

Nota: El examen resuelto puede tener algún error que se haya deslizado.

Por favor no tomarlo al pie de la letra y preguntar en tutoría las dudas que se tengan sobre la resolución.

CUESTIONES:

C1. Enumera y comenta brevemente los procesos que realiza el enlazador o “linker” como última fase de creación de programas para un sistema empotrado (0.5 puntos).

Solución:

El proceso de enlazado de programas aparece en la página 7 del tema 2. Un montador de enlaces o “linker” es una aplicación cuya misión es la de enlazar adecuadamente los distintos módulos objeto de un programa. Para ello, en esencia, ha de reubicar los diferentes módulos objeto, resolver las referencias cruzadas entre los mismos y añadir al código las funciones de librería necesarias.

En el caso de los sistemas empotrados, generalmente no existe un sistema operativo que realice la tarea de carga del programa, por lo que el programa montador debe incluir un “ubicador” que convierta el programa “exe” en código objeto absoluto, de manera que pueda ser grabado en una memoria ROM y ser ejecutado posteriormente.

C2. Explica que es una “MACRO” en lenguaje C y para que se emplean. (0.5 puntos).

Solución:

Una “macro” en lenguaje C es una función o “rutina” que se inserta “en línea” con el código a través de la función #define del precompilador. En la línea de código C:

```
#define doble(x) ((x)*2)
```

*el precompilador sustituye la expresión doble(cualquier cosa) por ((cualquier cosa)*2) de forma que “cualquier cosa” puede ser cualquier expresión válida en lenguaje C. Esto provoca que cuando se llama a la macro doble(x) no se haga una llamada a función de C con lo que este conlleva de trabajo con la pila, si no que se sustituye el código “en línea” que sea necesario, lo que trae como consecuencia un aumento del código ocupado por el programa.*

C3. Indica las diferencias entre los monitores de sistema y los emuladores en circuito empleados para la depuración de sistemas empotrados. (0.5 puntos).

Solución:

Los monitores de sistema o residentes son soluciones software frente a los emuladores en circuito que son soluciones hardware. Cuando no se tiene acceso físico al sistema y/o las tareas de depuración son simples se suelen emplear monitores residentes que son soluciones más económicas pero que tienen limitaciones, al no permitir el acceso completo al estado del microcontrolador. También interfieren con el programa a testear porque utilizan memoria y recursos hardware como patillas del microcontrolador que ya no se podrán usar para otras tareas.

El emulador en circuito es una solución más potente pero más cara. Con ella es posible visualizar todos los registros internos y el estado del microcontrolador y suelen tener muchas más funciones sin interferir con el programa que se está depurando.

C4. Indica cómo harías para extraer los bits 2,3,4 y 5 de la variable valor y copiarlos a otra variable tmp de tal forma que el resultado estuviese acotado entre 0 y 15. (0.5 puntos).

Solución:

```
tmp=(valor&0x3C)>>2;
```

C5. Explica como funcionan y como se emplean las instrucciones en ensamblador de manipulación de bits del PIC **BCF** y **BSF**. (0.5 puntos)

Solución:

Las instrucciones de manipulación de bits tienen la sintaxis:

```
B(S/C)F puerto,bit
```

Ponen a uno(Set) o a cero(Clear) el bit del puerto indicado respectivamente.

PROBLEMAS:

P1. (2.5 puntos) El siguiente código fuente C para PIC a 4 MHz pretende enviar un byte a través del bus I2C en formato serie. Tiene varios errores, tanto de sintaxis como semánticos. Encontrarlos y corregir el código fuente.

```
#define SCL RC3
#define SDA RC4
#define I2C_DELAY for(delay=0;delay<10;delay++);

unsigned char delay;
void i2c_byte(unsigned char x){
int i;

i=0x40;
do {
SCL=0;
TRISC4 =0; /* PONE SDA SALIDA */
I2C_DELAY;
if (x&i) SDA=0;
else SDA=1;
I2C_DELAY;
SCL=1;
I2C_DELAY;
i>>1;
} while(i!=0);
}
```

