SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

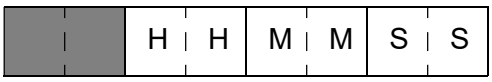
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

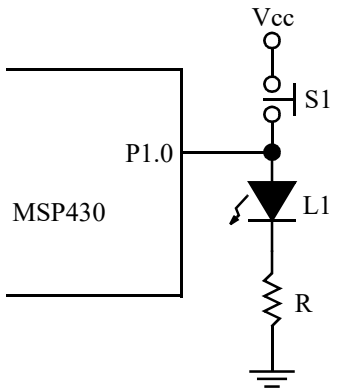
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

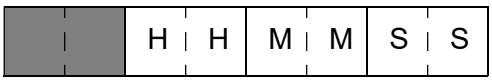
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

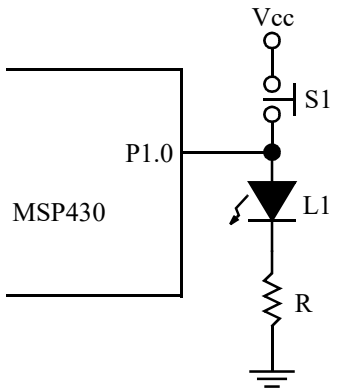
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

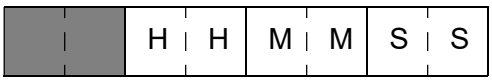
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

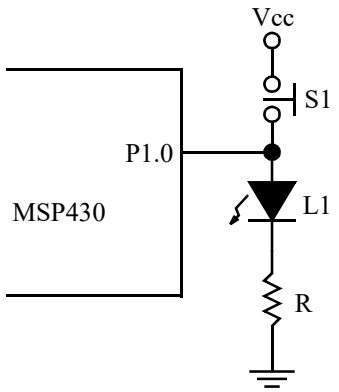
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

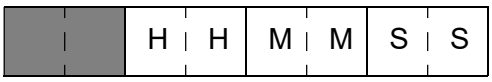
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

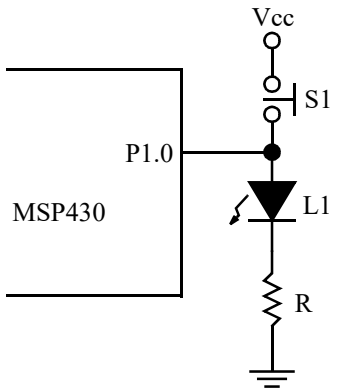
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

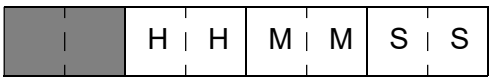
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

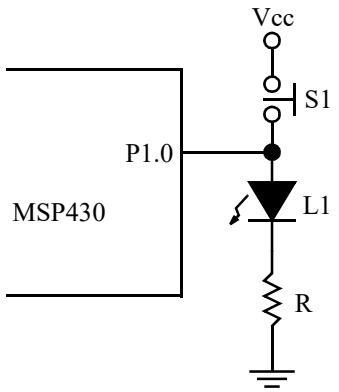
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

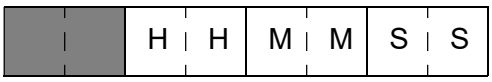
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

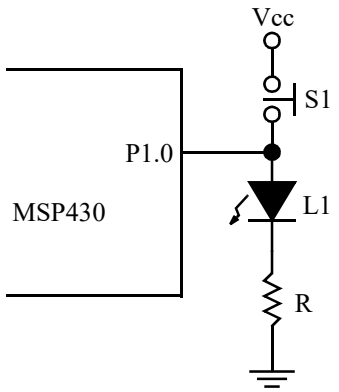
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

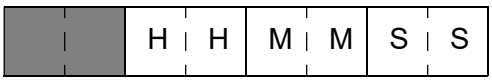
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

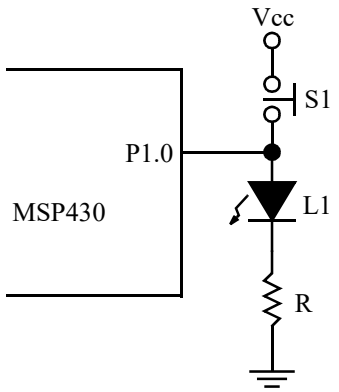
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

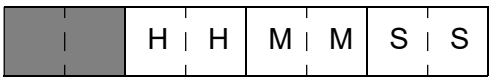
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

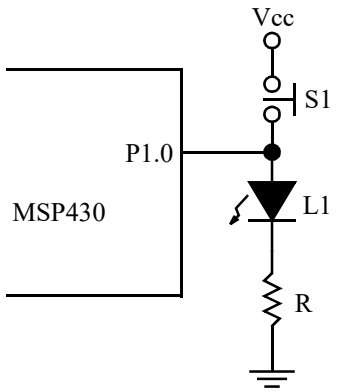
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

