TERCER CURSO. INFORMÁTICA INDUSTRIAL II

Escuela Politécnica Superior
Universidad de Huelva

Tema 2:
Desarrollo de Algoritmos para sistemas empotrados

Manuel Sánchez Raya
Versión 0.91
12 de Enero de 2004

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 desarrollo de algoritmos tiene la finalidad de resolver los problemas comunes de programación mediante el diseño de rutinas que sean lo más genéricas posibles con el fin de que puedan emplearse independientemente de la arquitectura del microcontrolador empleado. Estudiaremos la solución a los problemas más sencillos, como la transferencia de bloques de datos, realización y consulta de tablas de datos, conversión de datos entre distintos formatos, 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.

Las tablas de datos siempre se colocan en la memoria ROM, en el espacio de código. Este espacio de direccionamiento se identifica por el compilador mediante el modificador de variable "code".

Mediante tablas se puede también obtener números primos según un índice por ejemplo 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

```
/************************************************
  Funcion Bin_Ascii
  Convierte un número hexadecimal en ASCII
 ************************************************/
unsigned char code
tabla_ascii[]={0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
                        0x38,0x39,0x41,0x42,0x43,0x44,0x45,0x46};

unsigned char Bin_Ascii(unsigned char car) {
      return tabla_ascii[car];
   }
```

El núcleo de esta subrutina es el acceso al array unidimensional "tabla_ascii". Si "car" vale 0 la función retornará el dato 0x30, puesto que la variable "car" se usa como índice del array y 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.

Cuando programamos el 8051, 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.

En ensamblador la transferencia de bloques 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.

En lenguaje C el compilador añade una extensión al lenguaje consistente en un modificador de punteros que fija la zona de memoria a la que se refiere el puntero en si.

Zona de memoria 8051	Modificador	Tipo puntero	Cantidad máxima de datos	Depuración
Interna de Datos	data	char	256 bytes	D:
Externa de Datos	xdata	int	64 Kbytes	X:
Código	code	int	64 Kbytes	C:
Acceso Indirecto	idata	char	128 bytes	I:

Por ejemplo, la rutina de transferencia de datos siguiente transfiere un bloque de memoria de la zona de código a la zona de datos externa.

```
/************************************************
 Funciones TRANS (transferencia de datos)
 Punteros: ptr1, ptr2
 Tamaño: tam
************************************************/
void trans(unsigned char data *ptr1,unsigned char code *ptr2, unsigned
int tam) {
int i;

      i=0;
      while (i<tam) {
            *(ptr1+i)=*(ptr2+i); i++;
      }
}
```

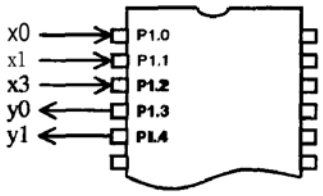
Con esta construcción es posible transferir datos desde cualquier tipo de zona de memoria a cualquier otra. Sin embargo hay que tener de antemano fijado el tipo de datos de los punteros, esto quiere decir que si necesitamos acceder a un bloque de datos en zona de código no podemos emplear el mismo tipo de puntero que en el caso de que los datos estén en otra zona de memoria. O sea los punteros tienen tipo y cada puntero apunta a una zona de memoria en concreto. Esto lo podemos observar en el simulador del compilador cuando le solicitamos que nos muestre una determinada zona de memoria, la zona de código se indica mediante el **prefijo C:**, la zona de datos mediante **D:** y la de datos externa mediante el **prefijo X:**.

3.- Funciones booleanas

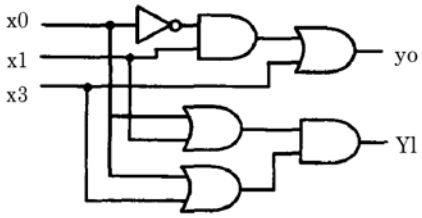
Los microcontroladores de la familia MCS-51 están capacitados para realizar **funciones booleanas**, tanto a nivel de bit como a nivel de byte. A nivel de bit, el microcontrolador puede realizar funciones lógicas y realizar otras tareas como leer el estado de un teclado, activar un relé, etc.

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 X3. 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.

```
#include <reg51.h>

/*********************************
 Funcion    F_Bol
 Entradas:  P1.0, P1.1, P1.2
 salidas:   P1.3, P1.4
 *********************************/

/* entradas */
sbit X0=P1^0;
sbit X1=P1^1;
sbit X3=P1^2;
/* salidas */
sbit Y0=P1^3;
sbit Y1=P1^4;

void F_Bol(void) {
      Y0=(~X0&X1)|X3;
      Y1=(X0|X1)&(X0|X3);
}
```



Funciones lógicas



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 la variable msec.

```
/*********************************************
 Subrutina RETARDO

 *********************************************/

void retardo(int msec) {
int i;

while (msec--)
      for(i=0;i<100;i++);
}
```

El bucle interno formado por la variable i está dentro del bucle externo formado por la variable msec; a cada vuelta del bucle externo le corresponden 100 ejecuciones del bucle interno.

Una alternativa a la rutina anterior consiste en la siguiente función:

```
/*********************************************
 Subrutina RETARDO2

 *********************************************/
```

```
void retardo2(int msec) {
int i,j;

while (msec--)
      for(i=0;i<100;i++)
            for(j=0;j<100;j++);
}
```

En esta función se han realizado dos bucles anidados con diferentes contadores para alargar más el tiempo de retardo.

5.- Operaciones matemáticas.

La suma/resta y multiplicación/división de datos en lenguaje C se simplifica hasta tal punto que cuando programamos los microcontroladores que no tienen instrucciones de multiplicar y dividir, como sucede con los PIC, es la librería estándar de C la encargada de realizar estas operaciones. Tan solo hay que tener en cuenta que en la programación de estos dispositivos no se debe emplear punto flotante.

En lugar de emplear los tipos float y double, deben emplearse tipos enteros y realizar un escalado acorde con la precisión que necesitemos en la aplicación. Por ejemplo, si queremos realizar la función raíz cuadrada, es mejor emplear una tabla y almacenar todos los resultados multiplicados por 1000 si queremos disponer de tres dígitos decimales de precisión.

```
/*******************************************
 Subrutina raiz_cuadrada (Raiz cuadrada por 1000)
 Entrada: valor entre 0 y 10 (restringido y multiplicado por 10)
 Salida: El resultado multiplicado por 1000

*******************************************/

unsigned int code
tabla_raiz[10]={0,1000,1414,1732,2000,2236,2449,2645,2828,3000};

unsigned int raiz_cuadrada(unsigned int var) {

if (var==0) return 0;
if (var<10) return tabla_raiz[var];
else return 0;
}
```

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 basta con emplear variables del tamaño apropiado e instrucciones aritméticas; 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 y el micro solo maneja formato binario.

