 bits, ZN2-ZN0 al multiplicar XL por YH-YL y ZM2-ZM0 al multiplicar XH por YH-YL. Estos resultados intermedios se suman para obtener un resultado final de 32 bits (ZS3-ZS0). En consecuencia, la multiplicación de dos números de 16 bits estará basada en una subrutina, MUL8_16, que multiplica un número de 8 bits con un número de 16 bits.

La subrutina MUL8_16 tiene un primer operando XL de 8 bits almacenado en R1 y un segundo operando de 16 bits; el byte alto está almacenado en R2 y el byte bajo en R3. La subrutina genera un resultado de 4 bits que se sitúa en los registros R5, R6 y R7, donde se almacenan los resultados ZT2, ZT1 y ZT0, respectivamente.

```
;*******************************************
; Subrutina MUL8_16 (8bit x 16bit, XL x YH-YL)
; Entrada:  Rl=XL, R2=YH, R3=YL.
; Salida:   24 bits. R7=ZT0, R6=ZT1 y R5=ZT2.
; Modifica: A, B, C
;*******************************************

MUL8_16:    MOV   A,R1        ;Lee XL
            MOV   A,R3        ;Lee YL
            MUL   AB          ;(XL x YL)
            MOV   R7,A        ;Salva ZT0
            MOV   R6,B        ;Salva ZN1
            MOV   A,R1        ;Lee XL
            MOV   B,R2        ;LeeYH
            MUL   AB          ;(XL x YH)
            ADD   A,R6        ;(ZM0+ZN1)
            MOV   R6,A        ;Salva ZTl
            CLR   A           ;Borra A
            ADDC  A,B         ;(ZMl+acarreo)
            MOV   R5,A        ;Salva ZT2
            RET
```

La subrutina MULl6 multiplica dos datos de 16 bits cada uno. El primer y el segundo operando deben estar ubicados en los registros R0, Rl, R2 y R3, donde se sitúan XH, XL, YH e YL, respectivamente. La subrutina genera un resultado de 32 bits, ZS0, ZSl, ZS2 y ZS3, que se coloca en los registros R7, R6 , R5 y R4, respectivamente.

```
;*********************************************
; Subrutina MULl6 (l6bits x 16 bit, YH-YL x XH-XL)
; Subrutina R0=XH, Rl=XI_, R2=YH, R3=YL
; Salida: 32 bits. R7=ZS0, R6=ZSl, R5=ZS2, R4=ZS3
; Modifica: A, B, Registros R7 a Rl del banco 1
;*********************************************

MULl6:      LCALL MUL8_16     ;(XL x YH-YL)
            SETB  RS0         ;Cambia al banco 1
            MOV   Rl,00H      ;Lee XH
            MOV   R2,02H      ;Lee YH
            MOV   R3,03H      ;Lee YL
            LCALL MUL8_16     ;(XH x YH-YL)
            CLR   C           ;Borra acarreo
            CLR   RS0         ;Cambia a banco 0
            MOV   A,R6        ;Lee ZN1
```

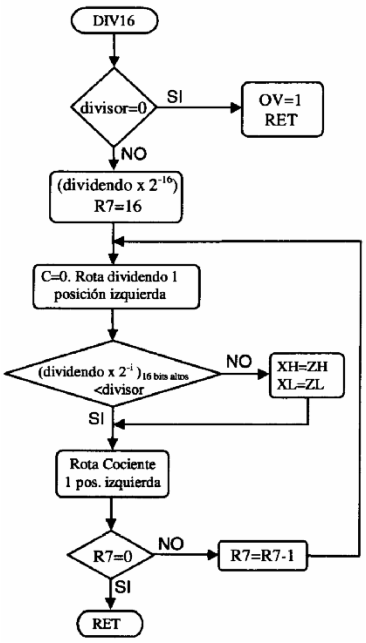
```
ADD   A,0FH       ;(ZN1+ZM0)
MOV   R6,A        ;Salva ZS1
MOV   A,R5        ;Lee ZN2
ADDC  A,0EH       ;(ZN2+ZM1+C)
MOV   R5,A        ;Salva ZS2
CLR   A           ;Borra A
ADDC  A,0DH       ;(ZM2+C)
MOV   R4,A        ;Salva ZS3
RET
```

La subrutina MUL16 utiliza el banco 0 y el banco 1 de registros para almacenar los resultados intermedios de la subrutina MUL8_16. La primera multiplicación, XL x YH-YL, utiliza el banco 0 de registros y la segunda multiplicación, XH x YH-YL, utiliza el banco 1 de registros. Se cambia de banco de registros con el bit RS0 del registro de estado PSW. Al final de la subrutina MUL16 se trabaja con los registros del banco 0 y se accede a los registros del banco 1 mediante direccionamiento directo.

Para usar la subrutina MUL16 se debe inicializar el puntero de la pila, SP, al menos con una dirección superior a la dirección más alta del banco 1 de registros, ya que las instrucciones CALL, PUSH y POP escriben en las posiciones apuntadas por este puntero. Se trata de evitar, en consecuencia, que las instrucciones mencionadas escriban accidentalmente sobre el banco 1. El valor del SP se puede modificar fácilmente con la siguiente instrucción: MOV SP,#0FH.

De la misma forma que la expuesta, se pueden realizar subrutinas de multiplicación de 24x16 bits, 24x24 bits, 32x16 bits y 32x32 bits, dependiendo de las necesidades de la aplicación que se quiera realizar.

La división de datos numéricos sin signo de 16 bits de longitud se puede efectuar mediante un proceso iterativo en el cual de debe multiplicar el dividendo por 2^{-i}, donde i esta comprendido entre 0 y 15, y se debe restar el resultado del divisor. La primera multiplicación se realiza para i=15, por lo que desplazan XH y XL 16 bits hacia la derecha, inicialmente XH está en R0 y XL en R1; luego se copia XH en R2 y XL en R3, y se pone R0 y R1 a cero, con lo que el dividendo pasa a estar formado por los registros R0, R1, R2 y R3, 32 bits en total, o sea, es una multiplicación por 2^{-16}, y se rota el dividendo obtenido una posición hacia la izquierda, consiguiendo que la multiplicación sea por 2^{-15} . Posteriormente, se restan los 16 bits altos del dividendo, R0 y R1, con el divisor, es decir:



$$(\text{dividendo} \times 2^{-i})_{16\ \text{bits altos}} - \text{divisor}$$

Esta operación se efectúa para comprobar si los 16 bits altos del dividendo son mayores, menores o iguales que los 16 bits del divisor, hecho que se refleja en el valor del bit de acarreo, pues en la resta se pueden dar dos casos:

1. El bit C se pone a 1, lo que implica
$$(\text{dividendo} \times 2^{-i})_{16\,bits\,altos} < \text{divisor}$$
2. El bit C de pone a 0, lo que implica:
$$(\text{dividendo} \times 2^{-i})_{16\,bits\,altos} \geq \text{divisor}$$

En el primer caso se complementa el bit de acarreo y se introduce en el cociente, CH y CL, mediante una rotación con acarreo hacia la izquierda. En el segundo caso se sustituye XH y XL por el resultado de la resta, en ZH y ZL, se complementa el bit de acarreo y se introduce éste en los registros del cociente.

Este proceso se itera 16 veces hasta que i = 0, entonces el resto de la división, que se halla situado en los mismos registros XH y XL, se coloca en los registros R2 y R3, respectivamente, y el cociente de la división (registros CH y CL), se coloca en los registros R0 y R1, respectivamente.

```
;*****************************************************
; Subrutina DIV16 División de 16 bits (XI-I-XL)/(YH-YL)
; Entrada: Dividendo: XH=R0 y XL=R1. Divisor: YH=R4 y YL=R5.
; Salida:   Cociente.: CH=R0 y CL=R1. Resto: ZH=R2 y ZL=R3
; Modifica: A, R0, R1, R2, R3, R7. Direcc. 08H, 09H, 0AH y 0BH.
;*****************************************************
;Variables temporales:
XL    EQU 0lH           ;XL en R1 (dir. directo)
XH    EQU 00H           ;XH en RO
YL    EQU 05H           ;YL en R5
YH    EQU 04H           ;YH en R4
CL    EQU 09H
CH    EQU 08H
ZL    EQU 0BH
ZH    EQU 0AH

DIV16:      MOV   A,YL        ;Lee dividendo
            ORL   A,YH        ;Comprueba si éste es cero
            JNZ   DIV_ON      ;Si es cero retorna
            SETB  OV          ;Pone antes bit OV a 1
            RET
DIV_ON:     MOV   R3,XL       ;-(XH-XL) x 2^-16
            MOV   R2,XH
            MOV   XL,#0       ;XL=XH=00H
            MOV   XH,#0
            MOV   CL,#0       ;Cociente
            MOV   CH,#0       ;CH=CL=00H
            MOV   ZL,#0       ;Resto
            MOV   ZH,#0       ;ZH=ZL=00H
            MOV   R7,#16      ;Itera 16 veces
            BUC_DIV:    CLR   C
            LCALL ROTA        ;Rota 1 pos izq. 2^-i
            LCALL SUBl6       ;(dividendo·2^-i)16 bits - divisor
            JC    XmenorqY    ;si (dividendo·2^-i)16 bits < divisor
            MOV   XL,ZL       ;en caso contrario
            MOV   XH,ZH       ;XH=ZH, XL=ZL
XmenorqY:   LCALL ROTAC
            DJNZ  R7,BUC_DIV  ;Hasta 16 veces
            MOV   R3,XL       ;División finalizada
            MOV   R2,XH       ;Pone resto en R2 y R3
            MOV   R0,CH       ;Pone cociente en R0 y R1
            MOV   Rl,CL
            CLR   OV          ;Pone a cero bit de overflow
            RET

;*****************************************************
; ROTAC
;*****************************************************
ROTAC:      CPL   C           ;Complementa acarreo
            MOV   A,CL        ;Rota cociente 1 bit
            RLC   A           ;hacia la izquierda
            MOV   CL,A
            MOV   A,CH
            RLC   A
            MOV   CH,A
            RET
```

```
;*************************************************
;SUB16      -Resta de 16 bits (XH-XL)-(YH-YL)
;Entrada: XH=R0, XL=R1, YH=R2, YL=R3.
;Salida: ZH en 0AH y ZL en 0BH
;*************************************************
SUB16:      CLR   C           ;Subrutina de resta
            MOV   A,XL        ;(XH-XL) - (YH-YL)
            SUBB  A,YL        ;Resultado en ZH y ZL
            MOV   ZL,A
            MOV   A,XH
            SUBB  A,YH
            MOV   ZH,A
            RET
;*************************************************
; ROTA
;*************************************************
ROTA:       MOV   A,R3        ;Subrutina de rotación del
            RLC   A           ;dividendo 1 bit hacia la derecha
            MOV   R3,A
            MOV   A,R2
            RLC   A
            MOV   R2,A
            MOV   A,XL
            RLC   A
            MOV   XL,A
            MOV   A,XH
            RLC   A
            MOV   XH,A
            RET
```

8.- Suma y resta de datos con signo.

La suma y resta de números con signo se realiza de forma similar a la suma y resta de números sin signo. La única diferencia estriba en que el **bit de overflow, OV**, se pone a 1 cuando el resultado de las operaciones está fuera de rango. En la suma y resta de datos de 16 bits el bit OV se pone a 1 si la operación con los bytes altos de los operandos el resultado está fuera de rango. En consecuencia, las rutinas realizadas de suma y resta de números sin signo se pueden emplear para el caso de números con signo, con la salvedad que tras la operaci6n hecha se ha de comprobar el estado del bit de overflow, por si el resultado está fuera de rango.

9.- Multiplicación y división de 16 bits con signo.