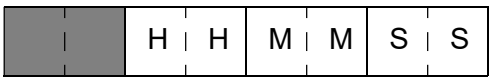
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

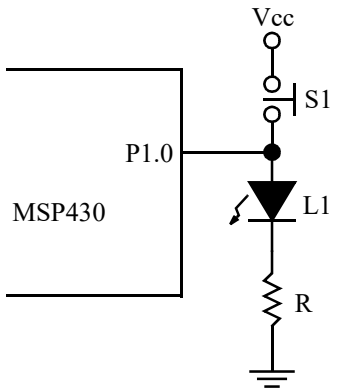
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%

SISTEMAS BASADOS EN MICROPROCESADOR **Segunda convocatoria 2023/2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Sea el siguiente código:

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;_______________|    |_____________________________
            mov.w  #8, r15      ;_______________|    |_____________________________
bucle       rla.b  r12          ;_______________|    |_____________________________
            adc.w  r14          ;_______________|    |_____________________________
            dec.w  r15          ;_______________|    |_____________________________
            jnz    bucle        ;_______________|    |_____________________________
            mov.w  r14, r12     ;_______________|    |_____________________________
            ret                 ;_______________|    |_____________________________
```

a) Analícelo en 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.

b) Calcule el tiempo total de ejecución para F_{MCLK}=1MHz.

c) Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código?

SOLUCIÓN

a) Análisis:

El código inicializa R14=0 y R15=8 antes de entrar en un bucle. Del mismo se sale cuando R15=0 después de decrementarlo. Se trata, por tanto, de un bucle `for` que da 8 vueltas. Dentro del mismo se desplaza el byte bajo de R12 un bit a la izquierda. El bit de salida se guarda en el carry, que se suma a R14. Es decir, el código calcula el número de bits que están a 1 en el byte bajo de R12. Se trata de una subrutina que sigue el convenio de llamada de C, puesto que la entrada y la salida se hace por R12 y se conservan los registros R4-R10. El prototipo sería:

```
int QuienSabe (unsigned char b);
```

b) Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14  1
            mov.w  #8, r15      ;               1
bucle       rla.b  r12          ;add.b r12, r12 1     \
            adc.w  r14          ;add.w #0, r14  1     | 8 veces
            dec.w  r15          ;sub.w #1, r15  1     |
            jnz    bucle        ;               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 bucle tarda 5 ciclos y se ejecuta 8 veces, lo que da un total de 40 ciclos. Las instrucciones externas al bucle dan un total de 6 ciclos más, dando un resultado de 46 ciclos. Para una frecuencia de 1MHz, el tiempo total sería $46/10^6$=46μs.

c) Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos) 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 1111111100. Cada instrucción ocupa una palabra, dando un total de 8 palabras o 16 bytes.

```
;                               Emulación           Código máquina
```

```
;                                                       OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                       -OpCd- Desplazami(salto)
;-------------------------------------------------------------------------------
QuienSabe   clr.w  r14          ;mov.w #0, r14      0100 0011  0000 1110 = 430E
            mov.w  #8, r15      ;                   0100 0010  0011 1111 = 423F
bucle       rla.b  r12          ;add.b r12, r12     0101 1100  0100 1100 = 5C4C
            adc.w  r14          ;addc.w #0, r14     0110 0011  0000 1110 = 630E
            dec.w  r15          ;sub.w #1, r15      1000 0011  0001 1111 = 831F
            jnz    bucle        ;                   001000 1111111100    = 23FC
            mov.w  r14, r12     ;                   0100 1110  0000 1100 = 4E0C
            ret                 ;mov.w @r1+,r0      0100 0001  0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

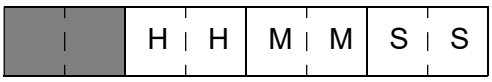
a) 30%

b) 30%

c) 40%

2.- [3 puntos] Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```



`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):

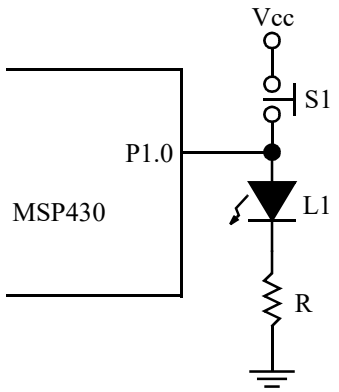
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13      ;Dejar libre R12 para la salida
            clr.w  r12           ;Por defecto, salir con Ok
            clrc                 ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)    ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeMM     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)        ;SS=0
            dadd.b #1, 1(r12)    ;Sumar 1 a MM
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
```

```
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- [4 puntos] Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que mantenga el led encendido 1 segundo más desde la liberación del pulsador. Una vez que pase ese tiempo, se volverá al estado inicial. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Se configurará el P1.0 como entrada con resistencia de *pull-down* para poder detectar la pulsación de la tecla y mantener el led apagado. Tanto la lectura de la tecla como la activación automática del led serán con lógica positiva.

