

Apellidos:.....**SOLUCIÓN**.....

P1	P2	P3

Nombre:.....

Duración 4:00 horas

- 1.- (2425 Conv2 3/10) Analice el siguiente código e indique qué hace. Proponga un código alternativo que haga lo mismo, pero de forma más eficiente. 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 $f_{MCLK}=1MHz$ y compárelo con la alternativa que ha propuesto. Ensamble manualmente el programa dando el código máquina en hexadecimal.

	Emulación	Ciclos	Código máquina
;			
;			
K .equ 0x8000	;		
misterio: mov.w #K, r11	;		
misterioB: rrc.w r12	;		
rrc.w r13	;		
rrc.w r11	;		
rrc.w r13	;		
rrc.w r12	;		
rrc.w r11	;		
jnc misterioB	;		
mov.w r11, r12	;		
ret	;		

SOLUCIÓN

a) Análisis:

```

K .equ 0x8000
misterio: mov.w #K, r11 ;Contador del bucle y registro salida
misterioB: rrc.w r12 ;Desplazar a la derecha R12. LSB de R12 a C
rrc.w r13 ;Desplazar a la derecha R13. LSB de R13 a C
rrc.w r11 ;Desplazar a la derecha R11. Entra C por izquierda
rrc.w r13 ;Desplazar a la derecha R13. LSB de R13 a C
rrc.w r12 ;Desplazar a la derecha R12. LSB de R12 a C
rrc.w r11 ;Desplazar a la derecha R11. Entra C por izquierda
jnc misterioB ;C=0?, Si. Otra vuelta
mov.w r11, r12 ;...no, fin. Salida por R12
ret
    
```

Supongamos que R12=a y R13=b. El bucle saca primero el lsb de a y b por la derecha e introduce el de b en R11 por la izquierda (el de a entra en R13, pero no se hace nada con él). Después se hace lo propio, pero invirtiendo el orden de a y b, con lo que es el bit de a el que se guarda y el de b se introduce en R12 (y no se hace nada con él). El bucle se cierra comprobando si en el último desplazamiento de R11 ha salido un bit a 1, lo que indicaría fin de bucle. Como en R11 se carga el número 0x8000, esto ocurrirá después de 8 iteraciones (16 desplazamientos de cada registro). El resultado es que los bits pares de R11 corresponden con los de b, mientras que los impares son los de a.

b) Código alternativo:

Se propone el siguiente código alternativo. No usa bucles. Sólo 3 operaciones lógicas que tardan 5 ciclos (más 3 del RET).

```

;-----
; uint16_t misterio (uint16_t a, b); v1.0
;
; Baraja a y b y devuelve el resultado. Los bits pares de la salida son de b
; y los impares de a
;-----
    
```

```

misterio:  and.w  #0x5555, r13  ;Borrar bits impares de b
           bic.w  #0x5555, r12  ;Borrar bits pares de a
           bis.w  r13, r12     ;Fusionar
           ret

```

c) Convenio de llamada:

Tal como se muestra en la cabecera de la versión optimizada de `misterio`, sigue el convenio de llamada de C, ya que se pasan 2 enteros de 16 bits sin signo por R12 y R13, la salida se hace por R12 (parámetros a y b mezclados) y no se alteran los registros R4-R10.

d) Cálculo de ciclos y tiempo de ejecución:

		Emulación	Ciclos	Código máquina
K	.equ	0x8000		
misterio	mov.w	#K, r11	2	
misterioB	rrc.w	r12	1	\
	rrc.w	r13	1	
	rrc.w	r11	1	
	rrc.w	r13	1	8 iteraciones
	rrc.w	r12	1	
	rrc.w	r11	1	
	jnc	misterioB	2	/
	mov.w	r11, r12	1	
	ret		3	

El bucle tarda 8 ciclos y da 8 vueltas. El saldo total de ciclos es $2 + 8 \cdot 8 + 4 = 70$. Compárese con los 8 ciclos de la versión optimizada. Ejecutándose a $f_{MCLK} = 1\text{MHz}$, el ciclo de reloj es de $1\mu\text{s}$, lo que nos lleva a un tiempo total de ejecución de $70\mu\text{s}$.

e) 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, -7 para `jnc misterioB`, que en complemento a 2 es `1111111001`. Cada instrucción ocupa una palabra salvo la primera, que tiene que almacenar la constante K, dando un total de 11 palabras o 22 bytes.