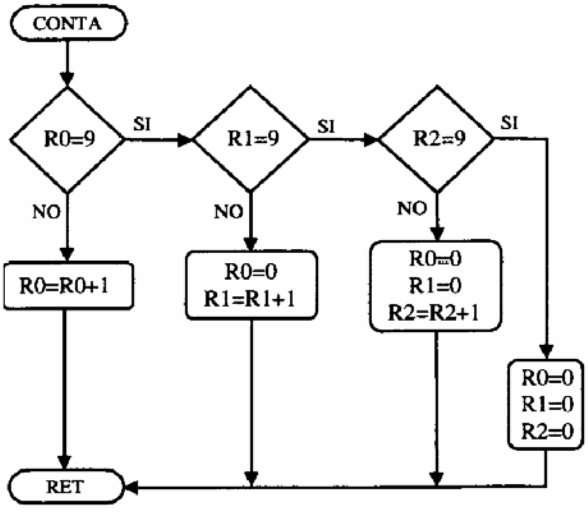
El siguiente ejemplo muestra una subrutina para realizar un contador en BCD de cuatro cifras, es decir, que el máximo valor que puede alcanzar es 999. Cada cifra del contador esta contenida

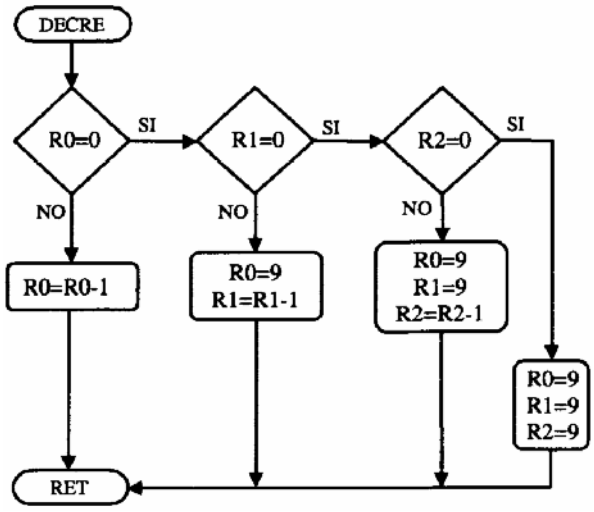
en un char: las unidades están en la variable *unidades*, las decenas en *decenas* y las centenas en *centenas*. De esta forma, para visualizar posteriormente el valor del contador, sólo se ha de volcar los cuatro bits menos significativos de cada una de las variables en el visualizador numérico correspondiente, ya sea de tipo LED o LCD.

```
/*********************************************
  Funcion Conta
  Incrementa unidades
 *********************************************/

unsigned char uni=0,dec=0,cen=0;

void Conta(void) {

if (uni==9) {
      if (dec==9) {
            if (cen==9) uni=dec=cen=0;
            else {uni=dec=0;cen++;}
      }
      else {uni=0;dec++;}
}
else uni++;

}
```

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 incrementan. 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:

```
/*********************************************
  Funcion Decre
  Decrementa unidades
*********************************************/

void Decre(void) {

if (uni==0) {
      if (dec==0) {
            if (cen==0) uni=dec=cen=9;
            else {uni=dec=9;cen--;}
      }
      else {uni=9;dec--;}
}
else uni--;
}
```

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 en ensamblador 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 variables "dig01" y "dig23" harán de contador; los cuatro bits bajos de "dig01" serán las unidades y sus cuatro bits altos, las decenas, los cuatro bits bajos de "dig23" serán las centenas y sus cuatro bits altos la unidad de millar del contador. Con un contador de cuatro cifras se puede llegar hasta un valor de 9999, es decir, ambas posiciones contienen el valor 0x99. A continuación se muestra la rutina CONTA que hace esta función:

```
/*****************************************************
  Funcion Conta2
  Ajuste decimal en C
*****************************************************/

unsigned char dig01=0x00,dig23=0x00;

void Conta2(void) {

/* incrementa contador */
dig01++;
/* ajuste decimal en unidades */
if ((dig01&0x0F)>9) dig01+=6;
/* ajuste decimal en decenas */
if ((dig01&0xF0)>0x90) dig01+=0x60;

/* si ha habido acarreo se incrementan
 centenas y se vuelve a ajustar */
if (dig01==0) {
      dig23++;
      if ((dig23&0x0F)>9) dig23+=6;
      if ((dig23&0xF0)>0x90) dig23+=0x60;
      }
}
```

Esta rutina incrementa en una unidad el valor de las variables dig01 y dig23. La función, al principio, incrementa la variable dig01 de 0x00 a 0x01, de 0x01H a 0x02, y así sucesivamente

hasta llegar a 0x09, pues la comprobación del if no modifica el valor de la variable tras la instrucción de suma. En el siguiente incremento el valor de dig01 pasa de 0x09 a 0x0A, por lo que el cuerpo del primer if transforma este dato en 0x10, al ser el nibble bajo mayor que 9. En los siguientes incrementos no se modificará la variable, al menos hasta que llegue al valor de 0x1A, momento en el cual lo transformará en 0x20. Así, el algoritmo de ajuste decimal (que es lo mismo que realiza la instrucción DA A de ajuste decimal) actúa en los instantes en que la variable vale 0xlA, 0x2A, 0x3A, ... 0x9A; este último caso lo transforma en 0x00 y con la segunda comprobación incrementa en una unidad el contenido de la variable dig23, pasando el contador a valer 0x0100.

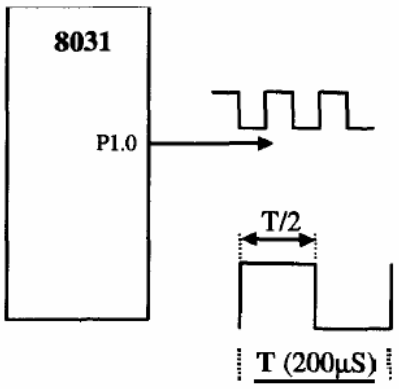
La rutina CONTA basada en la instrucción de ajuste decimal tiene un menor tamaño que la anterior y, además, permite extender el máximo valor del contador a cualquier tamaño; basta, para ello, con aumentar el número de bytes del contador, aumentado el número de posiciones de memoria que ocupa.

7.- 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 generalmente una función principal, función “Main” y varias funciones auxiliares colocadas delante de ésta. Al principio de la función main se configuran los recursos internos del microcontrolador y el modo de operar de éste. Una vez realizada esta inicialización, se pasa al bucle general del programa, realizado normalmente con una estructura while o do..while, que se encarga de llevar a cabo las funciones requeridas por la aplicación. El cuerpo de la función main formará un bucle infinito, pues las aplicaciones a resolver necesitan que el microcontrolador esté constantemente pendiente de las tareas que debe efectuar. La parte de inicialización en algunos ejemplos no aparecerá, porque los programas son sencillos, y no será necesario poner un valor inicial a las variables empleadas.