La multiplicación y división de datos con signo resulta más fácil de llevar a cabo si los datos están en un formato de valor absoluto y signo, de manera que la operaci6n se pueda realizar utilizando los algoritmos del apartado 7, obteniendo el signo del resultado a partir del signo de los operandos.

Si el formato de los datos está en complemento a dos, éstos se deben pasar al formato módulo-signo, se debe realizar la multiplicación o división de los operandos y pasar el resultado a complemento a dos. Al considerar el bit de signo de los operandos se observa que el resultado binario de la multiplicación coincide con la función booleana XOR (figura 5.8). La función XOR a nivel de bit no existe en la familia MCS-51; no obstante, esta función se puede realizar sencillamente teniendo en cuenta que si a las entradas de la funci6n XOR se las denomina a y b, y z a su salida, z es igual a b, z = b, si a es cero y z es igual a b negado, z = $\overline{b}$, si a vale 1 1ógico.

Multiplicación de signos	Multiplicación binaria de signos
+ · + = +	0 · 0 = 0
+ · - = -	0 · 1 = 1
- · + = -	1 · 0 = 1
- · - = +	1 · 1 = 0

A continuación se muestra la subrutina MUL16S que realiza la multiplicación de dos números de 16 bits con signo. Esta subrutina utiliza la subrutina anterior MUL16 para multiplicar en valor absoluto los dos operandos, por lo que MUL16S tendrá los mismos registros de entrada y de salida que MUL16. Para determinar el bit de signo resultante de la multiplicación se utilizan los **bits de propósito general FO y UD** del registro de estado PSW, es decir, los bits PSW.5 y PSW.1, que son definibles por el usuario.

```
;*************************************************
; Subrutina MUL16S
; Multiplica 16bitsxl6bits con signo, (YH-YL)x(XH-XL)
; Entrada:  R0=XH, R1=XL, R2=YH, R3=YL.
; Salida:   32 bits. R7=ZS0, R6=ZS1, R5=ZS2, R4=ZS3.
; Modifica: A, B, Registros R7 a R1 del banco 1.
;*************************************************
MUL16S:     MOV   A,RO         ;Lee XH
            RLC   A            ;Rota
            MOV   PSW.5,C      ;Guarda bit de signo de X
            MOV   A,R2         ;Lee YH
            RLC   A            ;Rota
            MOV   PSW.l,C      ;Guarda bit de signo de Y
            JNB   PSW.5,MOD_Y;IX es negativo?
            CLR   C            ;Borra acarreo
            CLR   A            ;Borra A
            SUBB  A,Rl         ;Complemento a 2 XL
            MOV   R1,A         ;Guarda XL
            CLR   A            ;Borra A
            SUBB  A,R0         ;Complemento a 2 XH
            MOV   R0,A         ;Guarda XH
            CLR   C            ;Borra acarreo
            CLR   A            ;Borra A
            SUBB  A,R3         ;Compl. a 2 de YL
            MOV   R3,A         ;Guarda YL
            CLR   A            ;Borra A
            SUBB  A,R2         ;Compl. a 2 de YH
            MOV   R2,A         ;Guarda YH
MOD_Y:      JNB   PSW.1,M_ABS ;¿Y es negativo?
            CLR   C            ;Borra acarreo
            CLR   A            ;Borra A
            SUBB  A,R3         ;Compl. a 2 de YL
            MOV   R3,A         ;Guarda YL
            CLR   A            ;Borra A
            SUBB  A,R2         ;Compl. a 2 de YH
            MOV   R2,A         ;Guarda YH
M_ABS:      LCALL MUL16        ;Multiplica en valor absoluto
            MOV   C,PSW.5      ;Función XOR de los signos
            JNB   PSW.l,ES_0
            CPL   C
ES_0:       JC    CPU          ;Resultado negativo
            RET
CPL4:       MOV   08H,#07      ;Carga direcc. ZSO
            SETB  RS0          ;Cambia a banco 1
            MOV   R7,#4        ;N° de bytes de ZS
            LCALL COMPL2_N     ;Compl. a 2 de ZS
            CLR   RS0          ;Cambia a banco 0
            RET
```

La subrutina MUL16S hace uso además de la subrutina COMPL2_N que realiza el complemento a dos del resultado ZS de 4 bytes, generado por la multiplicación. Esta subrutina es genérica y utiliza la dirección del operando (puntero @R0) y el tamaño del operando (R7) como datos de entrada.

```
;*****************************************************
; Subrutina COMPL2_N
; Realiza el complemento a 2 de un número de N bytes
; Entrada:  R0 (Puntero a X), R7 (Tamaño)
; Salida:   Resultado sustituye al operando X
; Modifica: C, A, R0, R7
;*****************************************************
COMPL2_N:   CLR   C
BUC_CP:     CLR   A
            SUBB  A,@R0
            MOV   @R0,A
            DEC   R0
            DJNZ  R7,BUC_CP
            RET
```

La subrutina siguiente DIV16S realiza división con signo de dos números de 16 bits, utilizando para ello la subrutina DIV16; por tanto, tiene los mismos parámetros de entrada y de salida que esta subrutina. En cuanto al resultado, en el caso que el dividendo sea positivo y el divisor sea negativo, y viceversa, el cociente deberá ser negativo; si ambos, dividendo y divisor, son positivos o negativos, entonces el cociente debe ser positivo. En cualquier caso, el resto se deja siempre positivo.

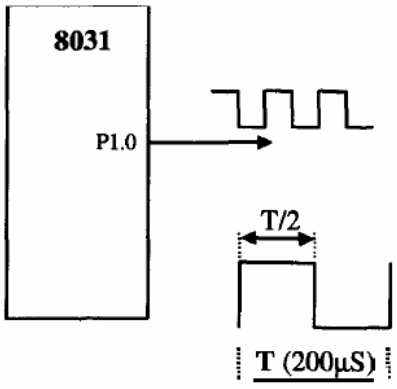
```
;*****************************************************
; Subrutina DIV16S
; Entrada:  Dividendo: XH=R0 y XL=R1. Divisor: YH=R4 y YL=R5
; Salida:   Cociente: CH=R0 y CL=R1. Resto: ZH=R2 y ZL=R3
; Modifica: A, R0, R1, R2, R3, R7.
;            Direcc. 08H, 09H, 0AH, 0BH, 20H y 21H
;*****************************************************
DIV16S:     MOV   20H,R0      ;Pone XH en 20H
            MOV   21H,R4      ;Pone YH en 21H
            JNB   20H.7,MOD_Y ;¿X es negativo?
            SETB  RS0         ;Cambio a banco 1
            MOV   R0,#0lH     ;Carga dir. XL en puntero R0
            MOV   R7,#2       ;W de bytes
            LCALL COMPL2_N    ;Complemento a 2
            JNB   21H.7,M_ABS ;¿Y es negativo?
            MOV   R0,#03H     ;Carga dir. YL en puntero R0
            MOV   R7,#2       ;W de bytes
            CLR   RS0         ;Cambio a banco 0
M_ABS:      LCALL DIV16       ;División en valor absoluto
            JB    OV,SALlR
            JB    20H.7,DIVDN ;Si dato X es negativo
            JB    21H.7,CPLDIV      ;Si dato Y es negativo
SALIR:      RET                     ;Dato X y dato Y son positivos
DIVDN:      JNB   21H.7,CPLDIV      ;Si dato Y es positivo
            RET                     ;Dato X y dato Y son negativos
CPLDIV:     SETB  RS0               ;Cambio a banco 1
            MOV   R0,#0lH           ;Carga dir. cociente
            MOV   R7,#2             ;N° de bytes
            CALL  COMPL2_N          ;Complementa a 2
            CLR   RS0               ;Cambio a banco 0
            RET
```

10.- Ejemplos de aplicación.

A continuación se describen una serie de ejemplos resueltos donde se aplican parte de las rutinas expuestas en este capitulo. Los ejemplos descritos se pueden realizar, de forma más efectiva y con una mejor explotación de los recursos del microcontrolador, utilizando interrupciones, por lo que para algunos de los ejemplos se propondrá su solución alternativa en capítulos posteriores, teniendo ya un mejor conocimiento del proceso de interrupciones y de los recursos internos de los microcontroladores de la familia MCS51.

El programa tendrá una rutina inicial, rutina "Inicio", en la cuál se configuran los recursos internos del microcontrolador y el modo de operar de éste. De esta rutina se pasa a una rutina "Principal", que se encarga de llevar a cabo las funciones requeridas por la aplicación. La rutina principal será una rutina que forme un bucle infinito, pues las aplicaciones a resolver necesitan que el microcontrolador esté constantemente pendiente de las tareas que debe efectuar. La rutina "Inicio" en algunos ejemplos no aparecerá, porque los programas son sencillos, y no será necesario poner un valor inicial a las variables empleadas.

10.1 Generación de una señal cuadrada.



En un sistema es frecuente tener que generar una señal periódica de frecuencia determinada y lo más simétrica posible. En este ejemplo, se propone la creación de una señal cuadrada de 5kHz de frecuencia, que se extraerá por la patilla P1.0 de un microcontrolador 8031.

Para generar una frecuencia de 5kHz se precisa de un periodo de 200 us y, por tanto, cambiar el estado de la patilla P1.0 cada bucle. Para cambiar el estado de P1.0 se utilizará la instrucción CPL de complemento de bit, y para generar un retardo de 100 us se utilizará una rutina de retardo como la descrita en el apartado 4.

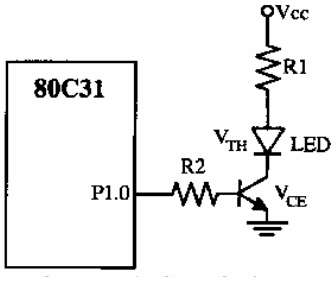
```
;*********************************************
; Programa de generación de una señal cuadrada
;*********************************************
ORG   0H
            LJMP  Principal
; Rutina Principal
ORG   0l00H

Principal:  CPL   P1.0        ;Complementa estado lógico de P1.0
            CALL  Retardo     ;Llama a la rutina de retardo
            SJMP  Principal   ;Bucle infinito a principal

Retardo:    MOV   R7,#46      ;Pone 46 en R7
L1:         DJNZ  R7,L1       ;Bucle sobre L1
            RET               ;Retorno de subrutina
```

Para que la instrucci6n CPL se ejecute cada 100 us, se debe determinar de forma precisa el valor de carga del registro R7 de la subrutina de retardo. Para ello, si se considera que la frecuencia de reloj del microcontrolador es de 12MHz y que, entonces, un ciclo máquina tarda en efectuarse 1 useg, y que la instrucción CPL se ejecuta en 1 ciclo máquina, indicado como CM, la instrucción CALL se ejecuta en 2 CM, SJMP en 2 CM, "MOV R7, #46" en l CM, DJNZ en 2 CM y RET en 2 CM. El tiempo de ejecución de la rutina de retardo viene dado por:

$$T_{Ret} = 1 + 2 \cdot R7 + 2CM$$

Y el tiempo total en que se ejecuta la instrucción CPL a cada vuelta del bucle principal es:

$$t_{CPL} = 1 + 2 + 2 + t_{Ret} = 100CM \text{ o } 100\mu s$$



Se deduce que t_{Ret} debe valer 95 us, y que el registro R7 debe valer 46. El programa realizado se puede modificar para efectuar el parpadeo de un diodo led conectado por medio de un transistor al microcontrolador. La resistencia R1 debe calcularse para limitar la corriente del diodo led a unos 20 mA, valor más que suficiente para mantener al diodo encendido de manera adecuada.