		Emulación	Código máquina
			OpCd -Rf- DdBdf -Rd- (tipo I)
			000100 OpC Bdf -Rd- (tipo II)
			001 Cnd Desplazami (salto)
K	.equ	0x8000	
misterio	mov.w	#K, r11	0100 0000 0011 1011 = 403B 8000
misterioB	rrc.w	r12	0001 00 000 000 1100 = 100C
	rrc.w	r13	0001 00 000 000 1101 = 100D
	rrc.w	r11	0001 00 000 000 1011 = 100B
	rrc.w	r13	0001 00 000 000 1101 = 100D
	rrc.w	r12	0001 00 000 000 1100 = 100C
	rrc.w	r11	0001 00 000 000 1011 = 100B
	jnc	misterioB	;(salto -7) 001 010 1111111001 = 2BF9
	mov.w	r11, r12	0100 1011 0000 1100 = 4B0C
	ret		;mov.w @esp+, pc 0100 0001 0011 0000 = 4130

CRITERIO DE CORRECCIÓN

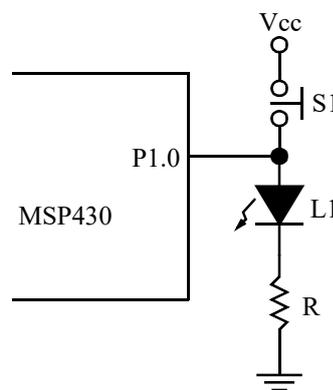
- Análisis: 30%
- Código alternativo: 15%
- Convenio de llamada: 10%
- Tiempo de ejecución: 15%
- Codificación: 30%

2.- (2425 Conv2 3/10) Considere el circuito de la figura. Haga un programa en ensamblador del

MSP430 **basado en multitarea cooperativa** que inicialmente configura el puerto como entrada para poder leer la tecla. Cuando el usuario la pulsa, el led se enciende sin intervención del MSP. El objetivo del programa es alargar el encendido del led para asegurar que el mismo está encendido un segundo más que el tiempo que ha estado pulsada. Para ello, cuando se suelta, se pone el puerto en modo salida para encender el led. Después de un segundo, el puerto se vuelve a poner como entrada, apagándose el led (si el pulsador no está accionado). Para evitar observar el apagado momentaneo del led, ejecute la tarea con un periodo máximo de 10ms.

Notas:

- Nótese que se pide un programa que use multitarea cooperativa, no que lea las teclas por interrupciones ni use el TimerA.
- Minimice el consumo del sistema manteniéndolo todo el tiempo posible en modo de bajo consumo.
- Considere el pulsador libre de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada y LFXT en marcha con cristal de 32768Hz.
- Dispone del módulo `st.asm` y la función `cmp32`.



SOLUCIÓN

Dado que el pulsador no tiene resistencia externa, hay que usar la interna. En este caso debe ser de *pull-down*. El control del led es con lógica positiva (1 enciende, 0 apaga).

Hay que medir tiempos con una resolución de 10ms (más que suficiente para una pulsación de tecla). Eso supone que el proceso de medida debe tener una frecuencia de al menos 100Hz. Se ha escogido 128Hz.

Inicialmente el proceso establece una frecuencia de trabajo de 128Hz con la que escaneará la tecla. Cuando la misma se suelte pulse (es decir, en una lectura estaba pulsada y en la siguiente suelta), se pondrá el puerto en modo salida y se encenderá el led. Acto seguido se sale actualizando el próximo instante de ejecución al instante actual más un segundo. Eso va a hacer que el proceso no se vuelva a ejecutar hasta que pase el segundo en el led está encendido después de la liberación de la tecla. Después del segundo, el led se apaga y se pone el puerto en modo entrada, reiniciando el ciclo.