Todo el control se hará con el CCR0 del TA0 (TA0.0) con el procesador dormido en LPM3 ya que se requiere ACLK en marcha, tanto la lectura de la tecla, como la activación del led con un tiempo preciso. Por tanto, el puerto P1.0 se activará en su función primaria.

Inicialmente se configurará el TA0.0 con entrada ACLK a 32768Hz en modo captura, sensible en flanco de bajada para detectar la liberación de la tecla. En la ISR se pondrá el puerto como salida, se pondrá la misma a 1 inmediatamente (OUTMOD 0 con OUT=1) y se configurará el TA0.0 en modo de comparación con modo de salida RESET (OUTMOD 5). El tiempo de 1 segundo se consigue sumando 32768 al valor capturado en el CCR0.

Cuando pase el segundo, el timer apagará automáticamente el led y en la ISR se hará lo necesario para que la tecla pueda ser leída en un nuevo ciclo: poner el puerto en modo entrada y el TA0.0 en modo captura sensible en flanco de bajada.

Si se desea bajar el consumo, se podría bajar la frecuencia del TA0 lo que se quisiera si la precisión de la medida no es crucial. La resolución a 32768Hz es de 1/32768 = 30μs. Bajando la velocidad al mínimo (divisor principal a /8 y secundario a /8), el TA0 iría a 512 Hz y el error sería inferior a 2ms.

Nótese que el led llega a apagarse cuando el usuario suelta la tecla. Sin embargo, se enciende poco tiempo después como consecuencia de la entrada de la interrupción. El tiempo estimado es el de la salida del modo de bajo consumo (6μs), 6 ciclos de latencia de la interrupción más unos 20 ciclos del código que detecta el modo de comparación y activa la salida. Unos 32μs en total suponiendo que f_{MLCK}=1 MHz. Imperceptible a la vista. Si este pequeño apagón es

intolerable, la solución es usar dos puertos en vez de uno: uno para la tecla y otro para controlar el led.

```
;Datos de configuración
PUERTO      .equ   BIT0
DIVPRI      .equ   0              ;Divisor principal:0(1), 1(2), 2(4) o 3(8)
DIVSEC      .equ   1              ;Divisor secundario:1 a 8
FACLK       .equ   32768
;Datos calculados
DIV1        .equ   $pow(2,DIVPRI)
FTA         .equ   FACLK/(DIV1*DIVSEC)


main        ;PUERTO entrada, pulldown
            ;bic.b #PUERTO, &P1DIR;Entrada
            bis.b  #PUERTO, &P1REN;Habilitar resistencia
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Puerto controlado por TA0.0
            ;TA0.0 captura síncrona, entrada A, flanco bajada, IRQ
            ;También se configuran los parámetros para cuando se pase a modo
            ;comparación: salida a 1 (OUTMOD_0 y OUT)
            mov.w  #CAP|SCS|CCIS_0|CM_2|OUTMOD_0|OUT|CCIE, &TA0CCTL0
            ;TA0 ACLK/(DIV1*DIV2), Continuo
            mov.w  #DIVSEC-1, &TA0EX0
            mov.w  #TASSEL__TACLK|(DIVPRI<<6)|MC__CONTINUOUS|TACLR, &TA0CTL

            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                              v1.0
; Subrutina de servicio de la interrupción del TA0 CCR0
;-------------------------------------------------------------------------------
TA00ISR     ;Hay dos fuentes de interrupción (ambas se limpian automáticmaente):
            ;- Tecla liberada (evento de captura en flanco de bajada)
            ;- Fin de encendido automático (evento de comparación)
            bit.w  #CAP, &TA0CCTL0;Estamos en modo captura?
            jeq    TA00Cap        ;...sí, Encendido automático
                                  ;...no. Fin de encendido

            ;Evento comparación->Puerto como entrada. Pasar captura flanco bajada
TA00Comp    bic.b  #PUERTO, &P1DIR;Poner puerto en modo entrada
            bis.w  #CAP, &TA0CCTL0;TA0.0 captura
            jmp    TA00Fin        ;Salir

            ;Evento captura->Puerto como salida. Encender led. Programar apagado 1s
TA00Cap     bic.w  #CAP|OUTMOD_7, &TA0CCTL0;Pasar a COMPARACION y OUTMOD_0
            bis.b  #PUERTO, &P1DIR;Poner puerto en modo salida
            add.w  #FTA, &TA0CCR0;Sumar tiempo de encendido automático
            bis.w  #OUTMOD_5, &TA0CCTL0;Seguir en comparación, pero en OUTMOD_5 (0)

TA00Fin     reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

Configuración: 40%

ISR: 60%