

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1		P2		P3	

Duración 2:00 horas**1.- (3 puntos) Preguntas cortas:****a)** Escriba un trozo de código en ensamblador que multiplique por 24 el contenido de R12.**SOLUCIÓN**

```

mov.w r12, r13      ;R13 = R12 = A
rla.w r12            ;R12 = 2A
add.w r13, r12       ;R12 = 2A + A = 3A
rla.w r12            ;R12 = 6A
rla.w r12            ;R12 = 12A
rla.w r12            ;R12 = 24A

```

b) Escriba un trozo de código que salte a la etiqueta SeCumple si $3 \leq R12 \leq 10$ o $30 \leq R12 \leq 100$. En caso contrario, sigue ejecutándose la siguiente línea.**SOLUCIÓN**

```

Cond1    cmp.w #3, r12      ;R12 >= 3?
         jlo  Sigue        ;...no, hemos terminado
         cmp.w #10+1, r12   ;...sí. R12 < 11?
         jlo  SeCumple     ;...sí, se cumple la condición
Cond2    cmp.w #30, r12     ;...no. R12 >= 30?
         jlo  Sigue        ;...no, seguir con el código
         cmp.w #100+1, r12  ;...sí. R12 < 101?
         jlo  SeCumple     ;...sí, se cumple la condición
Sigue    ...               ;Aquí sigue el código si no se cumple la condición

```

c) Escriba el código necesario para llamar a la función suma64 que suma dos números de 64 bits. El prototipo es long long suma64 (long long a, long long b);

```

z = suma64 (x, y);

```

x, y y z son variables de 64 bits que debe declarar previamente.

SOLUCIÓN

```

.bss x, 8          ;Definir x
.bss y, 8          ;Definir y
.bss z, 8          ;Definir z

mov.w &x+0, r12
mov.w &x+2, r13
mov.w &x+4, r14
mov.w &x+6, r15     ;Pasar a
push.w &y+6
push.w &y+4
push.w &y+2
push.w &y+0         ;Apilar b
call #suma64       ;Hacer la suma
add.w #8, sp       ;Retirar b de la pila
mov.w r12, &z+0
mov.w r13, &z+2
mov.w r14, &z+4
mov.w r15, &z+6     ;Guardar resultado en z

```

d) Describa qué es el perro guardián y para qué se usa.**CRITERIO DE CORRECCIÓN**

Cada apartado 25%

- 2.- (3 puntos) Analice el siguiente código e indique qué hace. Señale si sigue o no el convenio de llamada de C. En caso afirmativo, muestre el prototipo. En caso negativo, enumere los parámetros de entrada/salida, ubicación y tipo. Calcule el número de ciclos de ejecución, así como el tiempo total para $f_{MCLK}=1\text{MHz}$. Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

		Emulación	Ciclos	Código máquina

nss	mov.w #BIT0, r13	;		
nssb	bit.w r13, r12	;		
	rlc.w r14	;		
	rla.w r13	;		
	jnc nssb	;		
	mov.w r14, r12	;		
	ret	;		

SOLUCIÓN

a) Análisis:

```

nss      mov.w #BIT0, r13      ;R13=1 (BIT0 activo)
nssb     bit.w r13, r12        ;Copiar bit activo de R13 en C
          rlc.w r14            ;Bit saliente de R12 entra por la derecha en R14
          rla.w r13            ;Preparar sig. bit (de menos a más significativo)
          jnc nssb             ;Hemos terminado? ...no. Cerrar el bucle
          mov.w r14, r12       ;...sí, salida por R12
          ret

```

La subrutina prepara una máscara en R13 que recorre los bits del registro desde el menos al más significativo. Con esa máscara se copia el bit que corresponda al carry para, a continuación, copiar dicho bit de R12 en R14 introduciéndolo por la derecha. Finalmente se desplaza R13 para preparar el siguiente bit a sondear. Si el único bit a 1 de R13 sale del registro y pasa al C, el proceso se ha terminado. Se sale dejando en R12 la palabra construida, que resulta ser la misma de entrada invirtiendo el orden los bits (el más significativo con el menos).