7.1 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 el complemento de bit, y para generar un retardo de 100 us se utilizará una función de retardo como la descrita en el apartado 4.

```
#include <reg51.h>

/*********************************************
  Programa de generación de una señal cuadrada
 *********************************************/
```

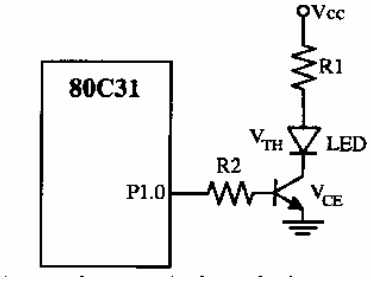
```
/* definición de salida */
sbit SALIDA=P1^0;

void retardo(unsigned char msec) {

while (msec--);
}

void main(void) {
/* Bucle infinito en función main */
while(1) {
      /* Complementa estado lógico de P1.0 */
      SALIDA=~SALIDA;
      /* Llama a la rutina de retardo */
      retardo(18);
      }
}
```

Una vez compilado el programa se obtiene el siguiente listado en ensamblador :

```
RSEG  ?PR?_retardo?GENERADOR
?C0001:
      MOV   R6,R7
      DEC   R7
      MOV   A,R6
      JNZ   ?C0001
      RET

RSEG  ?PR?main?GENERADOR
?C0004:
      CPL   SALIDA
      MOV   R7,#012H
      LCALL _retardo
      SJMP  ?C0004
```

Para que la instrucción 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 LCALL se ejecuta en 2 CM, DEC en 1 CM, MOV en l CM, JNZ en 2 CM y RET en 2 CM. El tiempo de ejecución de la función de retardo viene dado por:

$$T_{Ret} = 5 + 5 \cdot R7 + 2 \ (CM)$$



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

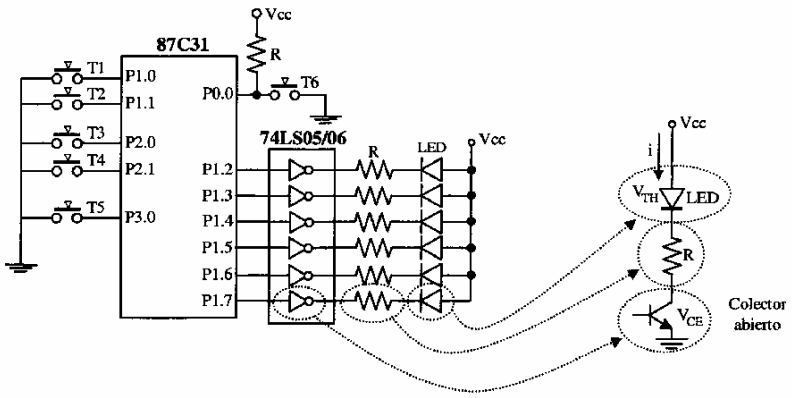
$$t_{CPL} = 1 + 2 + 2 + t_{Ret} = 100\,CM \text{ o } 100\,\mu s$$

Se deduce que t_{Ret} debe valer 95 us, y que el registro R7 debe valer aproximadamente 18. 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. Aproximadamente unos 330 ohmios.

La resistencia R2 se debe determinar para que el transistor entre en saturación cuando la salida de P1.0 esté a 1 1ógico. Con unos 10K ohmios se consigue la saturación.

7.2 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.



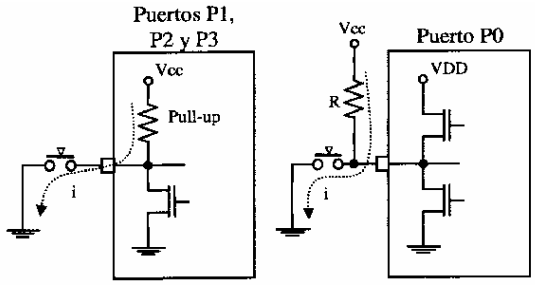
En la figura se emplea el circuito integrado 74LS05, o el 74LS06, que contiene hasta 6 puertas inversoras en colector abierto. El terminal de salida de la puerta inversora es el colector de un transistor interno, cuyo emisor está conectado a masa, tal y como se ve en el detalle de esta figura. La puerta 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 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.

```
#include <reg51.h>
/**************************************
 Programa para la lectura de la teclas
**************************************/
/* Definicion de teclas */
sbit Tecla_T1=P1^0;
sbit Tecla_T2=P1^1;
sbit Tecla_T3=P2^0;
sbit Tecla_T4=P2^1;
sbit Tecla_T5=P3^0;
sbit Tecla_T6=P0^0;
/* Definicion de LEDS */
sbit Led_L1=P1^2;
sbit Led_L2=P1^3;
sbit Led_L3=P1^4;
sbit Led_L4=P1^5;
sbit Led_L5=P1^6;
sbit Led_L6=P1^7;

/**************************************
 Rutina Principal (Testeo de teclas,encendido/apagado de leds)
**************************************/
void main(void) {
/* pongo a 1 entradas */
/* aunque despues del reset ya estan a uno */
Tecla_T1=Tecla_T2=Tecla_T3=Tecla_T4=Tecla_T5=1;


while (1) {
/* compruebo teclado */
       if(!Tecla_T1) P1=0x07; /*0000 0lllb Enciende P1.2 apaga resto*/
       else if(!Tecla_T2) P1=0x0B; /*0000l0llb Enciende P1.3 apaga resto*/
       else if(!Tecla_T3) P1=0x13; /*0001 00llb Enciende P1.4 apaga resto*/
       else if(!Tecla_T4) P1=0x23; /*0010 0011b Enciende P1.5 apaga resto*/
       else if(!Tecla_T5) P1=0x43; /*0l00 0011b Enciende P1.6 apaga resto*/
       else if(!Tecla_T6) P1=0x83; /*1000 0011b Enciende P1.7 apaga resto*/
       /* Apaga solo leds conectados a P1.2-7 */
       else P1&=0x03;
       }
}
```

Se puede escribir en cada patilla del puerto P1, sustituyendo las instrucciones P1&=0x03, con la instrucción:

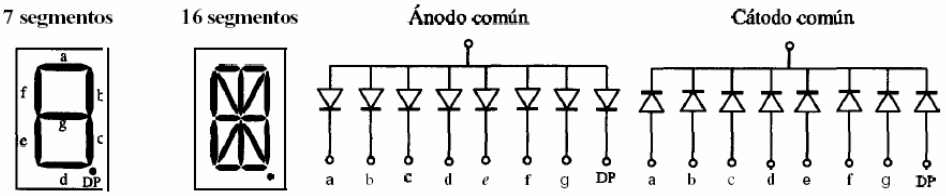
```
Led_L1=Led_L2=Led_L3=Led_L4=Led_L5=Led_L6=0;
```

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 al inicio del bucle while, 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.

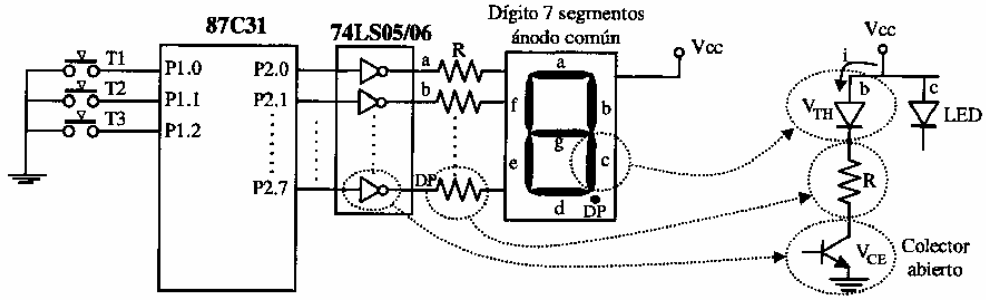
7.3.- Conexión de un dígito de 7 segmentos.

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.