La resistencia R2 se debe determinar para que el transistor entre en saturación cuando la salida de P1.0 esté a 1 1ógico. La rutina modificada se muestra a continuación:

```
;***************************************
; Programa de parpadeo de un diodo led
;***************************************
ORG   0H
            LJMP  Principal
;***************************************
; Rutina Principal
;***************************************
ORG 0l00H
Principal:  CPL   P1.0
            MOV   R6,#0
            MOV   R7,#0
Retardo:    DJNZ  R6,Retardo
            DJNZ  R7,Retardo
            SJMP  Principal
```

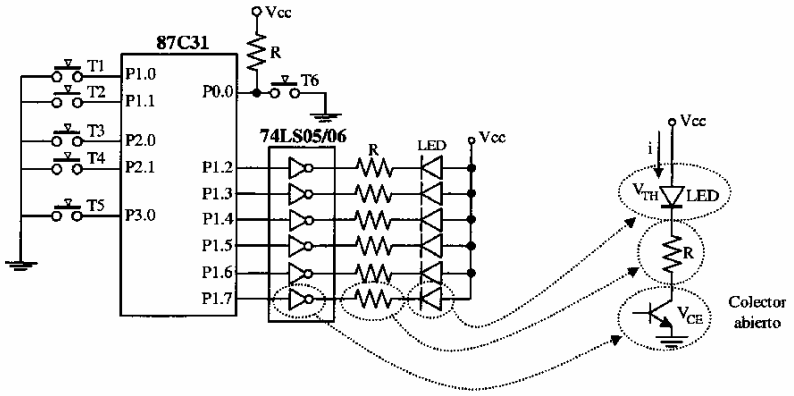
El tiempo que tarda en ejecutarse la rutina de retardo, bucle formado por L1, deber ser suficiente para que el parpadeo del led sea perceptible. Este tiempo es:

$$t_{L1} = [2 \cdot (255+1) + 2] \cdot (255+1) = 131584\mu s \approx 0{,}13s$$

La frecuencia de la señal cuadrada será en este caso de 3,8Hz, aproximadamente.

10.2 Conexión de teclas al microcontrolador.

Las teclas o pulsadores son elementos que aparecen en la mayor parte de los sistemas basados en un microcontrolador, por lo que en este ejemplo se propone la conexión de teclas a cada uno de los puertos del microcontrolador, tal y como indica la figura siguiente.



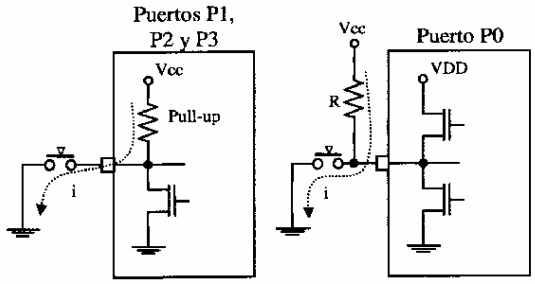
En la figura se emplea el circuito integrado 74LS05, o el 74LS06, que contiene hasta 6 puertas inversoras en colector abierto. El terminal de salida de la puerta inversora es el colector de un transistor interno, cuyo emisor está conectado a masa, tal y como se ve en el detalle de esta figura. La puerta inversora puede soportar una comente máxima de 25mA, para el 74LS05, y de 30mA a 40mA, para el 74LS06, valores más que suficientes para encender de manera adecuada el diodo led que tiene conectado.

La resistencia R se debe calcular de forma adecuada para que, considerando el valor de Vcc, de la tensión umbral del diodo led, V_m, y de la tensión colector-emisor del transistor en saturación, VCE, la comente I está limitada a unos 20mA, valor más que suficiente para iluminar adecuadamente el diodo led. El valor de la resistencia R se puede determinar a partir de la siguiente ecuación:

$$R = \frac{V_{CC} - V_{TH} - V_{CE}}{i}$$

Los puertos P1, P2 y P3 tienen una resistencia interna de pull-up conectada a cada una de las salidas, por lo que una tecla se puede conectar directamente a la patilla de uno de estos puertos. Debido a la resistencia de pull-up, el estado de una tecla no pulsada será de 1 1ógico y el estado de una tecla pulsada será de 0 lógico (conexión directa a masa), por lo que basta con leer el estado lógico del terminal del puerto para saber si la tecla está siendo pulsada o no.

El puerto P0 no tiene resistencia interna de pull-up y en su lugar hay un transistor NMOS, debido a que este puerto debe tener la característica de triestado. Por ello, para conectar una tecla se debe poner una resistencia externa de pull-up, que proporcione un estado 1 lógico, cuando no se pulsa la tecla, y un estado 0 lógico, cuando se pulsa la tecla.



El programa deberá leer el estado de las teclas y encender un diodo led asociado a la tecla, en el caso de que se pulse. El diodo led se mantendrá encendido mientras la tecla permanezca pulsada. Las instrucciones CLR que se utilizan para borrar los leds pueden sustituirse por una única instrucción que escriba en todo el puerto P1. Para ello, se debe tener en cuenta que hay dos teclas conectadas a P1.0 y P1.1, respectivamente; deben ponerse estos terminales a 1 lógico, de forma que sea posible leer si la tecla ha sido pulsada o no.

```
;*************************************
; Programa para la lectura de la teclas
;*************************************
ORG   0H
      MOV   P1,#00000011b     ;Leds apagados. P1.0 P1.l entradas
      LJMP  Principal
;*************************************
; Rutina Principal (Testeo de teclas,encendido/apagado de leds)
;*************************************
ORG   0100H
Principal:  JNB   P1.0,Tecla_T1     ;Comprueba si se ha pulsado T1
            JNB   P1.l,Tec1a_T2     ;Comprueba si se ha pulsado T2
            JNB   P2.0,Tecla_T3     ;Comprueba si se ha pulsado T3
            JNB   P2.1,Tecla_T4     ;Comprueba si se ha pulsado T4
            JNB   P3.0,Tecla_T5     ;Comprueba si se ha pulsado T5
            JNB   P0.0,Tecla_T6     ;Comprueba si se ha pulsado T6
            CLR   P1.2              ;Apaga led conectado a P1.2
            CLR   P1.3              ;Apaga led conectado a P1.3
            CLR   P1.4              ;Apaga led conectado a P1.4
            CLR   P1.5              ;Apaga led conectado a P1.5
            CLR   P1.6              ;Apaga led conectado a P1.6
            CLR   P1.7              ;Apaga led conectado a P1.7
            SJMP Principal
Tecla-T1:   MOV   P1, #0000 0lllb   ;Enciende P1.2 y apaga resto
            SJMP Principal          ;Regresa a Principal
Tecla-T2:   MOV   P1, #0000 l0llb   ;Enciende P1.3 y apaga resto
            SJMP Principal          ;Regresa a Principal
Tecla-T3:   MOV   P1, #0001 00llb   ;Enciende P1.4 y apaga resto
            SJMP principal          ;Regresa a Principal
Tecla-T4:   MOV   P1, #0010 0011b   ;Enciende P1.5 y apaga resto
            SJMP Principal          ;Regresa a Principal
Tecla-T5:   MOV   P1, #0l00 0011b   ;Enciende P1.6 y apaga resto
            SJMP Principal          ;Regresa a Principal
Tecla-T6:   MOV   P1, #l000 0011b   ;Enciende P1.7 y apaga resto
            SJMP Principal          ;Regresa a Principal
```

Se puede escribir sobre todo el puerto P1, sustituyendo a las instrucciones CLR Pl.X, con la siguiente instrucción MOV:

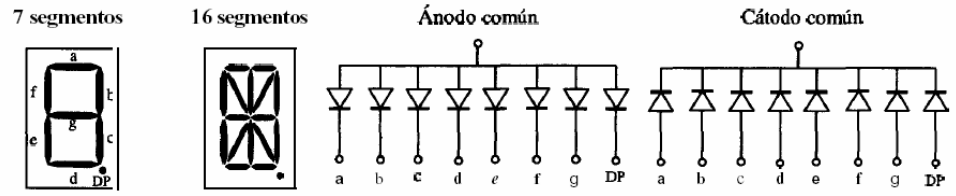
```
       MOV   P1,#0000 0011b          ;Bits Pl.0 y Pl.l a 1 lógico
```

La rutina principal lee el estado de cada tecla y apaga los leds conectados al puerto P1. Si se pulsa una tecla, se enciende el diodo led asociado y se salta a la etiqueta Principal, de forma que el led se mantiene encendido mientras se pulse la tecla. Al dejar de pulsar la tecla el programa apaga todos los leds.

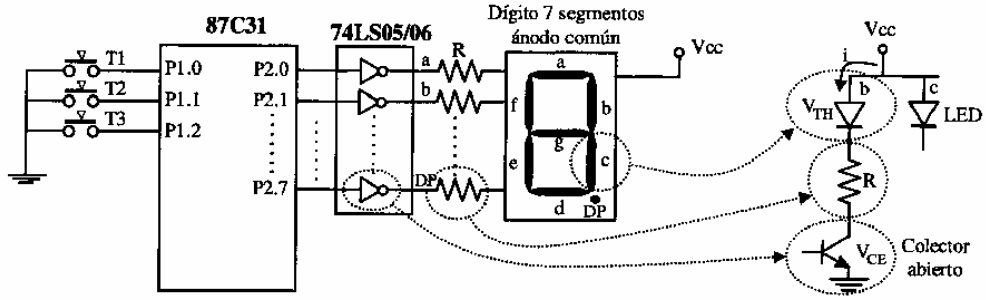
10.3.- Conexión de un dígito de 7 segmentos.

Los dígitos de visualización se usan de manera frecuente para indicar al usuario una determinada información. Los dígitos suelen estar formados por 7 segmentos o por 16 segmentos (hexadecimales), aunque existen en el mercado una gran variedad y gama de dígitos

a disposición del diseñador. Los dígitos de 7 segmentos son adecuados para visualizar números y algunas letras y símbolos, estos últimos con poca definición (figura 5.13). Los dígitos hexadecimales están formados por 16 diodos led, por lo que tienen una mejor definición para números, letras y símbolos.



En este ejemplo se conectan tres teclas y un digito de 7 segmentos al microcontrolador. El programa deberá leer el estado de las teclas y poner en el digito la letra "A" si se pulsa la tecla T1, la letra "b" si se pulsa la tecla T2, y la letra "C" si se pulsa la tecla T3.



