SOLUCIÓN EXAMEN DEL 5 DE SEPTIEMBRE DE 2003

CUESTIONES:

C1. Comenta el mecanismo de control de periféricos en el 8051, tanto desde un punto de vista hardware como software y la relación entre ambos controles. (0.5 puntos).

SOLUCIÓN:

El mecanismo de control de periféricos, como sucede en todos los microprocesadores se realiza mediante la escritura y lectura de una determinada dirección del mapa de direcciones del procesador, estas posiciones son los registros de control/estado del periférico. Cada periférico tiene una dirección o direcciones por las que se comunica con la CPU.

En el caso del 8051 los periféricos internos se encuentran mapeados en la parte alta de la memoria de datos interna y sus direcciones forman parte del espacio de direcciones de la memoria RAM interna, estas direcciones son de 8 bits. En el caso de periféricos que queramos añadir externos, estos se colocan en la memoria RAM externa en direcciones que no se encuentren ocupadas por memoria RAM del sistema u otros periféricos. Estas direcciones son de 16 bits.

La "interfaz hardware software" detalla los valores de estas direcciones y el significado y función de cada registro de control/estado.

C2. Enumera y explica los pasos y tareas que hay que realizar para una correcta modularización de los programas. (0.5 puntos).

SOLUCIÓN:

Los pasos o tareas a realizar son las siguientes:

1. Especificación de las tareas a realizar por el software. Lista de tareas.
2. Agrupación de tareas comunes en colecciones de funciones que formarán módulos.
3. Especificación de listas de funciones en cada módulo dentro del "fichero include" (*.h) propio de cada módulo, que contendrá los prototipos de las funciones agrupadas.
4. Implementación de todas las funciones que pertenezcan a un módulo dentro de un archivo (*.c) distinto para cada módulo.
5. Inclusión de las cabeceras de los módulos empleados (*.h) en el programa principal que podemos verlo como el módulo principal (main.c y main.h).

C3. Comenta el funcionamiento del mecanismo de interrupciones desde el punto de vista software en la programación en C empleando el compilador de KEIL. (Definición de funciones "interrupt"). (0.5 puntos).

SOLUCIÓN:

Para la codificación de funciones de servicio de interrupción en lenguaje C, KEIL tiene una palabra reservada: "interrupt", se emplea de la siguiente manera:

```
void RSI_timer1(void) interrupt n using k {

/* cuerpo de la function */

}
```

Donde, el n es obligatorio e indica el tipo de interrupción, de forma que el resultado de 8*n+3 es el vector de la interrupción autovectorizada disponible en la arquitectura del microcontrolador 8051. La palabra using es

optativa y permite cambiar el banco de registros utilizado, que normalmente es el 0, a la entrada de la interrupción para ahorrase el compilador tener que guardar los registros empleados en la pila.

C4. Enumerar las diferentes topologías de interconexión en RED LOCAL y compararlos indicando sus ventajas e inconvenientes. (0.5 puntos).

SOLUCIÓN:

Las topologías son:

- Estrella. Tiene la ventaja que es fácil de controlar y gestionar, pero la desventaja de que es cara de cablear y del posible fallo del nodo central.
- Anillo. Tiene la ventaja de que es barata de cablear, pero la desventaja de corte en el cable o averia de cualquier nodo.
- Bus. Tiene la ventaja que es fácil de cablear y un fallo en un nodo no influye en la red, pero la desventaja de un corte en el cable y que todos los nodos están conectados al mismo medio.

C5. Comenta razonadamente los parámetros que influyen en el rendimiento de las rutinas codificadas en C frente al empleo de lenguaje ensamblador. ¿En qué circunstancias es más aconsejable el empleo de lenguaje ensamblador? (0.5 puntos).

SOLUCIÓN:

Como hemos visto en los ejemplos de los apuntes del convertidor A/D, el lenguaje C:

- Es más fácil de entender, está más cerca del algoritmo que del computador.
- Es posible realizar las mismas tareas en el hardware que con el ensamblador (algunos dicen que el C es un ensamblador de alto nivel).
- Existen librerías estándar de funciones, por lo que no hay que recodificar algoritmos para realizar operaciones fundamentales como sumar y multiplicar y algunas más.