Solución:

```
#define SCL RC3
#define SDA RC4
#define I2C_DELAY for(delay=0;delay<10;delay++) ;*

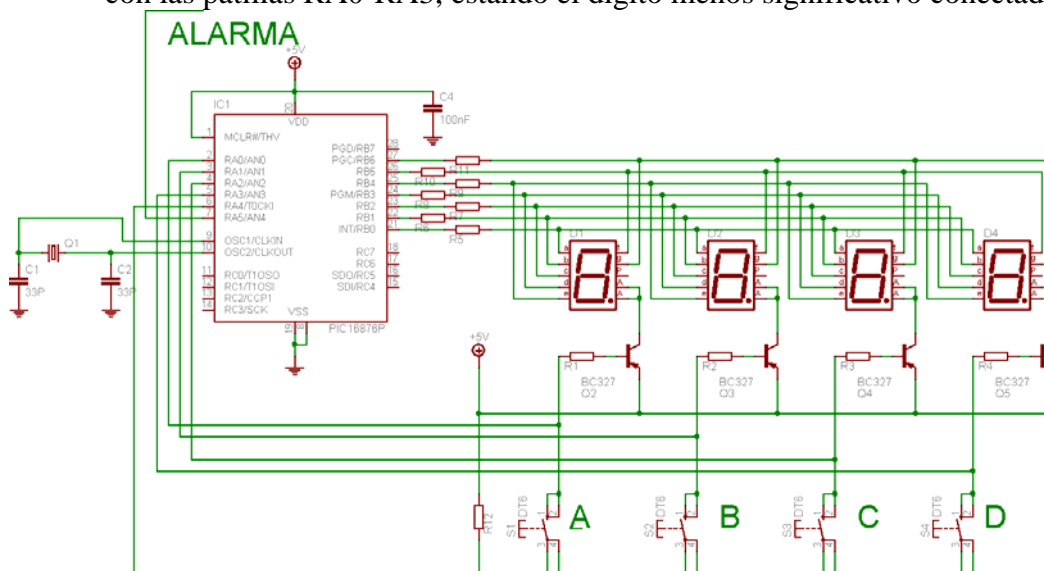
unsigned char delay;
void i2c_byte(unsigned char x){
int i;

i=0x80;
do {
SCL=0;
TRISC4 =0; /* PONE SDA SALIDA */
I2C_DELAY;
if (x&i) SDA=1;
else SDA=0;
I2C_DELAY;
SCL=1;
I2C_DELAY;
i=>>1;
} while(i!=0);
}
```

** este punto y coma no causa ningún problema porque esta línea no se emplea como expresión de lenguaje C sino como sentencia, por tanto no se trata de un error.*

P2. (3 puntos) La empresa CLOCK S.A. ha encargado el desarrollo de un reloj digital con alarma basado en un PIC 16F876 según las siguientes especificaciones:

1. El aparato dispone de cuatro pulsadores: A, B, C y D conectados a RA4.
2. Para la puesta en hora se procede: si se pulsa simultáneamente el pulsador A y el B se incrementan los minutos, si se pulsa A y C se incrementan las horas.
3. Para fijar la alarma se procede: si se pulsa simultáneamente el pulsador D y el B se incrementan los minutos, si se pulsa D y C se incrementan las horas.
4. La alarma está conectada al pin RA5 y se activa con un nivel lógico '1'. No se puede desconectar y sonará durante un minuto.
5. Los displays son de ánodo común, están conectados al puerto B y el control se realiza con las patillas RA0-RA3, estando el dígito menos significativo conectado a RA3.



Se pide:

- a) Escribir el diagrama de flujo y el programa principal del sistema. Se deben emplear interrupciones para que la temporización del tiempo sea exacta. Para simplificar se pueden hacer llamadas a las siguientes cuatro funciones: *leer teclado*, *refrescar el display*, *inicializar interrupciones* y *rutina de atención a interrupción*, las cuales se realizan en los siguientes apartados. Se pueden escribir otras si se consideran necesario siempre que se definan y completen.
- b) Escribir la función o funciones para leer el teclado y refrescar el display.
- c) Escribir la función de interrupción que se encarga de mantener el tiempo y su correspondiente función de inicialización de interrupciones.

Solución:

a) El diagrama de flujo es relativamente simple. Según las pulsaciones de teclas se actualizan las variables que almacenan la hora y la alarma. En este caso, al tener solo cuatro dígitos es más conveniente usar código BCD. A la hora de sacar por pantalla la información no habrá que realizar ninguna conversión. Debido a la longitud del programa se va a tener esto en cuenta en la evaluación de este problema, siendo lo más importante el primer apartado y la rutina de refresco.