Una forma sencilla de gestionar todo esto es con una variable `Estado` que pasará por los modos `ENOPUL`, `EPUL` y `EENC`. Se han añadido otros dos estados adicionales para encapsular el código de inicialización (`EINI`) y de error (`EULTIMO`). Nótese cómo, en el estado `EPUL`, cuando se suelta la tecla, se actualiza `teclaPE` para que no ejecute el proceso hasta dentro de un segundo y se pasa al estado `EENC` que gestionará el apagado. Esto permite un control sencillo del tiempo de encendido y evita numerosas ejecuciones del proceso para ver si ha transcurrido el tiempo.

La existencia de 2 estados para saber si la tecla está pulsada o no (`ENOPUL` y `EPUL`) puede parecer caprichosa, pero es necesario para detectar los flancos de la tecla. Nótese que la liberación de la tecla ocurre cuando, en dos lecturas consecutivas de la tecla, la primera da pulsada y la segunda no. Muy distinto de, simplemente, ver si la tecla está libre, ya que, en ese caso, generaríamos un segundo de encendido del led cada vez que se leyera la tecla suelta.

```
.cdecls C,LIST,"msp430ports.h"
```

```

;-----
; Datos de configuración
;-----
; *** Puertos de E/S ***
LEDBIT    .equ    BIT0
LEDPORT   .equ    P1IN

;Tiempo de alargamiento del encendido del led
TALARGA   .equ    1*FST           ;Tiempo añadido al encendido
;Frecuencia de los distintos procesos. En Hz
FTECLA    .equ    128           ;Frecuencia de escaneo de la tecla

; *** Frecuencia del System Timer ***
FTA       .equ    32768         ;Frecuencia del reloj del TA. Hz
FST       .equ    128          ;Frecuencia del SystemTimer. Hz

;-----
; Constantes calculadas
;-----
CCR0       .equ    FTA/FST-1     ;Valor a programar en CCR0 para stIni
PERIODO    .equ    FST/FTECLA    ;Tiempo entre ejecuciones (TICs)

;-----
; Variables
;-----
                .bss    teclaPE, 4      ;Próxima Ejecución del proceso de Parpadeo
                .bss    Estado, 1      ;Estado del proceso

;-----
; Codificación de estados
;-----
EINI       .equ    0*2          ;Estado inicial. Ficticio. Sólo para inicializar
ENOPUL     .equ    1*2          ;Estado tecla no pulsada
EPUL       .equ    2*2          ;Estado tecla pulsada
EENC       .equ    3*2          ;Estado led encendido manualmente
EULTIMO    .equ    EENC         ;Último estado. Sólo a efectos de comprobación

;-----
; main                                                                                               v1.0
;-----
main        ;Inicializar SystemTimer
            mov.w    #CCR0, r12
            call    #stIni

            ;Inicializar procesos
            call    #Inicializa

superbucle call    #AlargaTecla      ;Tarea que alarga la pulsación de la tecla
            bis.w    #LPM3+GIE, sr   ;Entrar en bajo consumo
            jmp     superbucle
            .intvec RESET_VECTOR, main
            .text

;-----
; Inicializa                                                                                         v1.0
;
; Inicializar proceso
;-----
Inicializa clr.w    &teclaPE         ;Ejecutar desde el principio
            clr.w    &teclaPE+2
            mov.b    #EINI, &Estado
            ret

;-----
; Proceso AlargaTecla                                                                              v1.0
;-----
AlargaTecla mov.w    &teclaPE, r12   ;R15:R14=Instante de próxima ejecución
            mov.w    &teclaPE+2, r13
            call    #stComp          ;Comparar con SystemTimer
            tst.w    r12             ;Toca?