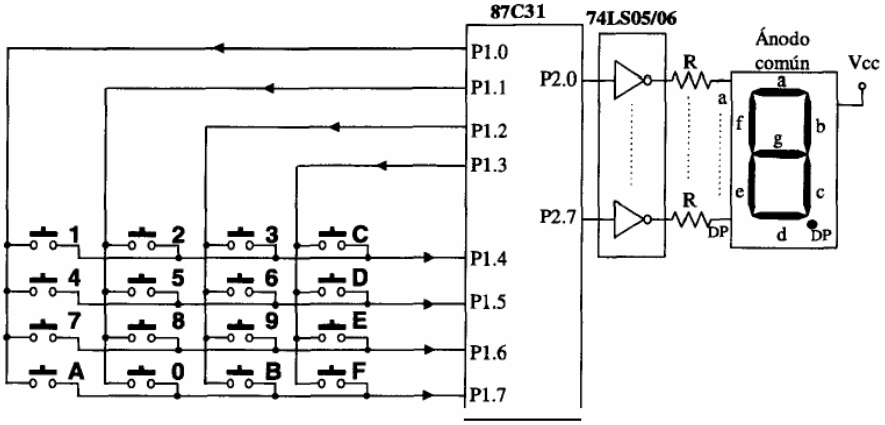
En la conexión se utiliza, del mismo modo que el ejemplo anterior, un par de circuitos integrados 74LS05 ó 74LS06 que contienen hasta 6 puertas inversoras en colector abierto. El digito utilizado es de ánodo común, el nodo está conectado a la tensión de alimentación, Vcc, y cada diodo a una resistencia y a un colector del circuito integrado. La resistencia se debe calcular para que limite la corriente a un valor mínimo de 20mA. El programa de este ejemplo se muestra a continuación:

```
;*******************************************************
; Programa para la conexión de un digito de 7-segmentos
;*******************************************************
            ORG   0H
            MOV   P2, #0            ;Apaga todos los leds del dígito
            LJMP  Principal
;*******************************************************
; Rutina principal
;*******************************************************

            ORG   0l00H
Principal:
            JNB   P1.0, Tecla-T1    ;Comprueba si se ha pulsado T1
            JNB   P1.1, Tecla-T2    ;Comprueba si se ha pulsado T2
            JNB   P2.0, Tecla-T3    ;Comprueba si se ha pulsado T3
            MOV   P2, #0            ;Borra el digito
            SJMP  Principal
Tecla-T1:   MOV   P2, #0111 0111b   ;Pone letra A en el digito
            SJMP  Principal         ;Regresa a Principal
Tecla-T2:   MOV   P2, #0111 1100b   ;Pone letra B en el digito
            SJMP  Principal         ;Regresa a Principal
Tecla-T3:   MOV   P2, #0111 l00lb   ;Pone letra C en el digito
            SJMP  Principal         ;Regresa a Principal
```

10.4 Conexión de un teclado matricial de 4 x 4 teclas.

Los teclados son un elemento imprescindible en un sistema debido a que posibilitan la relación entre éste y el usuario al permitir la introducción de datos y la selección de diferentes modos de operación del programa, cuando la aplicación es compleja.

La figura siguiente muestra la conexión de un teclado matricial de 4x4 teclas al puerto P1 de un 87C31. Existe una enorme variedad y diferentes tipos de teclados en el mercado; no obstante la disposición de las teclas en éstos suele ser matricial, disposición del tipo fila/columna, de manera que una serie de teclas comparten una misma columna y otra serie de teclas comparte la misma fila.



La conexión del teclado con el microcontrolador resulta sencilla, pues utiliza los cuatro bits bajos de P1 como salidas conectadas a las columnas del teclado, y los cuatro bits altos como entradas conectadas a las filas del teclado. En este ejemplo se conecta un digito de 7 segmentos para que se muestre un carácter determinado, correspondiente a la tecla pulsada. El programa deberá efectuar la lectura del teclado matricial de manera continua. Cuando se pulse una tecla, se mostrará el carácter que tiene asociado la tecla en el digito de 7 segmentos. En el digito siempre se mostrará el código de la última tecla pulsada.

La detección de una tecla pulsada en el teclado matricial se realizará columna a columna, de forma secuencial, es decir, se pondrá un dato en las columnas del teclado (patillas P1.0, P1.l, P1.2 y P1.3), que habilite sólo una de las columnas, y se leerá a continuación el estado lógico de las filas del teclado patillas P1.4, P1.5, P1.6 y P1.7), de manera que si se ha pulsado una tecla se detectará mediante el dato leído en la filas. Tras haberse comprobado la columna, ésta se inhabilitará y se procederá a habilitar la siguiente columna; este proceso se repetirá hasta que se hayan testeado todas las columnas del teclado. Este procedimiento se debe repetir de forma continuada, comprobando cada una de las columnas del teclado matricial.

Tal y como se ha conectado el teclado matricial al microcontrolador, la habilitación de una columna se hace poniendo un 0 1ógico en una de las patillas de salida de P1, o sea, P1.0, P1.l, P1.2 6 P1.3, y se habilita poniendo un 1 lógico. Luego, una columna estará habilitada, mientras el resto no. Si se pulsa una tecla de la columna habilitada, el estado lógico de la columna, 0 lógico, pasará a una de las filas, dependiendo de cuál haya sido la tecla pulsada. En cambio, si no se pulsa ninguna tecla, el nivel lógico que se lee es un 1 lógico, puesto que los terminales P1.4, P1.5, P1.6 y P1.7 están configurados como entradas, es decir, su transistor NMOS está en corte y el estado 1ógico que se lee corresponde al que proporciona la resistencia interna de pull-up. Sin embargo, en esta situación, si se pulsa una tecla correspondiente de una columna inhabilitada, en la fila correspondiente se lee un 1 lógico, debido a que el estado de la columna es 1 1ógico por estar habilitada.

En definitiva, para habilitar la primera columna (teclas 1, 4, 7 y A) e inhabilitar el resto de columnas, se debe escribir en el puerto el dato 1111 1110b y configurar, además, las patillas conectadas a las filas como entradas (estado 1 lógico, transistor NMOS en corte). Para inhabilitar la segunda columna, filas 2, 5, 8 y 0) e inhabilitar el resto, se debe poner 1111 1101b en P1. Para habilitar la tercera columna (teclas 3, 6, 9 y B) e inhabilitar el resto, se debe poner 1111 1011b en P1. Y, por último, para habilitar la cuarta columna (teclas C, D, E y F) e inhabilitar el resto, se debe poner 1111 0111b en P1.

El programa deberá llevar a cabo la secuencia descrita de testeo por medio de un bucle infinito. Además, también se utilizará una tabla para hacer la conversión a 7 segmentos del carácter que se quiere visualizar.

```
;************************************************
; Programa de lectura de teclado matricial
;************************************************

            ORG   0H

            MOV   P2,#0             ;Apaga todos los leds del digito
            MOV   B,#0              ;Borra registro B
            LJMP  Principal
;************************************************
; Rutina Principal
;************************************************
ORG   0100H

Principal:        MOV   P1, #1111 lll0b   ;Habilita primera col.
                  CALL  Det_tecla         ;Sub. Detección tecla
                  JNZ   A,Colum_1         ;A≠0 si se pulsa tecla
                  MOV   P1, #1111 ll0lb   ;Habilita segunda col.
                  CALL  Det_tecla
                  JNZ   A,Colum_2         ;A≠0 si se pulsa tecla
                  MOV   P1,#llll l0llb    ;Habilita tercera col.
                  CALL  Det_tecla
                  JNZ   A, Colum_3        ;A≠0 si se pulsa tecla
                  MOV   P1, #1111 0lllb   ;Habilita cuarta col.
                  CALL  Det_tecla
                  JNZ   A, Colum_4        ;A≠0 si se pulsa tecla
Prin2:            CALL  Display
                  MOV   B,#0              ;Borra registro B
                  SJMP Principal          ;Bucle infinito
Column_1:         JNB   P1.4,Tecla_1      ;¿Se ha pulsado tecla 1?
                  JNB   P1.5, Tecla_4     ;¿Se ha pulsado tecla 4?
                  JNB   P1.6, Tala_7      ;¿Se ha pulsado tecla 7?
                  SJMP  Tecla_A           ;si no es ninguna es la A
Column_2:         JNB   P1.4,Tecla_2      ;¿Se ha pulsado tecla 2?
                  JNB   P1.5,Tecla_5      ;¿Se ha pulsado tecla 5?
                  JNB   P1.6,Tecla_8      ;¿Se ha pulsado tecla 8?
                  SJMP  Tecla_O           ;si no es ninguna es la O
Column_3:         JNB   P1.4,Tecla_3      ;¿Se ha pulsado tecla 3?
                  JNB   P1.5, Tecla_6     ;¿Se ha pulsado tecla 6?
                  JNB   P1.6, Tecla_9     ;¿Se ha pulsado tecla 9?
                  SJMP  Tecla_B           ;si no es ninguna es la B
Column_4:         JNB   P1.4,Tecla_C      ;¿Se ha pulsado tecla C?
                  JNB   P1.5,Tecla_D      ;¿Se ha pulsado tecla D?
                  JNB   P1.6,Tecla_E      ;¿Se ha pulsado tecla E?
                  SJMP  Tecla_F           ;si no es ninguna es la F

Tecla_0:          MOV   B,#1              ;Pone en B código tabla
                  LJMP  Prin2
```

```
Tecla_1:        MOV   B,#2              ;Pone en B código tabla
                LJMP  Prin2
Tecla_2:        MOV   B,#3              ;Pone en B código tabla
                LJMP  Prin2
Tecla_3:        MOV   B,#4              ;Pone en B código tabla
                LJMP  Prin2
Tecla_4:        MOV   B,#5              ;Pone en B código tabla
                LJMP  Prin2
Tecla_5:        MOV   B,#6              ;Pone en B código tabla
                LJMP  Prin2
Tecla_6:        MOV   B,#7              ;Pone en B código tabla
                LJMP  Prin2
Tecla_7:        MOV   B,#8              ;Pone en B código tabla
                LJMP  Prin2
Tecla_8:        MOV   B,#9              ;Pone en B código tabla
                LJMP  Prin2
Tecla_9:        MOV   B,#10             ;Pone en B código tabla
                LJMP  Prin2
Tecla_A:        MOV   B,#11             ;Pone en B código tabla
                LJMP  Prin2
Tecla_B:        MOV   B,#12             ;Pone en B código tabla
                LJMP  Prin2
Tecla_C:        MOV   B,#13             ;Pone en B código tabla
                LJMP  Prin2
Tecla_C:        MOV   B,#13             ;Pone en B código tabla
                LJMP  Prin2
Tecla_C:        MOV   B,#13             ;Pone en B código tabla
                LJMP  Prin2
Tecla_C:        MOV   B,#13             ;Pone en B código tabla
                LJMP  Prin2
;************************************************
; Subrutina para la detección de la tecla pulsada
; Lee el puerto P1 y retorna A=0 si no se pulsa una tecla
; y A≠0 si se pulsa tecla
;************************************************
Det_tecla:      MOV   A,P1              ;Lee puerto P1(filas)
                ORL   A,#0FH            ;Fuerza 4 bits bajos 1
                CPL   A                 ;Complementa acumulador
                RET                     ;Si no pulsa A=0,A=1 cont.

;************************************************
; Subrutina de visualización de un carácter en 7 segmentos
;************************************************
Display:        MOV   A,B               ;Lee registro B
                CALL  Vis_7seg          ;Llama rutina de carácter
                MOV   P2,A              ;Pone en P2 codigo 7 seg
                RET
Vis_7seg:       INC   A
                MOVC  A,@A+PC           ;Lectura tabla de convers.
                RET
                DB    0000 0000b        ;(A=1)digito apagado
                DB    0011 1111b        ;(A=2) carácter 0
                DB    0000 0110b        ;(A=3) carácter 1
                DB    0101 1011b        ;(A=4) carácter 2
                DB    0100 1111b        ;(A=5) carácter 3
                DB    0110 0000b        ;(A=6) carácter 4
                DB    0110 0000b        ;(A=7) carácter 5
                DB    0111 0000b        ;(A=8) carácter 6
                DB    0000 0000b        ;(A=9) carácter 7
                DB    0111 0000b        ;(A=10) carácter 8
                DB    0110 0000b        ;(A=11) carácter 9
```

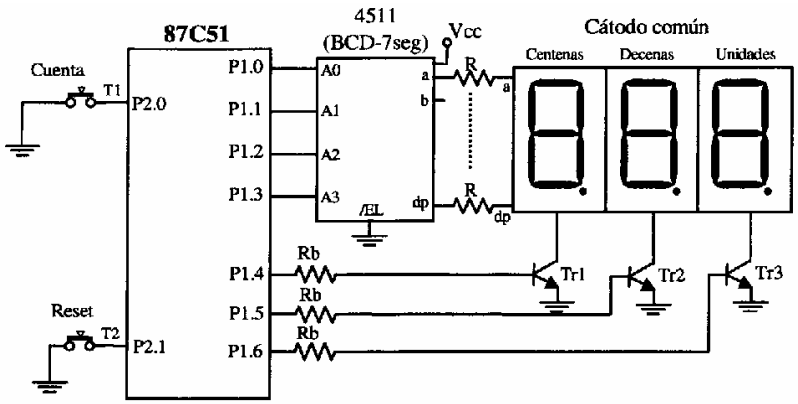
```
DB    0111 0000b        ;(A=12) carácter A
DB    0111 0000b        ;(A=13) carácter B
DB    0011 0000b        ;(A=14) carácter C
DB    0101 0000b        ;(A=15) carácter D
DB    0111 0000b        ;(A=16) carácter E
DB    0111 0000b        ;(A=17) carácter F
```

10.5.- Conexión de varios dígitos de 7 segmentos, aplicación de "Su turno".