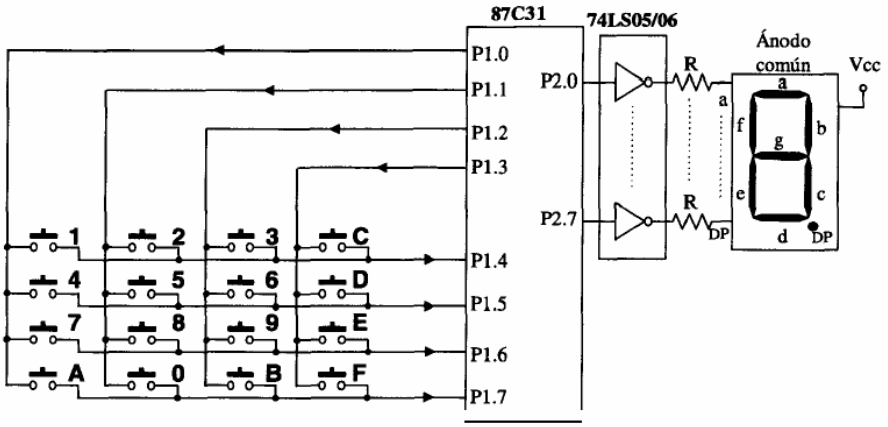
En la conexión se utiliza, del mismo modo que el ejemplo anterior, un par de circuitos integrados 74LS05 ó 74LS06 que contienen hasta 6 puertas inversoras en colector abierto. El digito utilizado es de ánodo común, el nodo está conectado a la tensión de alimentación, Vcc, y cada diodo a una resistencia 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:

```
#include <reg51.h>
/*******************************************************
 Programa para la conexión de un digito de 7-segmentos
*******************************************************/
/* definición teclas */
sbit Tecla_T1=P1^0;
sbit Tecla_T2=P1^1;
sbit Tecla_T3=P1^2;

#define DISPLAY P2
/*******************************************************
 Rutina principal
*******************************************************/
void main(void) {

DISPLAY=0; /* Apaga todos los leds del dígito */
while (1) {
      /* Comprueba si se ha pulsado T1 */
      if (!Tecla_T1) DISPLAY=0x77; /* 0111 0111b Pone letra A */
      /* Comprueba si se ha pulsado T1 */
      else if (!Tecla_T2) DISPLAY=0x7C; /* 0111 1100b Pone letra B */
      /* Comprueba si se ha pulsado T1 */
      else if (!Tecla_T3) DISPLAY=0x79; /* 0111 1001b Pone letra C */
      else DISPLAY=0;
      }
}
```

7.4 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.

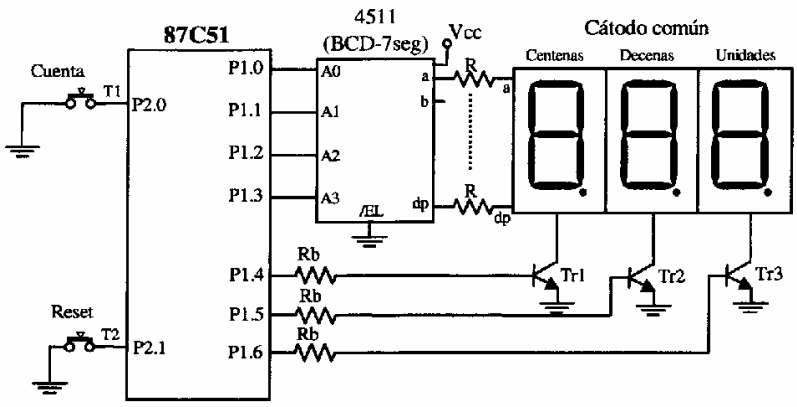
```
#include <reg51.h>
#define TECLAS P1
/************************************************
 Programa de lectura de teclado matricial
************************************************/

/************************************************
 Función para la detección de la tecla pulsada
 Lee el puerto P1 y retorna 0 si no se pulsa una tecla
 y !=0 si se pulsa tecla
************************************************/
bit Det_tecla(void) {
      /* Si alguno de los cuatro bits altos vale cero debe devolver
      verdadero */
      if ((TECLAS>>4)^0x0F) return 1
      else return 0;
}
/* convierte tecla pulsada en numero de tecla con prioridad*/
unsigned char code cod_pri[]={0,1,0,2,0,1,0,3,0,1,0,2,0,3};/*
convierte numero de tecla en carácter a generar */
unsigned char code tabla_teclado[]={'1','4','7','A','2','5','8','0',
                        '3','6','9','B','C','D','E','F'};

unsigned char lee_teclado(void) {
P1=0xFE; /* 1111 lll0b  Habilita primera col. */
if (Det_tecla()) return tabla_teclado[cod_pri[TECLAS>>4]];
P1=0xFD; /* 1111 ll01b  Habilita segunda col. */
if (Det_tecla()) return tabla_teclado[cod_pri[TECLAS>>4]+4];
P1=0xFB; /* 1111 l011b  Habilita tercera col. */
if (Det_tecla()) return tabla_teclado[cod_pri[TECLAS>>4]+8];
P1=0xF7; /* 1111 0111b  Habilita cuarta col. */
if (Det_tecla()) return tabla_teclado[cod_pri[TECLAS>>4]+12];
return 0;
}

/************************************************
 Visualización de un carácter en 7 segmentos
************************************************/

unsigned char code ascii_7seg[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,
                              0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,
                              0x5E,0x79,0x71};

void Display(unsigned char cod_ascii) {
/* si intentamos visualizar codigos sale */
if (cod_ascii<0x30) return;
/* si intentamos visualizar números resta 0x30 */
if (cod_ascii<0x39) {
```

```
      P2=ascii_7seg[cod_ascii-0x30]; /* restamos 0x30=Ascii('0') */
      return;
}
/* si intentamos visualizar letras mayúsculas(A-Z) resta 0x36 */
if (cod_ascii<0x47) {
      P2=ascii_7seg[cod_ascii-0x37]; /* restamos 0x37=Ascii('A') */
      return;
}
/* Todo para obtener un índice de 0 a 15 para el array ascii_7seg */
}

/************************************************
; Funcion main
************************************************/
void main(void) {
unsigned char tecla;

P2=0; /* Apaga todos los leds del digito */
while (1) {
      if (tecla=lee_teclado())
            Display(tecla);
   }
}
```

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

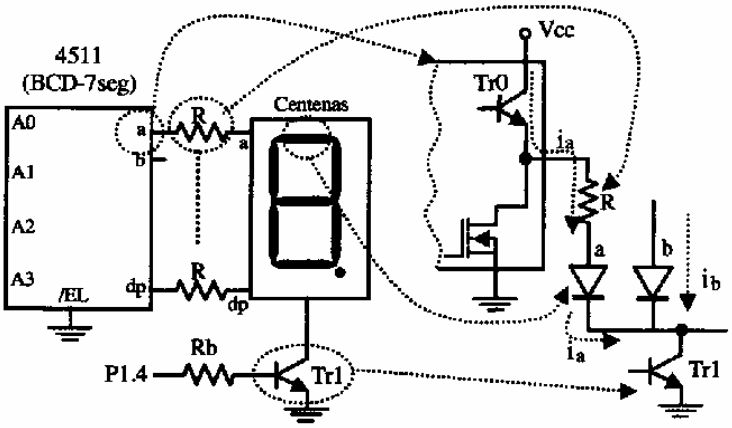
La figura muestra la forma de conectar varios dígitos de 7 segmentos a un 87C51 para una aplicación donde se desea controlar el turno de atención a un cliente, que es tan común en tiendas y locales comerciales. Debido al tipo de aplicación donde se ha de visualizar el número de orden del cliente, los 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.