Pero por el contrario:

- Los tiempos empleados por las funciones C no son predecibles a priori como sucede con el ensamblador.
- El ensamblador generado a partir de un programa C no está optimizado, a veces es necesario realizar su optimización a mano, como hemos hecho en los apuntes.

Estas premisas aconsejan el uso de ensamblador para algoritmos sencillos que requieran una temporización precisa, como por ejemplo la conversión A/D por aproximaciones sucesivas que vimos en el ejemplo de los apuntes.

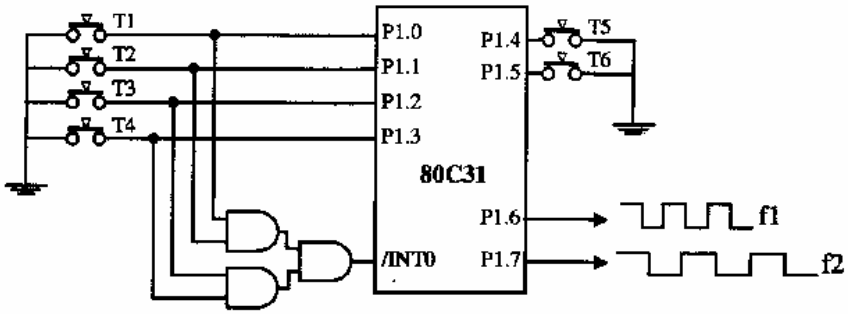
PROBLEMAS:

P1. (2 puntos) Escribir el código en C correspondiente a la función de copia de memoria `memcpy` que copia a `pDst` los `n` bytes que hay a partir de la posición `pSrc`. Suponer que no hay solapamiento entre los rangos fuente y destino:

SOLUCIÓN:

```
void memcpy(unsigned char *pDst, unsigned char *pSrc, unsigned int n)
// n va de 0 a 65535
{
/* bucle hasta que n sea cero y hayamos terminado */
/* tener en cuenta que como todos los parámetros en C siempre se */
/* pasan por copia se pueden modificar sin afectar al programa */
/* que llama a esta función */
/* También se podria haber hecho copiando los parámetros de entrada a */
/* la funcion en variables locales auxiliares */
while (n--) {
      *pDst=*pSrc;
      /* el contenido del puntero fuente se copia al contenido */
      /* del puntero destino */
      pDst++;
      pSrc++;
      /* Se incrementan en una unidad los punteros fuente y destino */
      }
}
```

P2. (3,5 puntos) Traducir el código ensamblador a C escribiendo código únicamente dentro de las cajas.




```
;*************************************
; Generación de una señal periódica
;*************************************
             ORG    0H           ; Vectorización de interrupciones
             LJMP   Inicio
             ORG    03H
             LJMP   RSI_Int0
             ORG    0BH
             LJMP   RSI_Timer0
             ORG    01BH
             LJMP   RSI_Timerl
;*************************************
; Rutina de Inicio
;*************************************
Inicio:             SETB   IT0          ; /INT0 activa por flanco descendente
             SETB   PTO          ; Asigna prioridad alta al Timer0
             SETB   PT1          ;Asigna prioridad alta al Timer 1
             SETB   EX0          ;Habilita interrupción de /INT0
             SETB   ET0          ;Habilita interrupción del Timer 0
             SETB   ET1          ;Habilita interrupción del Timer 1
             SETB   EA           ;Habilita bit de interrupción general
             MOV    TMOD,#22H    ;Timer 0 y 1 en Modo 2, con GATE=0 y C/T=0
             MOV    TL0,#6       ;(256-250) valor inicial para 2kHz
             MOV    TH0,#6       ;carga valor inicial para frec. de 2kHz
             MOV    TL1,#6       ;carga valor inicial para frec. de 2kHz
             MOV    TH1,#6       ;carga valor inicial para frec. de 2kHz
             SETB   TR0          ;Pone marcha el Timer 0
             SETB   TR1          ;Pone marcha el Timer 1
```

```
            LJMP   Principal    ;Ir hacia rutina principal

;************************************
; Rutina de servicio de /INT0
;************************************
RSI_IntO:    JNB    P1.0,Tecla_T1;¿Ha pulsado la tecla T1?
             JNB    P1.l,Tecla_T2       ;¿Ha pulsado la tecla T2?
             JNB    P1.1,Tecla_T3;¿Ha pulsado la tecla T3?
             SJMP   Tecla_T4     ;Tiene que ser la tecla T4
Tecla_T1:    MOV    TH0,#131     ;(256- 125) valor de recarga para 4kHz
             RETI
Tecla_T2:    MOV    TH0,#156     ;(256-100) valor de recarga para 5kHz
             RETI
Tecla_T3:    MOV    TH0,#206     ;(256-50) valor de recarga para 10kHz
             RETI
Tecla_T4:    MOV    TH0,#231     ;(256-25) valor de marga para 20kHz
             RETI

;************************************
; Rutina de servicio del Timer0
;************************************
RSI_Timer0:  CPL    P1.6         ;Complementa P1.6
             RETI                ;Fin de subrutina.
                                 ;El bit TF0 se borra automáticamente
;************************************
; Rutina de servicio del Timer1
;************************************
RSI_Timer1:  CPL    P1.7         ;Complementa P1.7
             RETI                ;Fin de subrutina.
                                 ;El bit TF0 se borra automáticamente
;************************************
; Rutina Principal
;************************************
Principal:   JNB    P1.4,Tecla_T5
             JNB    P1.5,Tecla_T6
             MOV    TH0,#6       ; por defecto 2kHz en Timer 0
             MOV    TH1,#6       ; por defecto 2kHz en Timer 1
             SJMP   Principal    ;Bucle infinito
Tecla_T5:    MOV    TH1,#236     ;(256-20) valor de recarga para 25kHz
             SJMP   Principal
Tecla_T6:    MOV    TH1,#246     ;(256-10) valor de recarga para 50kHz
             SJMP   Principal
```