La interrupción se encarga de llevar la hora. El programa principal se limita a atender el teclado y a presentar la hora. Sin embargo, como hay pulsaciones dobles de teclas no se puede retornar una sola tecla de la función lee teclado, se podrían retornar dos teclas, pero eso no es realmente necesario, porque solo estamos interesados en cinco situaciones o posibilidades:

| Combinación de Teclas | Código | ¿Que tiene que hacer el programa principal? |
|-----------------------|--------|---|
| A y B | 001 | Incrementar minutos de reloj hasta 60 |
| A y C | 010 | Incrementar horas de reloj hasta 24 |
| D y B | 011 | Incrementar minutos de alarma hasta 60 |
| D y C | 100 | Incrementar horas de alarma hasta 24 |
| D | 101 | Mostrar alarma |
| ninguna | 000 | No se ha pulsado nada, no hago nada |

Cualquier otra combinación de teclas no se emplea, así que lee_teclado_refresca, puede devolver uno de estos códigos en lugar de teclas, para simplificar el problema. Las variables y funciones que vamos a emplear en el programa principal van a ser:

```
unsigned char uni_min_reloj, dec_min_reloj, uni_horas_reloj,
dec_horas_reloj;
unsigned char uni_min_alar, dec_min_alar, uni_horas_alar, dec_horas_alar;

unsigned int msec; //para acumular milisegundos en interrupcion
unsigned char segundos; // para acumular segundos en interrupcion

unsigned char lee_teclado_refresca(unsigned char que_refresca); //refresca
y lee teclado
void init_int(void); // inicializa interrupcion
```

Ahora ya estamos en situación de escribir el programa principal:

```
#define RELOJ 0
#define ALARMA 1 //dos modos de presentacion
#define C_ALARMA RA5 //control de la alarma en patilla RA5

void main(void) {
unsigned char tmp, peticion, ant_minuto;

uni_min_reloj=dec_min_reloj=uni_horas_reloj=dec_horas_reloj;
//pongo a cero la hora
uni_min_alar=dec_min_alar =uni_horas_alar =dec_horas_alar;
//pongo a cero la alarma
C_ALARMA=0;
ant_minuto=0;

init_puertos(); //inicializo puertos
init_int(); //inicializo interrupciones
// a partir de aquí el reloj va solo automaticamente
tmp= RELOJ;
while (1) {
    peticion=lee_teclado_refresca(tmp);
    switch(petición) {
        case 1: {inc_min(RELOJ);break;}
        case 2: {inc_hora(RELOJ);break;}
        case 3: {inc_min(ALARMA);break;}
        case 4: {inc_hora(ALARMA);break;}
        case 5: {tmp=ALARMA; break;}
        default: {tmp= RELOJ;break;}
    }
    // detecta alarma
    if ( (dec_hora_reloj==dec_hora_alr) &&
        (uni_hora_reloj==uni_hora_alr) &&
        (dec_min_reloj ==dec_min_alr) &&
        (uni_min_reloj ==uni_min_alr)) {
        ant_min=uni_min_reloj;
        C_ALARMA=1;
    }
    // desconecta alarma pasado un minuto
    if ((C_ALARMA) && (ant_min!=uni_min_reloj)) C_ALARMA=0;
}
}
```

El diagrama de flujo es simple, se deja como ejercicio. Para simplificar el programa principal hemos creado dos funciones que tienen como misión incrementar las variables en código BCD como se hace en el tema 3. También añadimos la inicialización de los puertos aparte para hacer el código más limpio.

```
void inc_min(unsigned char tipo) {

if (tipo) { //incremento minutos del RELOJ
    uni_min_reloj++;
    if (uni_min_reloj>9) {
        uni_min_reloj=0;
        dec_min_reloj++;
        if (dec_min_reloj>5) dec_min_reloj=0;
    }
}
else { //incremento minutos de la ALARMA
    uni_min_alr++;
}
```

```

        if (uni_min_alr>9) {
            uni_min_alr=0;
            dec_min_alr++;
            if (dec_min_alr>5) dec_min_alr=0;
        }
    }

void inc_horas(unsigned char tipo) {

if (tipo) { //incremento horas del RELOJ
    uni_horas_reloj++;
    if (uni_horas_reloj>9) { //reloj de 24h
        uni_horas_reloj=0;
        dec_horas_reloj++;
    }
    if ((dec_horas_reloj==2) && (uni_horas_reloj>3)) {
        // pone a cero al llegar a 24h
        dec_horas_reloj=0;
        uni_horas_reloj=0;
    }
}
else { //incremento horas de la ALARMA
    uni_horas_alr++;
    if (uni_horas_alr >9) {
        uni_horas_alr =0;
        dec_horas_alr ++;
    }
    if ((dec_horas_alr ==2) && (uni_horas_alr>3)) {
        dec_horas_alr =0;
        uni_horas_alr =0;
    }
}

void init_puertos(void) {
    TRISA=0x10; // 0001 0000 todo salida menos RA4
    TRISB=0x00; // todo salida
}