```

```

jlo    teclaFin          ;...no. Salir
add.w  #PERIODO, &teclaPE;...sí.Actualizar instante próxima ejecución
adc.w  &teclaPE+2

;Proceso útil de la tarea. Procesar máquina de estados
mov.b  &Estado, r14
cmp.b  #EULTIMO+1, r14
jhs    MdError
add.w  r14, pc
jmp    MdIni             ;Inicialización
jmp    MdNoPul          ;No pulsada
jmp    MdPul            ;Pulsada
jmp    MdEnc            ;Encendido
;jmp   MdError          ;Error

;Estado Error. Desactivar tarea con el puerto como entrada
MdError bic.b  #LEDBIT, &PDIR+LEDPOR;Puerto como entrada
bic.b  #LEDBIT, &PREN+LEDPOR;sin resistencia de pulldown
mov.w  #-1, &teclaPE    ;Próxima ejecución... en el infinito
mov.w  #-1, &teclaPE+1
jmp    teclaFin

;Estado inicialización. Puerto como entrada con resistencia de pulldown
MdIni  bic.b  #LEDBIT, &PDIR+LEDPOR;Puerto como entrada
bis.b  #LEDBIT, &PREN+LEDPOR;Resistencia ...
bic.b  #LEDBIT, &POUT+LEDPOR;... de pulldown
mov.b  #ENOPUL, &Estado;Pasar a esperar pulsación de tecla
;jmp   teclaFin          ;Descomentar para esperar un ciclo
;Estado tecla no pulsada. Esperar a que se pulse
MdNoPul bit.b  #LEDBIT, &PIN+LEDPOR;Tecla pulsada?
jz     teclaFin          ;...no. Salir
mov.b  #EPUL, &Estado  ;Cambiar de estado a esperar liberación
jmp    teclaFin

;Estado tecla pulsada. Esperar a que se suelte
MdPul  bit.b  #LEDBIT, &PIN+LEDPOR;Tecla pulsada?
jnz    teclaFin          ;...sí. Salir
bis.b  #LEDBIT, &PDIR+LEDPOR;...no. Alargar encendido
bis.b  #LEDBIT, &POUT+LEDPOR;Puerto salida. Led encendido
mov.b  #EENC, &Estado  ;Cambiar estado a encendido
add.w  #TALARGA-PERIODO, &teclaPE;Próxima ejecución:fin de tiempo añadido
adc.w  &teclaPE+2
jmp    teclaFin

;Estado led encendido. Apagar el led, poner puerto como entrada
MdEnc  bic.b  #LEDBIT, &PDIR+LEDPOR;Puerto como entrada
bic.b  #LEDBIT, &POUT+LEDPOR;con resistencia de pulldown
MdANoPul mov.b  #ENOPUL, &Estado;Pasar a leer tecla
;jmp   teclaFin

teclaFin  ret

```

CRITERIO DE CORRECCIÓN

- Constantes: 10%
- Principal: 20%
- Proceso: 70%

- 3.- (2425 Conv2 4/10) Se desea hacer un transmisor que usa el código Morse (ver tabla). La duración del punto es $T = 0'25s$. Una raya tiene una duración de $3T$. Entre cada par de símbolos de una misma letra existe una ausencia de señal con duración T . Entre las letras de una misma palabra, la ausencia es de tres puntos ($3T$). Para la separación de palabras transmitidas el tiempo es de tres veces el de la raya ($9T$). En la tabla siguiente se recogen los códigos internacionales para las letras y los dígitos. Existen algunos símbolos más que no vamos a tener en cuenta:

A	.-	G	--.	M	--	S	...	Y	-.--	4-
B	-...	H	N	-.	T	-	Z	---.	5

C	-.-	I	..	O	---	U	..-	0	----	6	-....
D	..	J	.---	P	.-.	V	...-	1	.----	7	----
E	.	K	.-	Q	---	W	.-	2	..---	8	---..
F	...-	L	.-.	R	.-	X	.-.	3	9	----

Codificación en binario de los códigos Morse: se sustituyen los puntos por 0 y las rayas por 1. Sea N la longitud del código. Se añadirán 7-N '0' y un '1' por la izquierda. Por ejemplo, para la letra C, el código es “-.-.” lo que se traduce en “1010” de longitud 4. Se añade un “0001” por la izquierda, quedando “00011010”. De esta forma se codifica la longitud de la secuencia Morse y la propia secuencia en 8 bits. El proceso de decodificación consistirá en desplazar el código a la izquierda hasta que salga el primer 1. 8 menos el número de desplazamientos es la longitud de la secuencia, que ha quedado justificada a la izquierda en el byte desplazado. A continuación se muestran algunos ejemplos más:

```
TabMrsLetra.byte 0b00000101 ; A .-
                 .byte 0b00011000 ; B -...
                 .byte 0b00011010 ; C -.-.
```