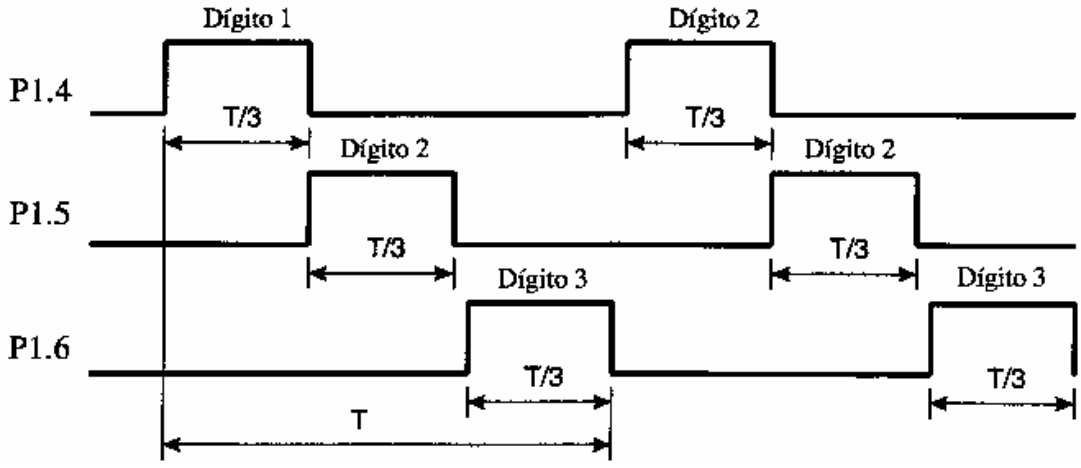
En esta aplicación se usará un microcontrolador 8751, el 4511, tres dígitos de 7 segmentos y tres transistores para controlar el encendido de cada uno de los dígitos. Por sencillez, las entradas /ILT y /IB del 4511 no se usarán, y la entrada /EL se conectará directamente a tierra, de forma que esté siempre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

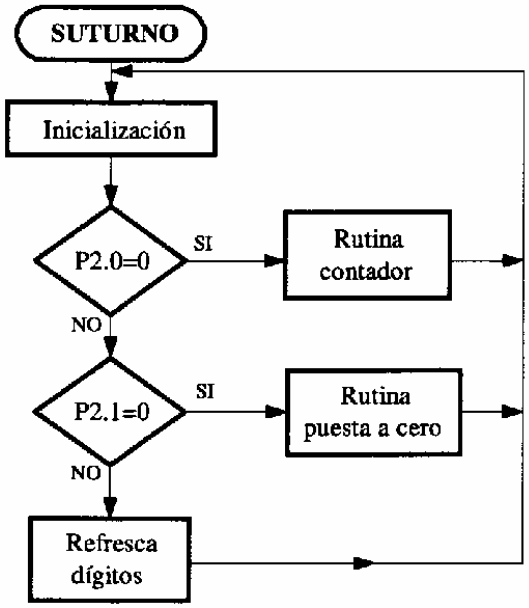
Para evitar el consumo excesivo por parte de los dígitos y para utilizar el circuito realizado, los dígitos se deben encender de manera secuenciada, es decir, primero se enciende el dígito de las centenas, luego el dígito de las decenas y, por último, el dígito de las unidades. Esta secuencia se puede 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 corrientes 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 000 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 anterior. En esta rutina intervienen las variables uni, dec, cen las cuales se emplean para las unidades, decenas y centenas respectivamente. El flujograma del programa se muestra en la figura siguiente.

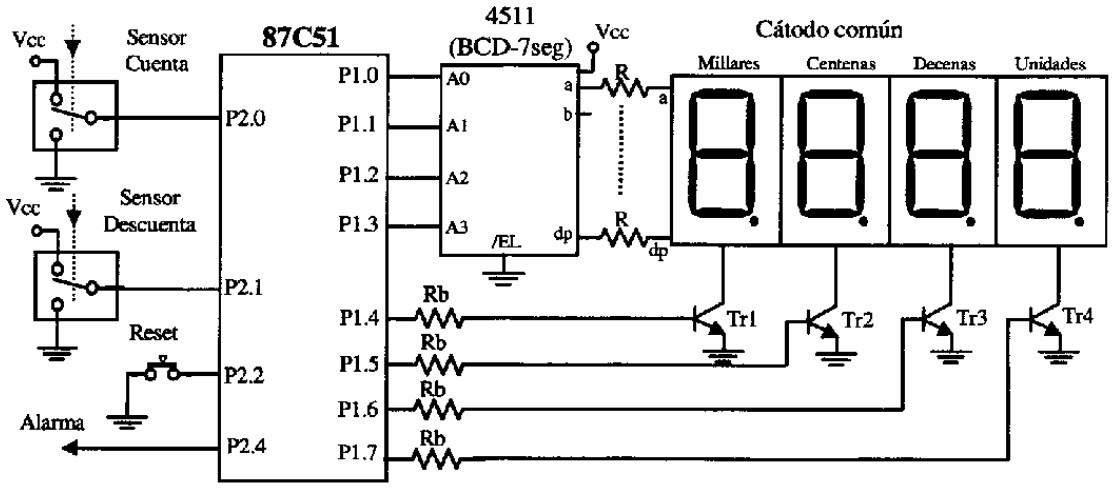
```
#include <reg51.h>

/* prototipo funcion delay */
void delay(unsigned int);

/***********************************************
  Programa para la gestión del turno de espera
 ***********************************************/

/* pulsadores */
sbit CUENTA=P2^0;
sbit RESET=P2^1;
/* salidas transistores */
sbit TRAN1=P1^4;
sbit TRAN2=P1^5;
sbit TRAN3=P1^6;

unsigned char uni,dec,cen; /* variables de cuenta */

/***********************************************
 Funcion CONTA: incrementa contador BCD
***********************************************/

void Conta(void) {
if (uni==9) {
      if (dec==9) {
            if (cen==1) uni=dec=cen=0;
            else {uni=dec=0;cen++;}
      }
      else {uni=0;dec++;}
}
else uni++;
}
```

```
/*********************************************
 Función Refresca: actualiza la pantalla numérica
*********************************************/
void Refresca(void) {
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=cen; /* coloco centenas */
TRAN1=1; /* enciendo centenas */
delay(10); /*espero un tiempo */
TRAN1=0; /* apago centenas */
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=dec; /* coloco decenas */
TRAN2=1; /* enciendo decenas */
delay(10); /*espero un tiempo */
TRAN2=0; /* apago decenas */
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=uni; /* coloco unidades */
TRAN3=1; /* enciendo unidades */
delay(10); /*espero un tiempo */
TRAN3=0; /* apago unidades */
}

/*********************************************
 Función Principal
*********************************************/
void main(void) {
 unsigned char i;

 P1=0x00; /* Pone a cero entrada del 4511 y en corte Tr1, Tr2 y Tr3 */
 uni=dec=cen=0;
 while (1) {
       if (!CUENTA) Conta();
       if (!RESET) uni=dec=cen=0;
       for (i=0;i<20;i++) Refresca(); /* muestra valor en pantalla */
       /* espera un poco para evitar rebotes de pulsadores */
      }
}
```

Cada uno de los dígitos se enciende durante un tiempo determinado por el número de instrucciones ensamblador que genere el compilador. La frecuencia de refresco del número visualizado debe ser mayor de 50Hz para que no se note parpadeo de los LED. Esta frecuencia puede ser superior, puesto que se tiene una limitación de frecuencia mínima, pero no en cuanto a frecuencia máxima. 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.