La entrada y la salida es por R12 y no se alteran los registros R4 a R10, por lo que podemos decir que sigue el convenio de llamada de C. El prototipo sería:

```
int nss (int a);
```

Tiempo de ejecución:

		Emulación	Ciclos

nss	mov.w #BIT0, r13	;mov.w #1, r13	1
nssb	bit.w r13, r12	;	1 \
	rlc.w r14	;addc.w r14, r14	1
	rla.w r13	;add.w r13, r13	1 16 iteraciones
	jnc nssb	;-4	2 /
	mov.w r14, r12	;	1
	ret	;mov.w @sp+,pc	3

Se ha comentado el código para expresar las instrucciones emuladas y el número de ciclos de cada instrucción. El número total de ciclos es $5+5*16=85$. Para una frecuencia de 1MHz, el tiempo total sería de 85µs.

Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos), las de tipo II (las que tienen 1 operando) y las de salto relativo. En éstas sólo hay que codificar la condición y el desplazamiento, que se calcula como el número de palabras desde el origen (instrucción posterior al salto), al destino, es decir, -4, que en complemento a 2 es 111111100. Cada instrucción ocupa una palabra, dando un total de 7 palabras o 14 bytes.

	Emulación	Código máquina
;		OpCd -Rf- DdBDf -Rd-(tipo I)

```

;
;
;-----
nss      mov.w #BIT0, r13      ;mov.w #1, r13      0100 0011 0001 1101 = 431D
nssb     bit.w r13, r12      ;
          rlc.w r14            ;addc.w r14, r14      0110 1110 0000 1110 = 6E0E
          rla.w r13            ;add.w r13, r13      0101 1101 0000 1101 = 5D0D
          jnc nssb             ;-4                  001 010 1111111100 = 2BFC
          mov.w r14, r12      ;
          ret                  ;mov.w @r1+,r0        0100 1110 0000 1100 = 4E0C
                                     0100 0001 0011 0000 = 4130

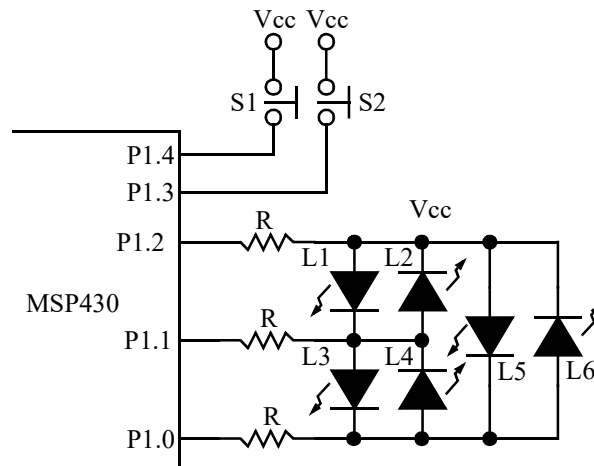
```

CRITERIO DE CORRECCIÓN

- Análisis: 40% (Función 8, convenio 2)
- Tiempo de ejecución: 20%
- Codificación: 40%

- 3.- (4 puntos) Observe el montaje de la figura. Usa la técnica llamada *charlieplexing* para controlar 6 leds con sólo 3 puertos de E/S. Para ello se juega con el estado de los puertos que puede ser 0, 1, o Z (alta impedancia o entrada). Por ejemplo, L2 se enciende haciendo P1.2=0, P1.1=1, P1.0=Z. Inicialmente se enciende L1. Cada vez que se pulsa S1, se apaga el led actual y se enciende el siguiente (el siguiente al L6 es el L1). Cuando se pulsa S2, se apaga el actual y se enciende el anterior (el anterior al L1 y es el L6). Realice un programa en ensamblador del E/S por interrupciones y usando el menor consumo posible.

Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada y ACLK=LFXT en marcha con cristal de 32768Hz.



SOLUCIÓN

Dado que el pulsadores no tienen resistencia externa, hay que usar las internas. En este caso deben ser de *pull-down*. Se configuran sensibles en flanco de subida y se activan las interrupciones.

Los puertos de los leds se configuran en función de cuál se vaya a encender. Para simplificar el proceso, el vector `TabLeds` guarda el estado de los puertos P1.0 a P1.2. En el MSB de cada palabra se guarda el valor del registro de dirección (0: entrada, 1: salida) y en el LSB el valor de salida (en caso de entrada, da igual el valor). Sólo son significativos los bits 0-2 y 8-10. Inicialmente se enciende L1 y se entra en modo de bajo consumo a la espera de las pulsaciones.