La transmisión se hará por el puerto P1.0 que tiene conectado un led con lógica positiva. La función primaria del puerto está asociada a TA0.1. Implemente el módulo `morse.asm` que recoge los siguientes servicios con el mínimo consumo posible:

- `void morseIni (void)`. Inicializa las variables del módulo, configurando el puerto de salida y arrancando el TA0.
- `int morseTx (char c)`. Transmite el carácter `c` cuyo código ASCII se pasa. Si el carácter es un espacio, emite un silencio interpalabra. Si es una letra o un dígito, primero obtiene el código morse asociado y transmite el primer símbolo del mismo. El resto se hace por interrupciones. Devuelve 0 si todo fue bien, 1 si había una transmisión en marcha y 2 si el carácter `c` no es correcto (no es una letra, espacio o dígito). No se distingue entre mayúsculas y minúsculas.
- ISR. Subrutina de servicio de interrupciones del TA0. Transmite los símbolos morse salvo el primero que se hace en la función anterior. Añade los espacios intersímbolos e intercarácter y se desactiva cuando la transmisión ha terminado.

SOLUCIÓN

```
;Datos de configuración
MRSPORT .equ P1IN ;Puerto de salida
MRSBIT .equ BIT0 ;Bit de salida

FACLK .equ 32768 ;Frecuencia de ACLK. En Hz
TPUNTO .equ 250 ;Tiempo base del transmisor morse (en ms)

DIVPRI .equ 8 ;Divisor primario del TA0
DIVSEC .equ 8 ;Divisor secundario del TA0

;Constantes calculadas
FTA .equ FACLK/(DIVPRI*DIVSEC);Frecuencia del TA0. En Hz
NPUNTO .equ FTA*TBASE/1000 ;Número de ciclos del punto
NRAYA .equ 3*NPUNTO ;Número de ciclos de la raya

;Variables
.bss mrsEstado,1 ;Estado del transmisor morse
.bss mrsLong,1 ;Longitud de la trama de interrupciones
.bss mrsCaracter,1 ;Resto del carácter a transmitir

;*** Constantes simbólicas ***
;Estado del transmisor
MRS_LIBRE .equ 0 ;Transmisor morse libre
MRS_TXSIMB .equ 1 ;Transmisor morse transmitiendo símbolo
MRS_TXSLNC .equ 2 ;Transmisor morse tx silencio inter símbolo
```

```

MRS_TXEND .equ 3 ;Transmisor morse terminando
;Códigos de salida del transmisor
MRS_OK .equ 0 ;Transmisión de carácter iniciada
MRS_BUSY .equ 1 ;Transmisor ocupado
MRS_BADCHAR.equ 2 ;Carácter no transmisible

;-----
; void morseIni (void); v1.0
;
; Inicializa el transmisor Morse.
;-----
morseIni ;Inicializar variables
mov.b #MRS_LIBRE, &mrsEstado;Puerto morse libre
;Inicializar puerto
bis.b #MRSBIT, &PDIR+MRSPORT;Puerto morse como salida
bis.b #MRSBIT, &PSEL0+MRSPORT;Puerto morse controlado por Timer A0
;Inicializar CCR
mov.w #OUTMOD_0, &TAOCTL1;Puerto morse no transmite
;TA0 con ACLK/DIVTA modo CONT
mov.w #DIVSEC-1, &TAOEX0;Divisor extra
mov.w #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLRL, &TAOCTL
ret

;-----
; int morseTx (char c); v1.0
;
; Transmite el carácter c en código morse. Después transmite un silencio
; interletra. El primer símbolo se transmite aquí mismo. El resto se encola y se
; hace por interrupciones. Si el carácter es una espacio, transmite un silencio
; interpalabra. Devuelve un código de estado:
; 0: Ok
; 1: Transmisor ocupado
; 2: Carácter no soportado
;-----
morseTx mov.b r12, r13 ;Liberar R12
mov.w #MRS_BUSY, r12 ;Código de salida: transmisor ocupado
cmp.b #MRS_LIBRE, &mrsEstado;Transmisor libre?
jne morseTxFin ;...no. Salir con error
mov.w #MRS_BADCHAR, r12;...sí. Código de salida: carácter no soportado
;Traducir código ASCII a Morse
cmp.b #' ', r13 ;Es espacio? (se usa para separar palabras)
jeq morseTxPal ;...sí, Tx silencio interpalabra
cmp.b #'z'+1, r13 ;Es minúscula?
jge morseTxFin ;...no, salir sin transmitir
cmp.b #'a', r13 ;Es minúscula?
jlo morseNoMin ;...no, ver otras posibilidades
mov.b TabMrsLetra-'a' (r3), r13;Convertir a morse
jmp morseLong ;Es mayúscua?
morseNoMin cmp.b #'Z'+1, r13 ;Es mayúscula?
jge morseTxFin ;...no, salir sin transmitir
cmp.b #'A', r13 ;Es mayúscula?
jlo morseNoMay ;...no, ver otras posibilidades
morseLetra mov.b TabMrsLetra-'A' (r3), r13;Convertir a morse
jmp morseLong
morseNoMay cmp.b #'9'+1, r13 ;Es dígito?
jge morseTxFin ;...no, salir sin transmitir
cmp.b #'0', r13 ;Es dígito?
jlo morseTxFin ;...no, salir sin transmitir
mov.b TabMrsDigit-'0' (r3), r13;Convertir a morse
;Encontrar longitud de la cadena morse
morseLong mov.b #8, r12 ;R12=Longitud del código morse
morseLBuc dec.b r12 ;Descontar longitud
rlc.b r13 ;Desplazar código
jnc morseLBuc ;Ha salido un 1? No, seguir
;Transmitir primer símbolo y encolar el resto
mov.w #NPUNTO, r14 ;R14=Longitud del punto (por defecto)
dec.b r12 ;Actualizar longitud
rlc.b r13 ;Desplazar código