7.6.- 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 error en el proceso de control de calidad.

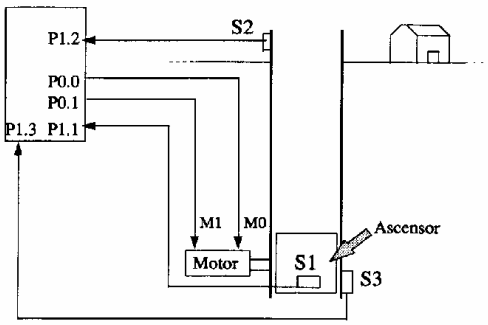
```
#include <reg51.h>
/*************************************************************
 Programa para el contador de piezas
*************************************************************/
#define OFF 0
#define ON 1
/* pulsadores */
sbit CUENTA=P2^0;
sbit DESCUENTA=P2^1;
sbit RESET=P2^2;
/* salidas transistores */
sbit TRAN1=P1^4;
sbit TRAN2=P1^5;
sbit TRAN3=P1^6;
sbit TRAN4=P1^7;
sbit ALARMA=P2^4;

/* prototipo de rutina retardo */
void delay(unsigned int);

unsigned char uni,dec,cen,mil; /* variables de cuenta */
/***********************************************
 Funcion Cuenta: incrementa contador BCD y señala alarma
***********************************************/
void Cuenta(void) {
if (uni==9) {
      if (dec==9) {
            if (cen==9)
                  if (mil==9) ALARMA=ON; /* señala alarma y no modifica
cuenta */
                  else {uni=dec=cen=0;mil++;}
            else {uni=dec=0;cen++;}
      }
      else {uni=0;dec++;}
}
else uni++;
```

```
}
/***********************************************
 Funcion Descuenta: decrementa contador BCD
***********************************************/
void Descuenta(void) {

if (uni==0) {
      if (dec==0) {
            if (cen==0)
                  if(mil==0) ALARMA=ON;
                  else {uni=dec=cen=9;mil--;}
            else {uni=dec=9;cen--;}
      }
      else {uni=9;dec--;}
}
else uni--;
}
/***********************************************
 Funcion Refresca: actualiza la pantalla numerica
***********************************************/
void Refresca(void) {
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=mil; /* coloco millares */
TRAN1=1; /* enciendo millares */
delay(10); /*espero un tiempo */
TRAN1=0; /* apago millares */
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=cen; /* coloco centenas */
TRAN2=1; /* enciendo centenas */
delay(10); /*espero un tiempo */
TRAN2=0; /* apago centenas */
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=dec; /* coloco decenas */
TRAN3=1; /* enciendo decenas */
delay(10); /*espero un tiempo */
TRAN3=0; /* apago decenas */
P1=P1&0xF0; /*pongo a cero las salidas */
P1|=uni; /* coloco unidades */
TRAN4=1; /* enciendo unidades */
delay(10); /*espero un tiempo */
TRAN4=0; /* apago unidades */
}

/***********************************************************
 Función Principal
***********************************************************/
void main(void) {
unsigned int i;

P1=0x00;                /* Pone a cero 4511.Corte Tr1,2,3 y 4 */
uni=dec=cen=mil=0;      /* Pone a cero los registros de digito */
ALARMA=OFF;             /* Pone P2.4 a cero, alarma desactivada */

while (1) {
      if (!CUENTA)      Cuenta();   /* incrementa */
      if (!DESCUENTA) Descuenta();  /* decrementa */
      if (!RESET) {
                  uni=dec=cen=mil=0; /* pone a cero */
                  ALARMA=OFF;       /* desactiva alarma */
                  }
      for(i=0;i<10;i++) Refresca(); /* Repite para evitar rebote */
```

```
        }
}
```

Las rutinas “Conta” y “Decre” son una modificación de las subrutinas iniciales.

7.7.- Control de un ascensor.

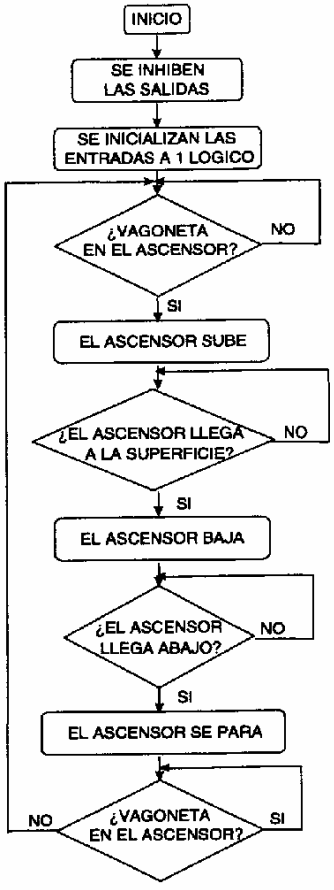


Esta aplicación consiste en el diseño del sistema de control, basado en el uC 8XC51, de un ascensor de carga de una mina de carbón. La tarea del ascensor consiste en elevar hasta la superficie las vagonetas cargadas de carbón. El ascensor se pone en funcionamiento cuando se coloca en su interior una 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.




```
#include <reg51.h>
/*****************************************
 Control del ascensor
*****************************************/
/* entradas sensores */
sbit VAGONETA=P1^1;
sbit SUPERFICIE=P1^2;
sbit SOTANO=P1^3;
/* defines de motores */
#define PARA 0x03
#define SUBE 0x01
#define BAJA 0x02

void main(void) {
P0|=PARA; /* Inicialización puertos, motor detenido */
P1=0xFF; /* P1 entrada */
while (1) {
       while (!VAGONETA); /* espera vagoneta */
       P0|=SUBE; /* si hay vagoneta sube */
       while (!SUPERFICIE); /* espera llegar arriba */
       P0|=BAJA; /* para ascensor */
       while (!SOTANO); /* espera llegar abajo */
       P0|=PARA;
```

```
        While (VAGONETA); /* espera sacar vagoneta */
        }
}
```

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 sentencias while vacías que congelan el programa hasta que no se alcanza la condición de salida. La primera condición de salida es 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 (SUPERFICIE). 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 un while vacío. Por último se detecta que se saca la vagoneta del ascensor, para seguidamente retornar al inicio del programa.

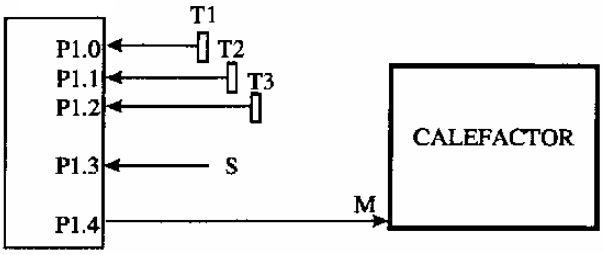
7.8.- Control de un calefactor.