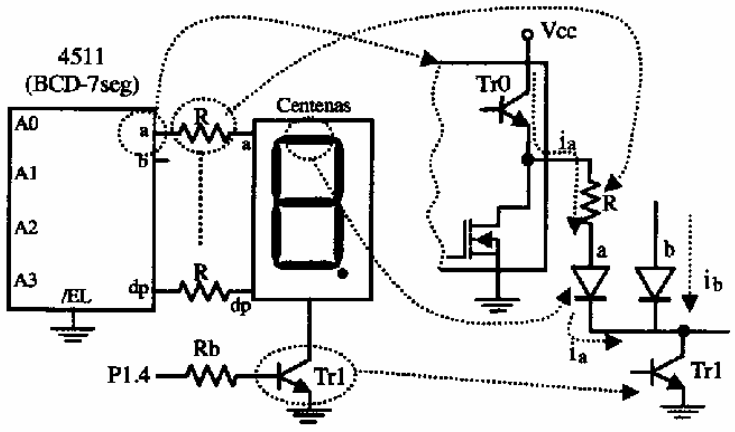
La figura muestra la forma de conectar varios dígitos de 7 segmentos a un 87C51 para una aplicación donde se desea controlar el turno de atención a un cliente, que es tan común en tiendas y locales comerciales. Debido al tipo de aplicación donde se ha de visualizar el número de orden del cliente, los dígitos se pueden controlar mediante el circuito integrado 4511, que es un conversor de formato BCD a 7 segmentos, de forma que colocando el número en BCD a la entrada del 4511, A0-A3, éste activa los leds necesarios (salidas a, b, c, d, e, f, g y dp) para que el número aparezca en el dígito.

El 4511 es capaz de suministrar hasta 25mA de corriente, valor más que suficiente para tener un diodo led encendido de forma adecuada. Este circuito integrado tiene un latch interno que le permite almacenar el valor del último número almacenado. El latch está gobernado por la entrada /EL, de manera que cuando /EL está a 0 lógico habilita el 4511 y activa los leds correspondientes al número de entrada (A0-A3), y cuando en /EL se produce un flanco positivo (paso de 0 a 1 lógico), el match almacena el Último dato presente en la entrada y mantiene encendidos los leds correspondientes.

El 4511 dispone de dos entradas de control adicionales, /ILT y /IB. La entrada /ILT enciende todos los leds del dígito conectado, para hacer así un chequeo del estado de los leds. La entrada /IB borra todos los leds del dígito conectado.



En esta aplicación se usará un microcontrolador 8751, el 4511, tres dígitos de 7 segmentos y tres transistores para controlar el encendido de cada uno de los dígitos. Por sencillez, las entradas /ILT y /IB del 4511 no se usarán, y la entrada /EL se conectará directamente a tierra, de forma que esté siempre habilitado.

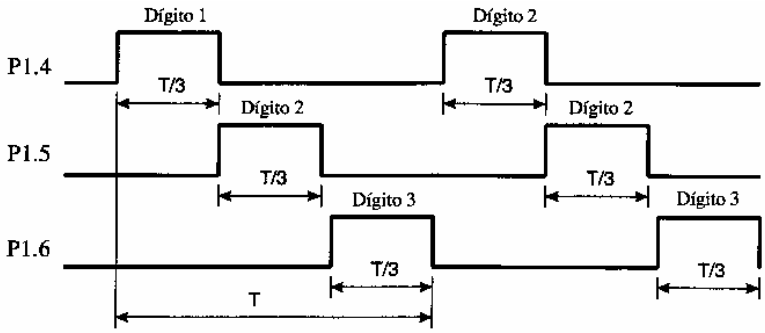


El circuito formado por el 4511, un dígito conectado a éste (centenas) y el transistor de control de encendido del dígito, Trl, se muestran en la figura siguiente. El 4511 tiene un transistor bipolar de salida, Tr0, conectado a la tensión de alimentación Vcc. El 4511 enciende un diodo led, por ejemplo el led "a" de la figura, cuando el transistor

bipolar Tr0 está en saturación, de forma que la comente circula a través de la resistencia R, del diodo led "a" y del transistor Trl. El diodo se encenderá siempre y cuando el transistor Trl esté en saturación.

Las salidas del 4511 están conectadas a los diodos de cada segmento, es decir, la salida que activa el diodo led "a" del dígito de las centenas, también está conectada al led "a" del dígito de las decenas y del dígito de las unidades. De esta forma, se simplifica el circuito y se reduce el número de terminales necesarios para los tres dígitos. En cuanto al consumo, debe considerarse que un dígito puede llegar a consumir hasta 160mA (8 diodos led y 20mA por diodo), por tanto, con tres dígitos se puede llegar a un consumo de 480mA, y si el número de dígitos es mayor el consumo de comente se dispara considerablemente.

Para evitar el consumo excesivo por parte de los dígitos y para utilizar el circuito realizado, los dígitos se deben encender de manera secuenciada, es decir, primero se enciende el dígito de las centenas, luego el dígito de las decenas y, por último, el dígito de las unidades. Esta secuencia se puede llevar a cabo por medio de la activación de los transistores conectados al cátodo común de cada dígito (transistores Trl, Tr2 y Tr3, respectivamente). Esta secuencia se debe repetir con una frecuencia de 25Hz, como mínimo, para que el parpadeo no sea perceptible por una persona y que la imagen se visualice de manera correcta.

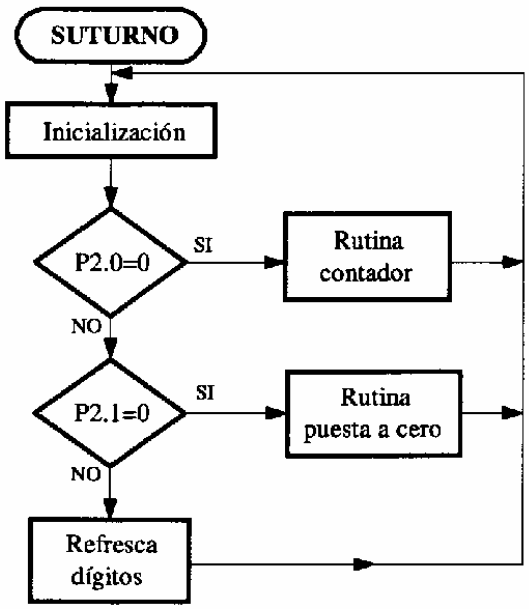


Los transistores TR1, Tr2 y Tr3 pueden ser cualquier tipo de transistor capaz de soportar como mínimo 200mA. Para el caso de un número elevado de dígitos se pueden emplear arrays de transistores darlintongs como los disponibles en los circuitos integrados de las series ULN2800A y ULN2980A, capaces de soportar comentes de 350mA y 500mA.

Las funciones básicas que deberá realizar el programa son:

1. Controlar hasta un máximo de 199 peticiones. Cuando la cuenta exceda esta cantidad se deberá poner a 000.
2. Se tienen 2 pulsadores, uno de cuenta y otro de reset (de puesta a cero). El pulsador de cuenta siempre incrementará en uno el turno actual. El pulsador de reset pondrá a O00 el valor de la cuenta.

La manera más sencilla de hacer este programa consiste en realizar una rutina de cuenta en formato BCD, como la rutina CONTA descrita en el apartado 3.6. En esta rutina intervienen los registros R0, R1 y R2. El registro R0 se emplea para las unidades, R1 para las decenas y R2 para las centenas. El flujograma del programa se muestra en la figura siguiente.

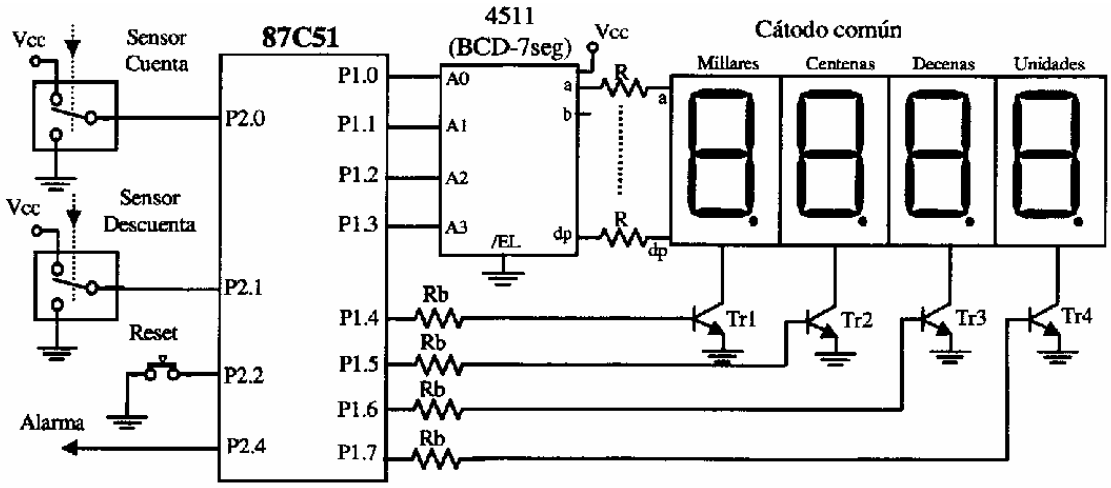


```
;***********************************************
; Programa para la gestión del turno de espera
;***********************************************

            ORG   0H
            MOV   P1, #0            ;Pone a cero la entrada del 4511
            MOV   R0, #0            ;Pone en corte Tr1, Tr2 y Tr3
            MOV   R1, #0
            MOV   R2, #0

;***********************************************
; Rutina Principal
;***********************************************
Principal:  JNB   P2.0, Cuenta
            JNB   P2.1,  PaCero
Prh2:       CALL  Display
            SJMP  Principal
Cuenta:     CALL  CONTA
            SJMP  Prin2
PaCero:     MOV   R0, #0
            MOV   R1, #0
            MOV   R2, #0
            SJMP Prin2
;***********************************************
; Subrutina CONTA
; Registros afectados: R0, R1, R2
;***********************************************
CONTA:      CJNE  R0, #9, UNIDAD          ;Compara unidad
            CJNE  R1, #9, DECENA          ;Compara decena
            CJNE  R2, #1, CENTENA         ;Compara centena
            MOV   R0, #0                  ;Al ser 199 pone a 000
            MOV   R1, #0
            MOV   R2, #0
            RET
UNIDAD:     INCR  0                 ;incrementa unidades
            RET
DECENA:     MOV   R0, #0            ;Pone a 0 las unidades
            INC   R1                ;incrementa decenas
            RET
CENTENA:    MOV   R0, #0            ;Pone a 0 unidades y decenas
            MOV   R1, #0
            INC   R2                ;incrementa centenas
            RET
;***********************************************
;Subrutina Display (Muestra en los dígitos el valor de la cuenta
;Registros afectados: RO, R1, R2
;***********************************************
Display:    MOV   P1, R2      ;Centenas P1 (4 bits altos R2 cero)
            SETB  P1.4        ;Enciende centenas. Trl conduce
            CALL  RETA        ;Retardo (tiempo encendido dígito)
            CLR   P1.4        ;Apaga dígito centenas. Trl en corte
            MOV   P1, R1      ;Decenas P1 (4 bits altos R1 cero)
            SETB  P1.5        ;Enciende decenas. Tr2 conduce
            CALL RETA         ;Retardo (tiempo encendido dígito)
            CLR   P1.5        ;Apaga dígito decenas. Tr2 en corte
            MOV   P1, R0      ;Unidades P1 (4 bits altos R0 cero)
            SETB  P1.6        ;Enciende unidades. Tr3 conduce
            CALL  RETA        ;Retardo (tiempo encendido dígito)
            CLR   P1.6        ;Apaga unidades. Tr3 en corte
            RET
```

Cada uno de los dígitos se enciende durante un tiempo de 6172 useg aproximadamente (suponiendo un reloj de 12MHz). Por tanto, el número se muestra cada 6172*3 useg, lo que supone una frecuencia de refresco del número visualizado de 54Hz, aproximadamente. Esta frecuencia puede ser superior, puesto que se tiene una limitación de frecuencia mínima, pero no en cuanto a frecuencia máxima. Luego, la frecuencia de refresco de todos los dígitos puede ser de 100Hz o de 1kHz, con la única limitación de la velocidad de los transistores Trl, Tr2 o Tr3 y del 4511.