SOLUCIÓN:

Quedaría algo así:

```
// Generacion de una señal periodica
// Ficheros incluye
// Solo es necesario definir los registros especiales del micro
#include <reg51.h>

// Declaraciones de constantes y puertos de E/S
// Tan solo hay que definir los puertos de E/S de los pulsadores
// Pulsadores de entrada:
sbit bT1=P1^0;
sbit bT2=P1^1;
sbit bT3=P1^2;
sbit bT4=P1^3;
sbit bT5=P1^4;
sbit bT6=P1^5;
// Patillas de salida de la señal:
sbit sF1=P1^6;
sbit sF2=P1^7;

// Variables globales
```

```
// No hay
// Rutina de inicialización
void inicializa (void) {

      TMOD=0x22;
      TL0=6;
      TH0=6;
      TL1=6;
      TH1=6;
      IP|=0x0A;   //los dos bits a la vez: 0000 1010;
      TCON|=0x57; //todos los bits a la vez: 0101 0011
}

// Como no se activan las interrupciones dentro de las rutinas RSI, todas
/* ellas pueden usar el mismo banco de registros sin problemas, pues */
/* no se podrán anidar las interrupciones. */

// Rutina de servicio de /INT0
void RSI_int0(void) interrupt 0 using 0 {
      if (!bT1) TH0=131;
      else if (!bT2) TH0=156;
           else if (!bT3) TH0=206
                else TH0=231;
}

// Rutina de servicio del Timer0
void RSI_Timer0(void) interrupt 1 using 0 {
      sF1=~sF1;
}
// Rutina de servicio del Timer1
void RSI_Timer1(void) interrupt 3 using 0 {
      sF2=~sF2;
}
// programa principal
void main(void) {
      // llamada a la funcion de inicializacion
inicializa();
// bucle sin fin
while(1) {
      if (!bT4) TH1=236;
      else if (!bT5) TH1=246;
            else {
                  TH0=6;
                  TH1=6;
            }
      }
}
```

En el programa anterior el microcontrolador debe generar en la patilla P1.6 una frecuencia de 4KHz, 5kHz, 10kHz y 20kHz, si se pulsan las teclas T1, T2, T3 o T4, respectivamente. Las teclas T5 y T6 afectarán a la frecuencia de la patilla P1.7. Al pulsar T5 se generará en P1.7 una frecuencia de 25kHz, al pulsar T6 la frecuencia será de 50kHz.

Inicialmente, por defecto, en P1.6 y en P1.7 se generará una Frecuencia de 2kHz. Cuando se pulse una tecla se generará la frecuencia correspondiente, y cuando no se pulse ninguna se volverá a generar la frecuencia inicial de 2kHz.