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:

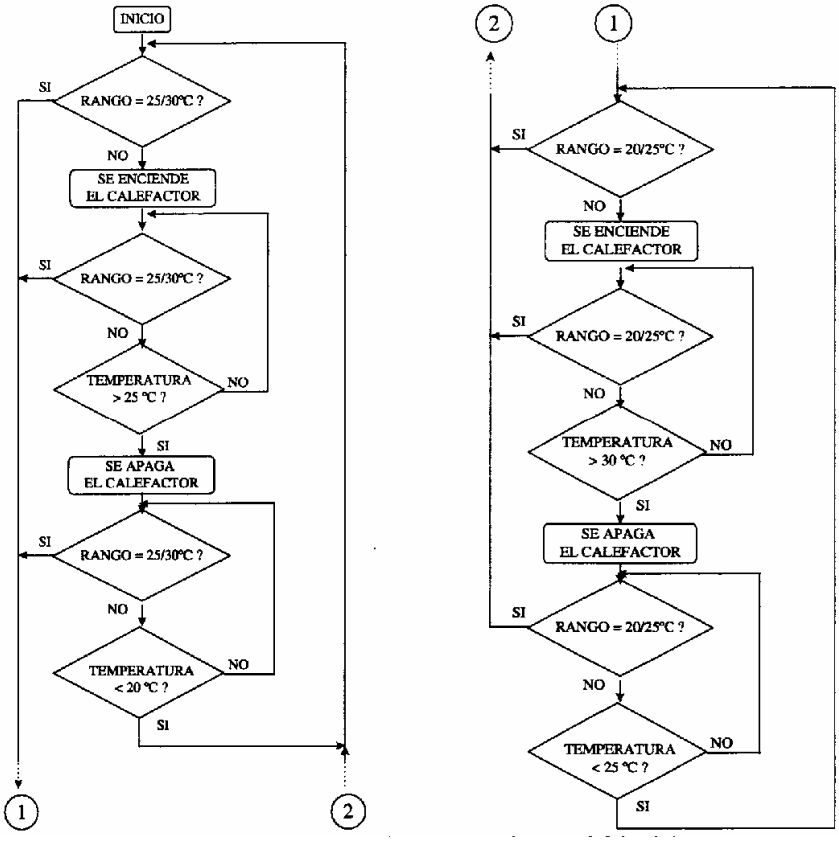
- T1: se activa a 1 lógico cuando la temperatura es igual o mayor que 20°C.
- T2: se activa a 1 lógico cuando la temperatura es igual o mayor que 25°C.
- T3: se activa a 1 lógico cuando la temperatura es igual o mayor que 30°C.

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 mínimo 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.

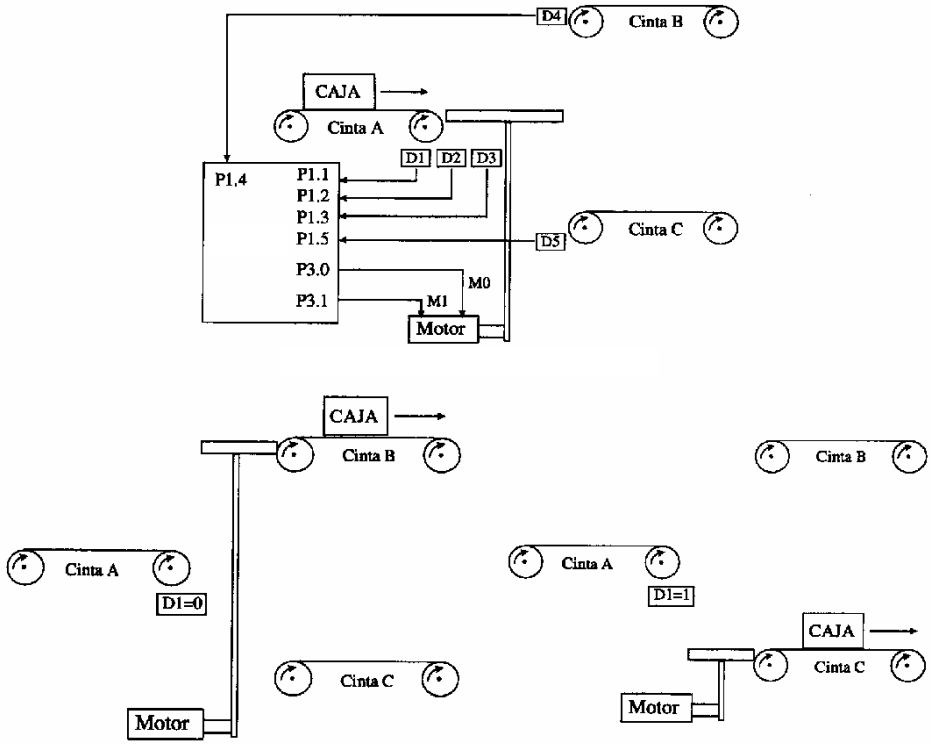
En el caso de que S sea igual a 0 lógico, se enciende el calefactor hasta que se activa el sensor T2, lo cual quiere decir que la temperatura ha superado los 25°C en cuyo caso se desactiva el calefactor. El calefactor permanece apagado hasta que se desactiva T1; entonces se vuelve a encender. En todo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

```
#include <reg51.h>
/*****************************************
 Control del calefactor
*****************************************/
#define OFF 0
#define ON 1
/* entradas */
sbit SEL=P1^3;
sbit ST1=P1^0;
sbit ST2=P1^1;
sbit ST3=P1^2;
/*salidas */
sbit CALEFACTOR=P1^4;
```

```
/* Funcion que devuelve temperatura aproximada */
unsigned char Temperatura(void) {
      if (ST3) return 31;
      if (ST2) return 26;
      if (ST1) return 21;
      return 19;
}
/* estructura para guardar max y min */
struct control {
      unsigned char min;
      unsigned char max;
      };
/* tabla de temperaturas */
struct control tabla_temp[2]={{20,25},{25,30}};

void main(void) {
unsigned char temp;

CALEFACTOR=ON;
while(1) {
  temp=Temperatura();
  if (temp>tabla_temp[SEL].max)     CALEFACTOR=OFF;
  else if (temp<tabla_temp[SEL].min) CALEFACTOR=ON;
  }
}
```

7.9.- 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.