```

```

        jnc     morsePunto      ;Es un punto? Seguir
        mov.w  #NRAYA, r14      ;R14=Longitud de la raya
morsePunto mov.w  #OUTMOD_0|OUT, &TA0CCTL1;Empezar transmisión
        mov.w  #CAP|CM_1|CCIS_2, &TA0CCTL1;Capturar TA0
        mov.w  #CAP|CM_2|CCIS_2, &TA0CCTL1;TA0CCR0=Ahora
        add.w  r14, &TA0CCR1    ;Programar fin del símbolo
        mov.w  #OUTMOD_4|CCIE, &TA0CCTL1;En la IRQ, apagar símbolo (silencio)
        mov.w  r12, &mrsLong    ;Guardar cuántos símbolos quedan
        mov.w  r13, &mrsCharacter;Guardar el resto del carácter
        mov.b  #MRS_TXSIMB, &mrsEstado;Cambiar estado a transmitiendo símbolo
        mov.w  #MRS_OK, r12     ;Salida Ok
        jmp    morseTxFin      ;Salir
        ;Transmitir un espacio interpalabra (con la aparición de un espacio)
morseTxPal mov.w  #OUTMOD_0, &TA0CCTL1;Línea en silencio
        mov.w  #CAP|CM_1|CCIS_2, &TA0CCTL1;Capturar TA0
        mov.w  #CAP|CM_2|CCIS_2, &TA0CCTL1;TA0CCR0=Ahora
        add.w  #3*NRAYA, &TA0CCR1;Programar fin del silencio interpalabra
        mov.w  #OUTMOD_5|CCIE, &TA0CCTL1;Apagar línea después. Ya por IRQ
        mov.b  #MRS_FIN, &mrsEstado;Cambiar estado a Fin
        clr.w  r12             ;Salida Ok
morseTxFin ret

```

```

;-----
; TA00ISR                                                    v1.0
;
; Subrutina de gestión de interrupciones del TA0CCR0. Gestiona la transmisión
; de los caracteres morse, incluyendo los silencios intersímbolos e intercarácter.
;-----