P3. (2 puntos) En una placa de desarrollo basada en el 8051 con 32K de ROM se pretende colocar una zona de RAM de 16Kbytes compartida con los mapas de memoria externa de

datos y de código a partir de la dirección 0x8000. Las señales de control de la memoria RAM son /CS, /RD y /WR y las de la ROM /CE y /OE.

a) Dibujar el mapa de memoria del sistema.

SOLUCIÓN:

La memoria ROM va de 0x0000 a 0x7FFF. La memoria RAM va de 0x8000 a 0xBFFF. Quedan dos huecos en el mapa de memoria externa de datos, uno de 0x0000 a 0x7FFF y otro de 0xC000 a 0xFFFF. Normalmente los periféricos se colocan arriba de la zona de memoria, pero también pueden colocarse debajo, donde más convenga.

El mapa de memoria de código queda con un hueco en la zona de 0xC000 a 0xFFFF, aquí no se pueden colocar periféricos, puesto que estos solo tienen cabida en zona de memoria externa de datos. La zona compartida en el mapa de memoria de código se extiende desde 0x8000 a 0xBFFFF.

b) Diseñar el circuito de control de los chips de RAM y ROM.

A15 y A14 definen cuatro bloques de 16K. El circuito de control tiene que activar el /CE de la ROM en los dos primeros bloques, o sea cuando A15 vale 0, luego /CE se conecta a A15 mediante un inversor. La lectura de la memoria ROM solo se realiza en el mapa de memoria de código, luego la patilla /OE de la ROM se conecta directamente a la señal /PSEN del microcontrolador.

La memoria RAM se accede en el tercer bloque de 16K definido por A15 y A14, luego la señal /CS se conecta a una puerta lógica que ponga un nivel lógico 0 cuando A15 y A14 valen respectivamente uno y cero. Este es el bloque 0x8000-0xBFFF. Por otro lado la señal /WR de la ROM se conecta directamente a la señal /WR del microcontrolador.

Para que la memoria RAM se pueda acceder de forma compartida desde la zona de memoria de datos y desde la zona de memoria de código, se debe activar la patilla /RD de la memoria cuando se activa la señal /RD del micro y también cuando se activa la señal /PSEN del micro, por lo que se emplea una puerta AND entre ambas señales para generar la señal /RD que va a la RAM, como hemos visto en clase.

a) Diseñar el circuito de control para la decodificación de tres dispositivos periféricos con ocho registros de control/estado cada uno. Colocarlos en las direcciones más apropiadas. ¿Cuántas líneas de direcciones llegan a cada dispositivo periférico?

Para realizar la decodificación de tres bloques de memoria de al menos 8 direcciones en el espacio de memoria de datos externa del 8051 es aconsejable emplear un decodificador LS138.

Hay muchas posibilidades de particionar el espacio que nos queda libre, pero la más sencilla sería dividir el espacio 0x0000-0x7FFF en cuatro partes:

A15	A14	A13	A12	Rango	Utilidad
0	0	0	X	0x0000-0x1FFF	Periférico 1
0	0	1	X	0x2000-0x3FFF	Periférico 2
0	1	0	X	0x4000-0x5FFF	Periférico 3
0	1	1	X	0x6000-0x7FFF	Libre

De la tabla se observa que el bit A12 no lo necesitamos para la decodificación, puesto que en cada rango varia, es decir, no nos importa su valor. También observamos que el bit A15 debe estar a 0, pero tampoco varía. De esta forma deducimos que los bits A13 y A14 definen cuatro bloques de un tamaño de 2K, bastante mayor que el rango de 8 registros que necesitamos, pero no importa, porque al menos cumplimos la premisa del enunciado y no sobrescribimos otros registros, se trata de una "decodificación incompleta".

Resumiendo, el LS138 se activaría con la señal A15 a nivel lógico cero y como entrada tendría los bits A14 y A13. Las salidas 0, 1 y 2 se conectarían a las entradas de selección de cada uno de los periféricos. Las señales /RD y /WR se conectarían a las homónimas del microcontrolador.

Puesto que cada periférico dispone de ocho registros, a cada periférico deben llegar las líneas de direcciones necesarias para direccionarlos, o sea, A0, A1 y A2. Es decir, tres líneas de direcciones.