10.6.- Contador de piezas.

Con una pequeña modificación del circuito y del programa anterior se puede realizar un contador de piezas para una máquina en un proceso industrial. En esta aplicación se dispone de dos sensores que se simbolizan por una caja con un conmutador. Un sensor indica cuándo se ha producido una pieza y el otro indica cuándo el proceso posterior de control de calidad descarta una pieza defectuosa. Por tanto, el contador muestra siempre el número neto de piezas fabricadas.



Al circuito anterior se le ha añadido un dígito más, por lo que la cuenta puede llegar hasta 9999 piezas. Si el número de piezas llega a su máximo valor y se rebasa éste, por un pulso más recibido, el contador debe mostrar el número máximo 9999 y activar la señal de alarma conectada a la patilla P2.4 del microcontrolador. Si el contador está a 0000 y recibe un pulso del sensor de descuenta, se activará también la señal de alarma conectada a P2.4, para indicar un mal funcionamiento del sensor o al error en el proceso de control de calidad.

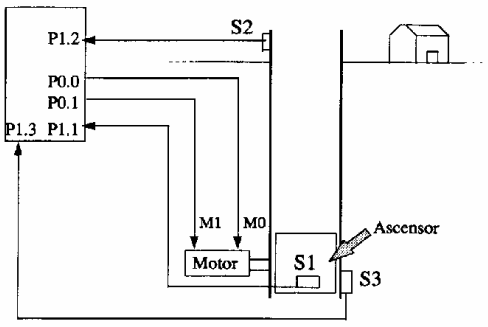
```
;************************************************************
; Programa para el contador de piezas
;************************************************************
            ORG   0H
            MOV   P1,#0       ;Pone a cero 4511.Corte Tr1,2,3 y 4
            MOV   R0,#0       ;Pone a cero los registros de digito
            MOV   R1,#0
            MOV   R2,#0
            MOV   R3,#0
            CLR   P2.4        ;Pone P2.4 a cero, alarma desactivada

;************************************************************
; Rutina Principal
;************************************************************
Principal:  JNB   P2.0,Cuenta
            JNB   P2.1,Descuenta
            JNB   P2.2,PaCero
```

```
Prin2:      CALL  Display
            SJMP  Principal
Cuenta:     CALL  CONTA
            SJMP  Prin2
Descuenta:  CALL DECRE
            SJMP Prin2
;************************************************************
;Rutina PaCero
;************************************************************
            MOV   R0, #0            ;Borra unidades
            MOV   R1, #0            ;Borra decenas
            MOV   R2, #0            ;Borra centenas
            MOV   R3, #0            ;Borra millares
            CLR   P2.4              ;Borra alarma
            SJMP  Prin2
;************************************************************
;Rutina CONTA
;Registros afectados: R0, R1, R2, R3
;************************************************************
CONTA:      CJNE  R0, #9, Unidad    ;Compara unidad
            CJNE  R1, #9, Decena    ;Compara decena
            CJNE  R2, #9, Centena   ;Compara centena
            CJNE  R3, #9, Millar    ;Compara millar
            SETB  P2.4              ;Activa la alarma
            RET
Unidad:     INC   R0                ;Incrementa unidades
            RET
Decena:     MOV   R0, #0            ;Pone a 0 las unidades
            INC   R1                ;incrementa decenas
            RET
Centena:    MOV   R0, #0            ;Pone a 0 unidades y decenas
            MOV   R1, #0
            INC   R2                ;Incrementa centenas
            RET
Millar:     MOV   R0, #0            ;Pone a 0 unid., dec. y centenas
            MOV   R1, #0
            MOV   R2, #0
            INC   R3                ;incrementa millares
            RET
;************************************************************
;Rutina DECRE
;Registros afectados: R0, R1, R2 y R3
;************************************************************
DECRE:      CJNE  R0, #0, D-uni     ;Compara unidad
            CJNE  R1, #0, D-dece    ;Compara decena
            CINE  R2, #0, D-cent    ;Compara centena
            CJNE  R3, #0, D-mill    ;Compara millar
            SETB  P2.4              ;Activa la alarma
            RET
D_uni:      DEC   R0                ;Decrementa unidad

            RET
D_dece:     MOV   R0,#9             ;Pone a 9 unidad
            DEC   R1                ;Decrementa decena
            RET
D_cent:     MOV   R0,#9             ;Pone a 9 unidad
            MOV   R1,#9             ;Pone a 9 decena
            DEC   R2                ;Decrementa centena
            RET
D_mill:     MOV   R0,#9             ;Pone a 9 unidad
            MOV   R1,#9             ;Pone a 9 decena
```

```
            MOV   R2,#9             ;Pone a 9 centena
            DEC   R3                ;Decrementa millar
            RET

;************************************************************
; Subrutina Display (Muestra en dígitos el valor de la cuenta)
; Registros afectados: R0, R1, R2 y R3
;************************************************************

Display:    MOV   P1, R3      ;Millares en P1(4 bits altos de R3 cero)
            SETB  P1.4        ;Enciende millares. Trl conduce
            CALL  RETA        ;Retardo (tiempo de encendido de dígito)
            CLR   P1.4        ;Apaga millares. Trl en corte
            MOV   P1,R2       ;Centenas en P1(4 bits altos de R2 cero)
            SETB  P1.5        ;Enciende centenas. Tr2 conduce
            CALL  RETA        ;Retardo (tiempo de encendido de dígito)
            CLR   P1.5        ;Apaga centenas. Tr2 en corte
            MOV   P1, R1      ;Decenas en P1(4 bits altos de R1 cero)
            SETB  P1.6        ;Enciende decenas. Tr3 conduce
            CALL  RETA        ;Retardo (tiempo de encendido de dígito)
            CLR   P1.6        ;Apaga decenas. Tr3 en corte
            MOV   P1,R0       ;Unidades en P1 (4 bits altos de R0 cero)
            SETB  P1.7        ;Enciende unidades. Tr4 conduce
            CALL  RETA        ;Retardo (tiempo de encendido de dígito)
            CLR   P1.7        ;Apaga unidades. Tr4 en corte
            RET
;*********************************************************
; Subrutina de retardo RETA (Retardo de 1+2*123+2= 251useg)
;*********************************************************
RETA:       MOV   R7,#125
Buc1:       DJNZ  R7, Buc1
            RET
```

En este caso cada dígito se enciende durante 251 useg, aproximadamente, lo que supone una frecuencia de refresco del número visualizado de 996Hz. Las rutinas “Conta” y “Decre” son una modificación de las subrutinas iniciales.

10.7.- Control de un ascensor.

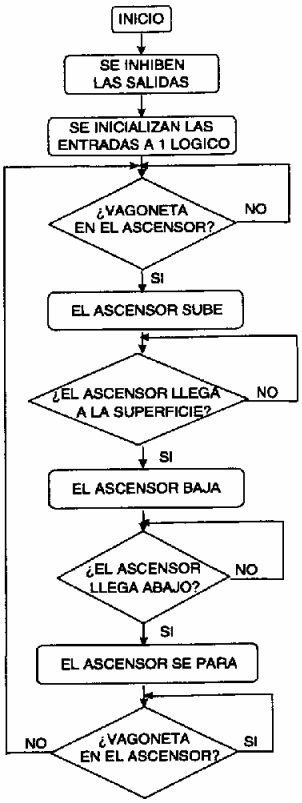


Esta aplicación consiste en el diseño del sistema de control, basado en el uC 8XC51, de un ascensor de carga de una mina de carbón. La tarea del ascensor consiste en elevar hasta la superficie las vagonetas cargadas de carbón. El ascensor se pone en funcionamiento cuando se coloca en su interior una vagoneta, a continuación asciende hasta que llega a la superficie, donde descarga de forma automática la vagoneta, y baja otra vez al interior de la mina con la vagoneta vacía.

Una vez que el ascensor llega abajo se detiene y los operarios sacan la vagoneta del ascensor, la llenan de carbón y la vuelven a colocar dentro del ascensor; se repite de nuevo la secuencia.

El sistema de control del ascensor incorpora tres sensores, S1, S2 y S3, activos a nivel alto, cuyo funcionamiento se describe a continuación:

- S1: Este sensor está colocado dentro del ascensor y detecta la presencia de la vagoneta, en cuyo caso se pone a 1 lógico. Si no hay vagoneta, el sensor se pone a 0 lógico.
- S2: Este sensor está colocado en la exterior de la mina y detecta que el ascensor ha llegado hasta la superficie.
- S3: Este sensor está colocado en el interior de la mina y detecta que el ascensor ha llegado al fondo de la mina.



Por otra parte, el sistema dispone de un motor que permite elevar o descender el ascensor. El motor está controlado mediante dos señales binarias, M1 y M0, que funcionan tal y como se indica en la tabla. En la figura se detalla la conexión entre los sensores, los actuadores y el microcontrolador 8XC51.

```
;*****************************************
; Control del ascensor
;*****************************************
ORG 0H
;Inicialización de los puertos
             SETB   P0.0          ;P0.0 y P0.1 a 1
             SETB   P0.1          ;motor detenido
             MOV    P1,#FFH       ;pines entrada
                                  ;a 1 lógico
; Programa de control
SIGUE:       JNB    P1.1,SIGUE    ;vagoneta en ascensor?
             CLR    P0.1          ;El ascensor sube
SIGUE1:      JNB    P1.2.SIGUE2   ;llega a superficie?
             SETB   P0.1          ;El ascensor baja
             CLR    P0.0
SIGUE2:      JNB    P1.3,SIGUE3   ;llega abajo ¿
             SETB   P0.0          ; se detiene
SIGUE4:      JB     P1.1,SIGUE4   ;sacan la vagoneta?
             JMP    SIGUE         ;vuelve al inicio
```

M1	M0	Funcionamiento del motor
0	0	El motor está parado
0	1	El motor gira a la derecha y el ascensor sube
1	0	El motor gira a la izquierda y el ascensor baja
1	1	El motor está parado

El programa que controla la secuencia de funcionamiento del ascensor debe detectar, mediante instrucciones de salto condicional, que se coloca una vagoneta en el interior del ascensor para, a continuación, activar el motor con el fin de subir el ascensor. Mientras el ascensor sube, el programa debe detectar cuándo llega a la superficie, testeando para ello el valor lógico de la entrada conectada al sensor S2. Cuando S2 se activa, se debe colocar en las salidas M0 y M1 el valor lógico 0 y 1, respectivamente, para que el ascensor baje. A continuación, se debe detectar que el ascensor ha llegado abajo, testeando el sensor S3 con una instrucción de salto condicional. Por último se detecta que se saca la vagoneta del ascensor, para seguidamente retornar al inicio del programa.