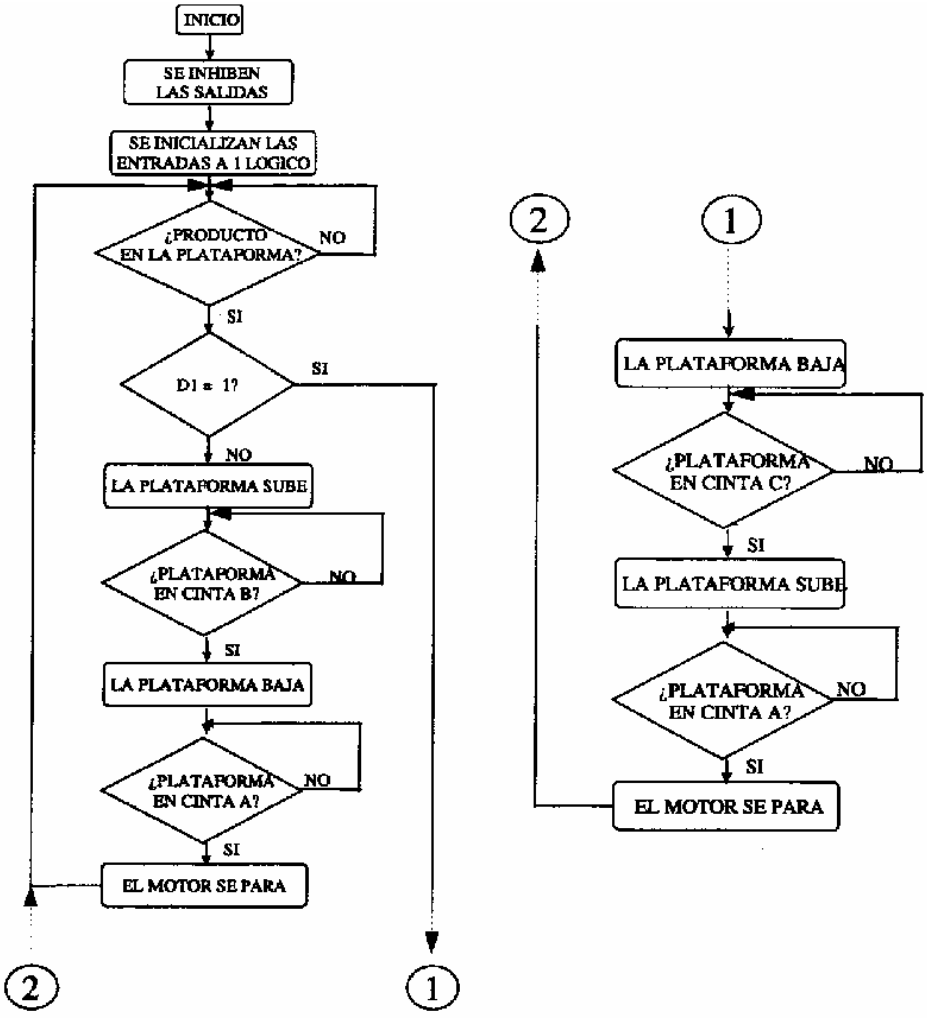


El sistema incorpora los siguientes detectores:

- 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.

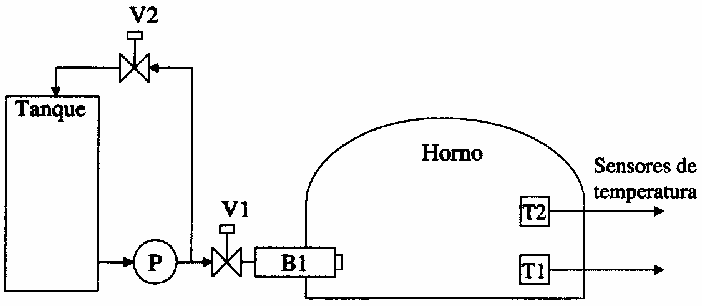
En una primera parte del programa se deben inicializar los puertos con el valor lógico adecuado. Los pines de los puertos que actúan como entradas de datos, han de estar inicializados a 1 lógico y los pines que trabajan como salida han de inicializarse por defecto con el nivel inactivo. En la figura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

```
#include <reg51.h>
/* entradas */
sbit SD1=P1^1;
sbit SD2=P1^2;
sbit SD3=P1^3;
sbit SD4=P1^4;
sbit SD5=P1^5;
/* control del motor */
#define MOTOR P3
#define PARA 0x00
#define SUBE 0x01
#define BAJA 0x02

/*********************************************
 Control de la plataforma elevadora
*********************************************/
void main(void) {
/* inicialización de las entradas y salidas */
      MOTOR=PARA;
      P1=0xFF; /* todas entradas */
while (1) {
      while (!SD3); /* Se espera a que llegue producto a plataforma */
      if (SD1) { /* llevar a cinta C */
            MOTOR=BAJA;
            while (!SD4); /* Se espera plataforma llegue abajo */
            MOTOR=SUBE;
      }
      else { /* llevar a cinta B */
            MOTOR=SUBE;
            while (!SD5); /* Espera plataforma llegue arriba */
            MOTOR=BAJA;
      }
      while(!SD2); /* Se espera plataforma llegue cinta A */
      }
}
```

7.10.- 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.

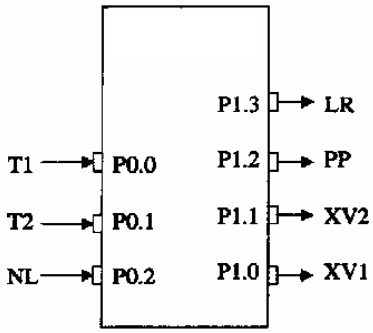
2. ***Actuadores***
   - PP: Accionador de la bomba P. Cuando se activa este actuador, la bomba P se pone en funcionamiento.
   - XV1, XV2: Son los actuadores que abren o cierran las electroválvulas V1 y V2, respectivamente. Cuando se activan, la válvula correspondiente se abre y permite el paso del 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.

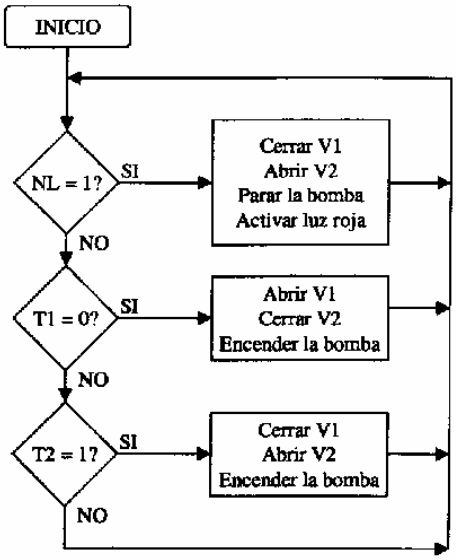
La estrategia de control, que debe implementar el sistema de control del horno, se puede resumir en tres puntos:

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.

```
#include <reg51.h>

/* entradas */
sbit ST1=P0^0;
sbit ST2=P0^1;
sbit SNL=P0^2;
/* salidas */
sbit ALARMA=P1^3;
sbit BOMBA=P1^2;
sbit XV2=P1^1;
sbit XV1=P1^0;
/* defines */
```

```
#define ON 1
#define ABRE 1
#define OFF 0
#define CIERRA 0

/*******************************
 Control del horno
*******************************/
void main(void) {
P1=0x00;              /* salidas a cero */
P0=0xFF;              /* entradas a uno */
while (1) {
  if (SNL) {          /* si disminuye nivel deposito */
     BOMBA=OFF;       /* para la bomba */
     XV1=CIERRA;      /* cierra electrovalvula 1 */
     XV2=ABRE;        /* abre electrovalvula 2 */
     ALARMA=ON;       /* enciende led fuera servicio*/
     }
  else if (ST1) {     /* si baja de 275 grados */
     XV1=ABRE;        /* abre electrovalvula 1 */
     BOMBA=ON;        /* enciende bomba */
     XV2=CIERRA;      /* cierra electrovalvula 2 */
     ALARMA=OFF;      /* apaga led fuera servicio */
     }
  else if (ST2) {     /* si sube de 300 grados */
     XV1=CIERRA;      /* cierra electrovalvula 1 */
     BOMBA=ON;        /* enciende bomba */
     XV2=ABRE;        /* abre electrovavula 2 */
     ALARMA=OFF;      /* apaga led fuera servicio */
     }
  }
}
```