La variable `PLedActual` va a contener en todo momento un puntero (dirección) a la máscara de configuración del led actual. Es decir, siempre va a contener una de las direcciones de `TabLeds`.

Dentro de la ISR se incrementa o decrementa la variable `PLedActual` y se corrige si se sale de los límites de la tabla. Finalmente se llama a `SetLeds` que establece el valor de todos los puertos para que se encienda solo el led escogido.

```

;Datos de configuración
SW1BIT .equ BIT4

```

```

SW2BIT      .equ    BIT3
SWPORT      .equ    P1IN
LEDPORT     .equ    P1IN
;Constantes calculadas
SWS         .equ    SW1BIT|SW2BIT
;Variables
            .bss    PLedActual, 2                ;Puntero a led actual
;-----
;-----
main         ;bic.b #SWS, &SWPORT+PDIR           ;Switches como entrada
            bis.b #SWS, &SWPORT+PREN             ;...con resistencia
            bic.b #SWS, &SWPORT+POUT            ;...de pulldown
            bic.b #SWS, &SWPORT+PIES           ;Sensibles en flanco de subida
            bic.b #SWS, &SWPORT+PIFG          ;Borrar banderas de IRQ
            bis.b #SWS, &SWPORT+PIE           ;Habilitar IRQ
            mov.w #TabLeds, &LedActual         ;TabLeds=dir. primer elem. de tabla
            call #SetLeds                      ;Encender L1
            bis.w #GIE|LMP4, sr                ;Habilitar IRQ y dormir
;-----
; void SetLeds (void);                                v1.0
; Enciende el led actual (1-6) y apaga el resto.
; Conserva todo para poder ser llamada desde ISR.
;-----
SetLeds      push.w r12
            mov.w &PLedActual, r12             ;R12=Dir de led actual en tabla
            mov.w @r12, r12                   ;R12=Valor de conf. del led actual

            bic.b #0x07, &LEDPORT+PDIR        ;Poner puertos de leds como entradas
            bic.b #0x07, &LEDPORT+POUT        ;Enmascarar 3 lsb (salidas a 0)
            bis.b r12, &LEDPORT+POUT          ;Actualizar salidas (bits 0-2)
            swpb r12                          ;Acceder a la parte de dirección
            bis.b r12, &LEDPORT+PDIR          ;Activar puertos de salida (bits 0-2)
            pop.w r12
            ret
;-----
; P1ISR                                v1.0
; Subrutina de servicio de interrupción de P1
;-----
P1ISR        add.w &P1IV, pc
            reti                               ;0: No IRQ
            reti                               ;2: P1.0
            reti                               ;4: P1.1
            reti                               ;6: P1.2
            jmp P1_3                           ;8: P1.3 SW2
            jmp P1_4                           ;10: P1.4 SW1
P1_4         ;Subrutina de atención al SW1
            incd.w &PLedActual                 ;Incrementar led actual
            cmp.w #TabLedsFin+2, &PLedActual   ;Se ha pasado?
            jlo P1ISRFin                       ;...no, salir cambiando led
            mov.w #TabLeds, &PLedActual         ;...sí, seguir con L1
            jmp P1ISRFin                       ;Salir cambiando led
P1_3         ;Subrutina de atención al SW2
            decd.w &LedActual                 ;Decrementar led actual
            cmp.w #TabLeds, &PLedActual         ;Se ha pasado?
            jhs P1ISRFin                       ;...sí, salir cambiando led
            mov.w #TabLedsFin, &PLedActual      ;...no, seguir con L6 y cambiar led
P1ISRFin     call #SetLeds                     ;Encender led. No alterar registros
            reti
;-----
; TabLeds
;
; El MSB de cada entrada indica la dirección del puerto (0:entrada)
; El LSB el valor en caso de salida (no importa si entrada).
; Sólo importan los 3 LSB's de cada byte.
;-----
;
; DIR OUT

```

```
TabLeds      .word  0x0604      ;1, 110 10-
              .word  0x0602      ;2, 110 01-
              .word  0x0302      ;3, 011 -10
              .word  0x0301      ;4, 011 -01
              .word  0x0504      ;5, 101 1-0
TabLedsFin   .word  0x0501      ;6, 101 0-1
              .intvecRESET_VECTOR, main
              .intvecPORT1_VECTOR, P1ISR
```

CRITERIO CORRECCIÓN

- Programa principal: 30%
- SetLeds: 30%
- ISR 40%