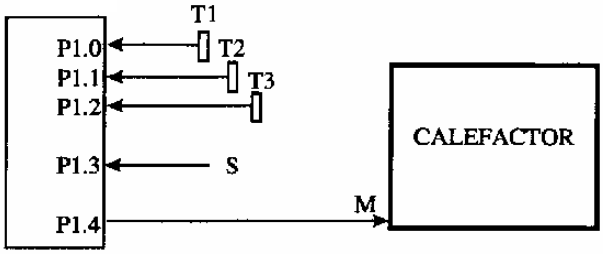
10.8.- Control de un calefactor.

Esta aplicación trata del diseño de un sistema de control de un calefactor, basado en el microcontrolador 8XC51. Para controlar el calefactor se disponen de 3 sensores de temperatura activos a nivel alto: T1, T2 y T3; de un selector de temperatura ambiente S y de un actuador M que enciende o apaga el calefactor. El funcionamiento de los sensores de temperatura se describe a continuación:

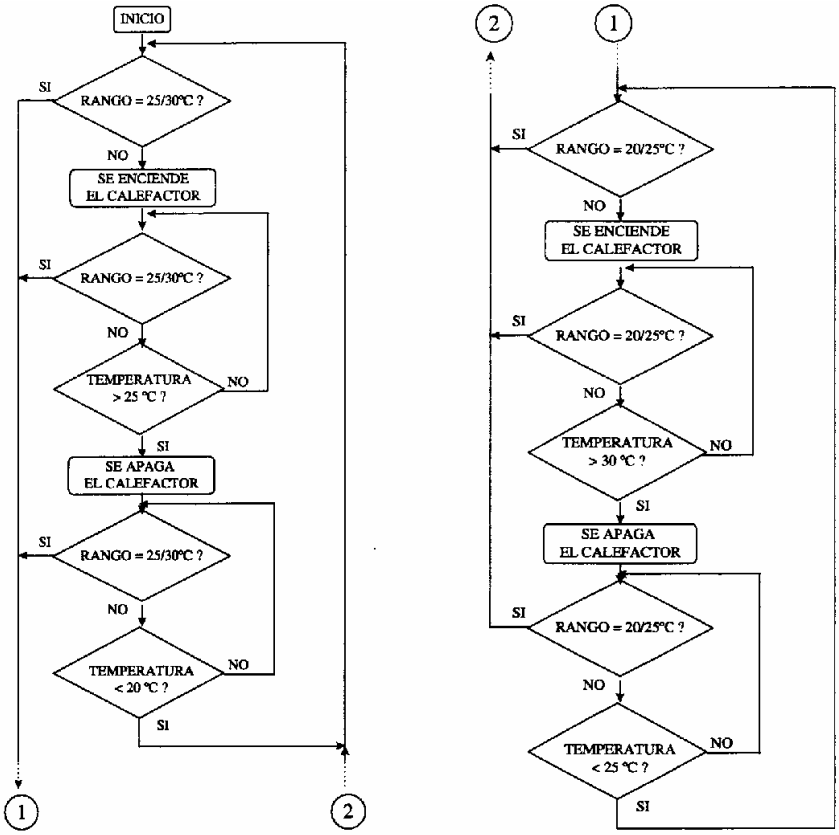
- T1: se activa a 1 lógico cuando la temperatura es igual o mayor que 20°C.
- T2: se activa a 1 lógico cuando la temperatura es igual o mayor que 25°C.
- T3: se activa a 1 lógico cuando la temperatura es igual o mayor que 30°C.

El selector de temperatura S es una entrada binaria que permite seleccionar entre dos rangos de temperatura ambiente: cuando S es igual a O lógico, la temperatura debe mantenerse entre 20 y 25°C.

Cuando S es igual a 1 lógico, la temperatura debe mantenerse entre 25 y 30°C. El microcontrolador 8XC51 encenderá el calefactor y pondrá la salida M a 1 lógico, cuando la temperatura sea inferior al valor mínimo del rango seleccionado con S. El calefactor debe permanecer encendido hasta que la temperatura supere el valor máximo del rango seleccionado, en cuyo caso, se debe apagar hasta que la temperatura sea de nuevo inferior al valor m'nimo del rango seleccionado.



El programa de control del calefactor debe detectar, en primer lugar, qué rango de temperatura está seleccionado. Esto se realiza testeando el valor lógico del pin P3.1. Si este bit está a 0 lógico, se salta a la rutina que mantiene la temperatura entre 20 y 25ºC, y si vale 1 lógico se ejecuta a la rutina que mantiene la temperatura entre 25 y 30°C.



Comprobando el valor lógico de los distintos sensores de temperatura y teniendo en cuenta el rango de temperatura seleccionado se decide si el calefactor debe estar activo o no. En la figura se presenta el flujograma de esta aplicación; a continuación se presenta el listado del programa.

```
;*****************************************
; Control del calefactor
;*****************************************
      ORG   0H
      MOV   C,P1.3      ;En primer lugar se lee el valor del
                        ;selector S para conocer en
                        ;que rango de temperatura debe
                        ;mantenerse la habitación

      JC    T25_30      ;Si S es igual a 1 lógico se salta
                        ;a la dirección T25_30
```

En el caso de que S sea igual a 0 lógico, se enciende el calefactor hasta que se activa el sensor T2, lo cual quiere decir que la temperatura ha superado los 25°C en cuyo caso se desactiva el calefactor. El calefactor permanece apagado hasta que se desactiva T1; entonces se vuelve a encender. En todo momento se comprueba la entrada S para saltar a la dirección T25_30 en el caso que valga 1 lógico.

```
;*****************************************
; Rutina del rango 20-25°C
;*****************************************
T20_25:     SETB  P1.4        ; Se activa el calefactor
SIGUE1:     JNB   P1.3,T25_30 ; Se comprueba si está el rango 20/25°C
            JNB   P1.1,SIGUE1 ; Si la T< 25°C, el calefactor
                              ; seguirá encendido
            CLR P1.4          ; Se apaga el calefactor
SIGUE2:     JNB   P1.3,T25_30 ; Se comprueba si rango 20/25°C
            JB    P1.0,SIGUE2 ; Si T>25 °C, el calefactor
                              ; seguirá apagado
            JMP   T20_25      ; Se vuelve al comienzo de la rutina
```

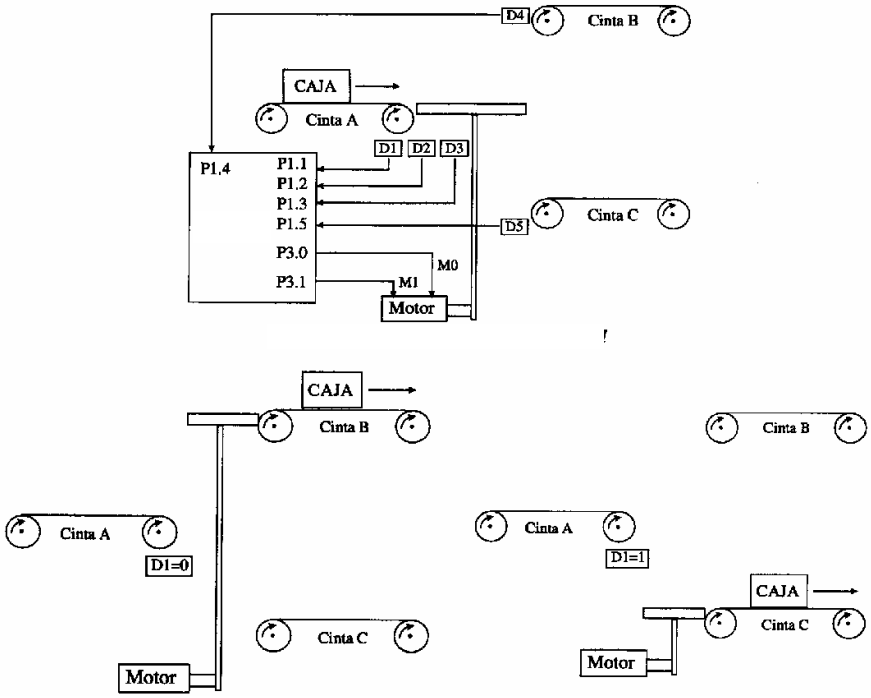
Cuando S es igual a 1 lógico, la temperatura de la habitación debe mantenerse entre 25 y 30 ºC. En este caso se debe encender el calefactor, y apagarse cuando se active el sensor T3. El calefactor permanecerá apagado hasta que se desactive T2, repitiéndose nuevamente la secuencia. Como en el caso anterior, en todo momento se sensa la variable S para saltar a la rutina T20_25 en el caso que valga 0 lógico.

```
;*****************************************
; Rutina del rango 25-30°C
;*****************************************
T25_30:     SETB  P1.4        ; Se activa el calefactor
SIGUE3:     JNB   P1.3,T20_25 ; Se comprueba si está el rango 25/30°C
            JNB   P1.2,SIGUE3 ; Si la T<30°C, el calefactor
                              ; seguirá encendido
            CLR P1.4          ; Se apaga el calefactor
SIGUE4:     JNB   P1.3,T20_25 ; Se comprueba si rango 25/30°C
            JB    P1.1,T25_30 ; Si T>20 °C, el calefactor
                              ; seguirá apagado
            JMP   T25_30      ; Se vuelve al comienzo de la rutina
```

10.9.- Control de una cinta elevadora.

En esta aplicación se debe diseñar un sistema de control de un elevador de una cinta transportadora basado en el microcontrolador 8XC51. La tarea del elevador es la llevar una caja de un determinado producto, que llega por la cinta A, hasta la cinta B o C. El sistema incorpora los siguientes detectores:

- D1: Este detector está colocado junto a la cinta A. Cuando llega una caja a la plataforma elevadora, este sensor decide si hay que llevarla a la cinta B o a la cinta C. Si D1 es igual a 0 lógico, hay que llevar el producto a la cinta B, y si D1 es igual a 1 lógico, a la cinta C.
- D2: Este sensor se activa (1 lógico) cuando la plataforma está a la altura de la cinta A.
- D3: Este sensor se activa (1 lógico) cuando llega un producto a la plataforma elevadora.
- D4: Este sensor se activa (1 lógico) cuando la plataforma elevadora ha llegado a la cinta B.
- D5: Este sensor se activa (1 lógico) cuando la plataforma elevadora ha llegado a la cinta C.