```

```

TA01ISR   bic.b  #CCIFG, &TA0CCTL1;Limpiar IRQ
          cmp.b  #MRS_FIN, &mrsEstado;Silencio generado?
          jeq   TA00ISREnd    ;...sí. Terminar
          cmp.b  #MRS_SLNC, &mrsEstado;Trama terminada?
          jeq   TA01ISRSil    ;...sí. Generar silencio intercarácter
          bit.b  #MRSBIT, &PIN+MRSPORT;...no. Transmitiendo trama. Salida a 1?
          jnz   TA01ISRSimb   ;...sí. Transmitir símbolo
          add.w  #NPUNTO, &TA0CCR1;...no. Transmitir silencio.Duración del punto
          jmp   TA01ISRFin     ;Salir
TA01ISRSimb add.w  #NPUNTO, &TA0CCR1;Se transmitirá al menos un punto
          rlc.b  &mrsCharacter  ;Siguiente elemento de la trama
          jnc   TA01ISRPont    ;Es 0? Sí, es un punto
          add.w  #NRAYA-NPUNTO, &TA0CCR1;...no. Añadir tiempo de raya
TA01ISRPont dec.b  &mrsLong     ;Elemento trasmitido. Quedan?
          jnz   TA01ISRFin     ;...sí, salir
          mov.b  #MRS_SLNC, &mrsEstado;...no. Añadir el silencio entre letras
          jmp   TA01ISRFin     ;Salir
TA01ISRSil add.w  #3*NPUNTO, &TA0CCR0;3 tiempos de punto en silencio
          mov.w  #OUTMOD_5|CCIE, &TA0CCTL0;Dejar línea apagada después de tiempo
          mov.b  #MRS_FIN, &mrsEstado;Trama terminada
          jmp   TA01ISRFin     ;Salir
TA01ISREnd mov.w  #OUTMOD_0, &TA0CCTL0;...sí. Salida a 0. IRQ desactivada
          mov.b  #MRS_LIBRE, &mrsEstado;Cambiar estado a libre
TA01ISRFin reti

```

```

;-----
; TabMrsDigit                                                    v1.0
;
; Tabla de codificación de los dígitos en Morse.
; Los puntos son 0 y las rayas 1. Dado que son códigos de longitud variable, se
; antepone 01 al código para indicar dónde empieza el mismo. Se completa con 0
; por la izquierda hasta 8 bits.
;-----

```

```

TabMrsDigit.byte 0b00111111 ; 0 ----
                .byte 0b00101111 ; 1 .----
                .byte 0b00100111 ; 2 ..---
                .byte 0b00100011 ; 3 ...--
                .byte 0b00100001 ; 4 ....-
                .byte 0b00100000 ; 5 .....
                .byte 0b00110000 ; 6 -....
                .byte 0b00111000 ; 7 --...

```

```

        .byte 0b00111100    ; 8 ----.
        .byte 0b00111110    ; 9 ----.
;-----
; TabMrsLetra                                     v1.0
;
; Tabla de codificación de las letras en Morse
; Los puntos son 0 y las rayas 1. Dado que son códigos de longitud variable, se
; antepone 01 al código para indicar dónde empieza el mismo. Se completa con 0
; por la izquierda hasta 8 bits.
;-----
TabMrsLetra.byte 0b00000101    ; A .-
        .byte 0b00011000    ; B -...
        .byte 0b00011010    ; C -.-.
        .byte 0b00001100    ; D -..
        .byte 0b00000010    ; E .
        .byte 0b00010010    ; F ..-.
        .byte 0b00001110    ; G --.
        .byte 0b00010000    ; H ....
        .byte 0b00000100    ; I ..
        .byte 0b00010111    ; J .---
        .byte 0b00001101    ; K -.-
        .byte 0b00010100    ; L .-..
        .byte 0b00000111    ; M --
        .byte 0b00000110    ; N -.
        .byte 0b00001111    ; O ---
        .byte 0b00010110    ; P .--.
        .byte 0b00011101    ; Q ---.
        .byte 0b00001010    ; R .-.
        .byte 0b00001000    ; S ...
        .byte 0b00000011    ; T -
        .byte 0b00001001    ; U ..-
        .byte 0b00010001    ; V ...-
        .byte 0b00001011    ; W .--
        .byte 0b00011001    ; X -.-.
        .byte 0b00011011    ; Y -.--
        .byte 0b00011100    ; Z --..

```

CRITERIO DE CORRECCIÓN

- Definiciones e inicialización: 10%
- morseIni: 20%
- morseTx: 40%
- ISR: 30%