```

b) *Las funciones para refresco y lectura de teclado son como las del tema 3.*

```

unsigned char lee_teclado_refresca (unsigned char que_refresca) {
    unsigned char tecla;

    tecla=0; //si no se pulsa ninguna
    PORTA&=0xF0; columnas a cero.

    if (que_refresca) PORTB=tabla7seg[dec_horas_alr];
        //refresco reloj o alarma
    else PORTB=tabla7seg[dec_horas_reloj];
    TRISA=0x1E; // 0001 1110 primera columna salida, decenas de horas RA0
    delay(); //mantenemos encendido el digito para que se vea

    if (!RA4) tecla='A'; //se ha pulsado A

    if (que_refresca) PORTB=tabla7seg[uni_horas_alr];
        //refresco reloj o alarma
    else PORTB=tabla7seg[uni_horas_reloj];
    TRISA=0x1D; // 0001 1101 segunda columna salida, unidades de horas RA1
    delay(); //mantenemos encendido el digito

    if (!RA4) {
        if (tecla=='A') tecla=1; //se ha pulsado A y B
    }
}

```

```

        else tecla='B'; //se ha pulsado solo B
    }

    TRISA=0x1B; // 0001 1011 primera columna salida, decenas de minutos RA2
    if (que_refresca) PORTB=tabla7seg[dec_min_alr];
        //refresco reloj o alarma
    else PORTB=tabla7seg[dec_min_reloj];
    TRISA=0x17;
        // 0001 0111 primera columna salida, unidades de minutos RA3
    delay(); //mantenemos encendido el dígito

    if (!RA4) {
        if (tecla=='A') tecla=2; //se ha pulsado A y C
        else tecla='C'; //se ha pulsado solo C
    }

    if ((!RA4) && (tecla=='A')) tecla=2; //se ha pulsado A y C

    if (que_refresca) PORTB=tabla7seg[uni_min_alr]; //refresco reloj o
    alarma
    else PORTB=tabla7seg[uni_min_reloj];
    delay(); //mantenemos encendido el dígito

    if (!RA4) {
        if (tecla=='B') tecla=3; //se ha pulsado D y B
        else if (tecla=='C') tecla=4; //se ha pulsado D y C
        else tecla=5; //se ha pulsado solo D
    }

    TRISA=0x10; Apagamos todo

return tecla;
}

```

c) *La interrupción se ejecuta cada milisegundo y acumula msec, segundos y cuenta...*

```

void init_int(void) {

    T0CS=0;
    PSA=1; //temporiza 1000useg aprox= 256*4
    PS0=1;
    PS1=0;
    PS2=0;
    T0IE=1;
    GIE=1;
}

void interrupt isr(void) {

    if (T0IF) {
        msec++;
        if (msec>=1000) {
            msec=0;
            segundos++;
        }
        if (segundos>59) {
            segundos=0;
            if ((uni_min_reloj==9) && (dec_min_reloj==5))
                inc_horas(RELOJ); //llamada funcion reentrante
            inc_min(RELOJ); //llamada funcion reentrante
        }
    }
    T0IF=0;
}

```

```
}
```

P3. (2 puntos) Realizar la función “void imprime_char (unsigned char valor)” que emplee la función de escritura de un carácter ASCII en la pantalla LCD: lcd_putchar(). La función debe imprimir una variable de 8 bits en formato decimal, o sea debe sacar un máximo de cinco caracteres que pueden ser desde el ‘0’ hasta el ‘9’ en función del número que se le pase a la función. Por ejemplo, si la variable valor contiene el número 45 en decimal, mostraría por pantalla: “00045”.

Solución:

```
void imprime_char (unsigned char valor) {  
  
    lcd_putchar((valor/100)+'0'); //centenas  
    lcd_putchar((valor/10)%10+'0'); //decenas  
    lcd_putchar(valor%10+'0'); //unidades  
}
```