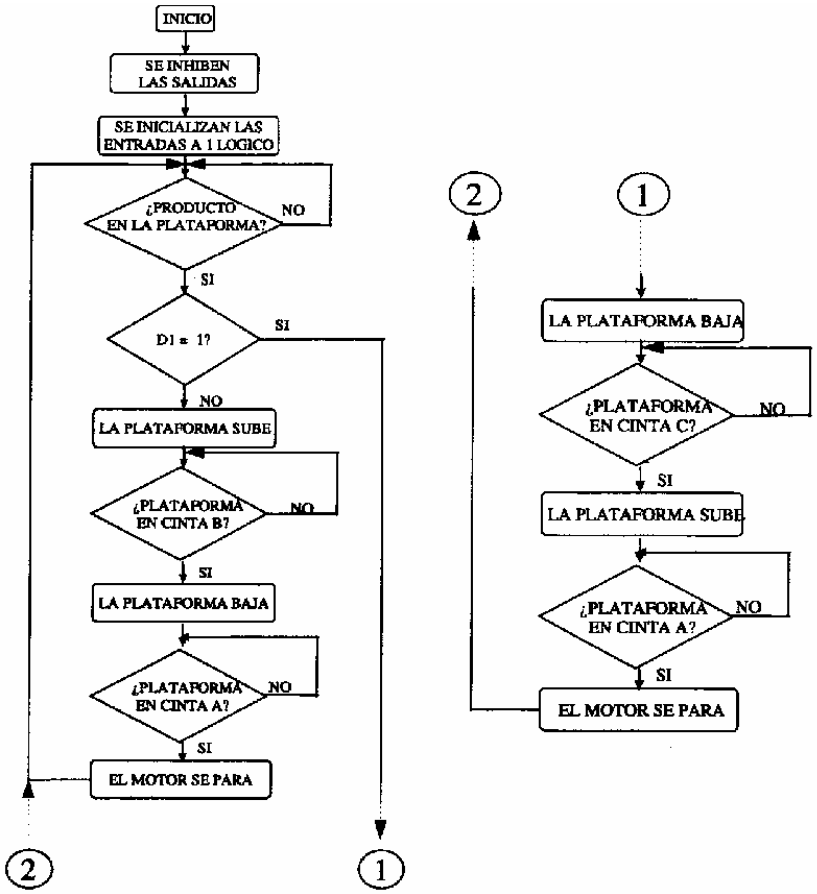
Por otra parte, el sistema dispone de un motor que permite ascender o descender la plataforma elevadora. El motor está controlado mediante dos señales binarias, M1 y M0, que funcionan tal y como se indica en la tabla.

M1	M0	Funcionamiento del motor
0	0	El motor está parado
0	1	El motor gira a la derecha y el ascensor sube
1	0	El motor gira a la izquierda y el ascensor baja
1	1	El motor está parado

El programa que controla el funcionamiento de la plataforma elevadora debe cumplir las siguientes indicaciones: cuando llegue un producto a la plataforma elevadora (D3 igual a 1 lógico) la plataforma deberá subir hasta la cinta B o bajar hasta la cinta C, dependiendo del valor que adquiera el detector D1. Una vez que la plataforma haya llegado a la cinta B o C, debe volver a la posición original, junto a la cinta A, para esperar a que llegue un nuevo producto;

entonces se vuelve a repetir el proceso. Inicialmente la plataforma elevadora se encuentra a la altura de la cinta A.

En una primera parte del programa se deben inicializar los puertos con el valor lógico adecuado. Los pines de los puertos que actúan como entradas de datos, han de estar inicializados a 1 lógico y los pines que trabajan como salida han de inicializarse por defecto con el nivel inactivo. En la figura siguiente se presenta el diagrama de flujo del programa de control de la plataforma elevadora y a continuación se detalla el listado.



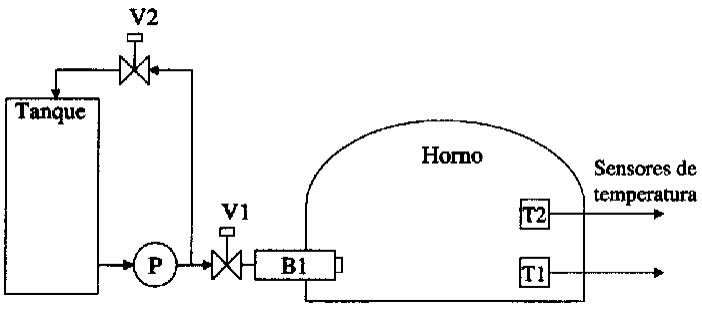

```
;***********************************************
; Control de la plataforma elevadora
;***********************************************
      ORG   00H   ; Programa a partir de dirección 00H
      ;     inicialización de las entradas y salidas
      CLR   P3.0  ; Los pines de salida se inicializan a 0 que es el
      CLR   P3.1  ; nivel inactivo en este caso
      SETB  P1.1  ; Pines correspondientes a puertos de entrada, como
      SETB  P1.2  ; están conectados a sensores, se inicializan
      SETB  P1.3  ; con 1, para evitar que se dañe el transistor
      SETB  P1.4  ; de salida
      SETB  P1.5
      ;     programa de control
SAL1: JNB   P1.3,SAL1   ; Se espera a que llegue producto a plataforma
      JB    P1.1,SAL2   ; Si D1=1 plataforma baja, caso contrario sube
      SETB  P3.0        ; La plataforma sube
      CLR   P3.1
SAL3: JNB   P1.4,SAL3   ; Se espera plataforma llegue arriba
      CLR   P3.0        ; La plataforma baja
      SETB  P3.1
```

```
SAL4: JNB   P1.2,SAL4   ; Se espera plataforma llegue a la cinta A
      CLR   P3.1        ; Paramos el motor
SAL2: CLR   P3.0        ; La plataforma baja
      SETB  P3.1
SAL5: JNB   P1.5,SAL5   ; Se espera plataforma baje hasta cinta C
      CLR   P3.1        ; La plataforma sube
      SETB  P3.0
SAL6: JNB   P1.1,SAL6   ; Se espera plataforma llegue a la cinta A
      CLR   P3.0        ; Se para el motor
      JMP   SAL1        ; Vuelta al inicio
```

10.10.- Control de la temperatura de un horno de cocción.

Se plantea, en este caso, una aplicación donde se controla la temperatura de un horno de cocción, mediante un microcontrolador 8XC51. Para poder controlar la temperatura del horno de cocción, el microcontrolador debe recibir la información del estado del sistema que le llega a través de los puertos, procesar esa información mediante el algoritmo de control correspondiente y actuar sobre el sistema, también a través de los puertos, para que el comportamiento del horno se mantenga dentro de los límites fijados. En concreto, la acción de control a la que se somete el horno, tiene como objetivo mantener su temperatura dentro del margen comprendido entre los 275°C y los 300°C. El horno de cocción, cuyo esquema está representado en la figura, contiene los siguientes elementos:

- Un tanque de fuel-oil, que almacena el combustible que alimenta el quemador.
- Una bomba P, que impulsa el fuel-oil hacia el quemador B1 instalado en el horno, o bien hacia el tanque.
- Dos electroválvulas V1 y V2. La electroválvula V1 se encarga de controlar el paso del fuel-oil al quemador B1. La electroválvula V2 se encarga de controlar el retorno del fiel-oil al tanque.
- Un horno de cocción, calentado por el quemador B1, que incorpora dos sensores de temperatura.



Para realizar el control del sistema se dispone de una serie de sensores y actuadores electromecánicos que permiten obtener información sobre el estado del sistema y actuar convenientemente sobre él. Todos los sensores y actuadores son activos a 1 lógico y presentan el siguiente funcionamiento:

1. ***Sensores***

    - T1: Detector de temperatura del horno que se activa cuando la temperatura del horno aumenta por encima de 275 °C.
    - T2: Detector de temperatura del horno que se activa cuando la temperatura del horno aumenta por encima de 300 °C.

- NL: Detector de nivel del tanque, que se activa cuando el nivel del tanque disminuye por debajo de un valor determinado.

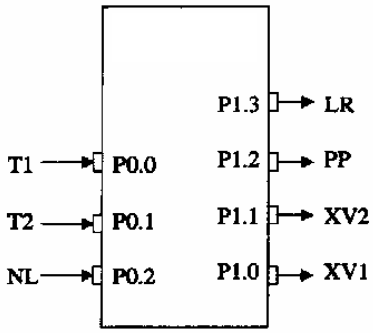
2. Actuadores

- PP: Accionador de la bomba P. Cuando se activa este actuador, la bomba P se pone en funcionamiento.
- XV1, XV2: Son los actuadores que abren o cierran las electroválvulas V1 y V2, respectivamente. Cuando se activan, la válvula correspondiente se abre y permite el paso del fuel-oil al quemador o al tanque.
- LR: Además de los detectores y activadores antes descritos, el sistema dispone de un indicador rojo cuya activación, a 1 lógico, indica que el horno se halla fuera de servicio.

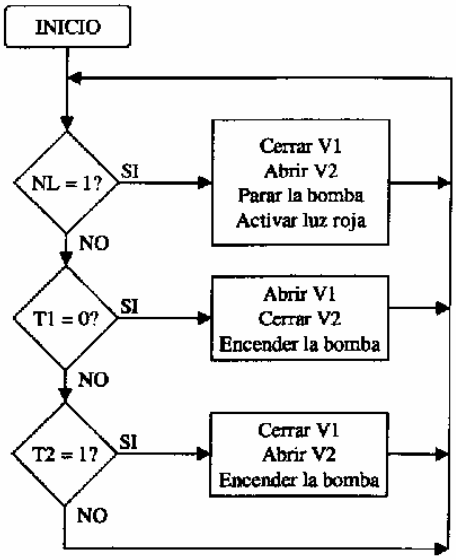
La estrategia de control, que debe implementar el sistema de control del horno, se puede resumir en tres puntos:

1. Si el nivel del tanque disminuye por debajo del nivel indicado por el sensor NL, se debe parar la bomba, abrir la electroválvula V2, cerrar V1 y señalizar que el sistema está fuera de servicio activando el indicador rojo.
2. Si la temperatura del horno desciende por debajo de 275ºC, debe alimentarse el quemador con combustible abriéndose la electroválvula V1, cerrándose la electroválvula V2, y activándose la bomba.
3. Si la temperatura del horno supera el valor de 300ºC, se debe suspender la alimentación del quemador cerrándose la electroválvula V1, abriéndose la electroválvula V2 y activándose la bomba.

En la figura siguiente está representado el esquema eléctrico de la conexión del microcontrolador 8XC51 con los diferentes sensores y actuadores que intervienen en esta aplicación.



En la figura siguiente está representado el diagrama de flujo de esta aplicación. Están reflejadas las tres opciones posibles, planteadas por la estrategia de control y la actuación que se debe llevar a cabo en cada caso.



Las primeras instrucciones del programa de control deben colocar, por motivos de seguridad, todos los pines de salida a su nivel inactivo, ya que las salidas sólo se deben activar después de procesar los datos de entrada, mediante el programa de control correspondiente. De igual modo, todos los pines de entrada deberán ponerse a 1 lógico para evitar la destrucción del **driver** de salida correspondiente. A continuación se detalla el listado del programa resultante.

```
;*******************************
; Control del horno
;*******************************

            ORG   00H          ; Dirección de inicio del progama
            MOV   Pl,#00H      ; Se inicializan salidas a nivel inactivo
            SETB  P0.0         ; Se inicializan pines de entrada a 1
            SETB  P0.1
            SETB  P0.2
INICIO:     JNB   P0.2,SALT1   ; Si nivel del depósito por debajo valor
            CLR   P1.2         ; permitido se para la bomba,
            CLR   P1.0         ; se cierra la electroválvula V1,
            SETB  P1.l         ; se abre la electroválvula V2,
            SETB  P1.3         ; y se enciende la luz roja
            JMP   INICIO       ; salto al INICIO
SALT1:      JB    P0.0,SALT2   ; Si el sensor T1 está inactivo
            SETB  P1.0         ; se abre la electroválvula V1,
            SETB  P1.2         ; se activa la bomba,
            CLR   P1.1         ; se cierra la electroválvula V2,
            CLR   P1.3         ; y se apaga la luz roja
            JMP   INICIO       ; Salto al INICIO
SALT2:      JNB   P0.1,INICIO  ; Si el sensor T2 está activo:
            CLR   P1.0         ; Se cierra la electroválvula V1,
            SETB  P1.2         ; Se activa la bomba,
            SETB  P1.l         ; Se abre la electroválvula V2,
            CLR   P1.3         ; y se apaga la luz roja
            JMP   INICIO       ; salto al INICIO
```