SISTEMAS BASADOS EN MICROPROCESADOR **Primer control 2021-2022**

Apellidos:.....SOLUCIÓN.....

Nombre:.....Puesto.....

P1	P2	P3

Duración 2:00 horas

1.- **[3 puntos]** Escriba en ensamblador del MSP430 las instrucciones necesarias para:

a) Dividir por 2 el número de 32 bits con signo que hay en R13:R12.

b) Dejar en R12 el valor absoluto de R12.

c) Sea R14 una máscara binaria. Por cada bit de R14 que sea 1, cambie el bit de R12 por el correspondiente de R13. Si el bit de R14 es 0, el correspondiente bit de R12 queda inalterado. Ejemplo con palabras de 4 bits: Si R12=$a_3a_2a_1a_0$, R13=$b_3b_2b_1b_0$ y R14=1010, en R12 quedaría $b_3a_2b_1a_0$.

d) Sumar 1 al elemento `R12`-ésimo de un vector de `long` que se encuentra a partir de `R13`.

SOLUCIÓN

a) Para dividir por dos, hay que hacer un desplazamiento a la derecha. Como es un número de 32 bits, se empezará por la MSW (R13). Y como es un número con signo, se usará la instrucción RRA para el primer desplazamiento y RRC para los demás.

```
        rra.w  r13          ;Desplazar MSW a la derecha con signo
        rrc.w  r12          ;Desplazar LSW a la derecha rellenando con C
```

b) Para calcular el valor absoluto de un número, se comprobará el signo. Si es positivo, se ha terminado. En caso contrario, se calcula el opuesto del mismo; en este caso se ha hecho el complemento a 2.

```
        tst.w  r12          ;Número positivo?
        jge    fin          ;...sí, salir
        inv.w  r12          ;...no. R12 = Ca1 (R12)
        inc.w  r12          ;R12 = Ca1 (R12) + 1 = Ca2 (R12) = -R12
fin     ...                 ;aquí sigue el código
```

c)El proceso se puede hacer de muchas formas. La más sencilla por número de instrucciones y recursos necesarios (registros adicionales) es la siguiente:

```
        ;En los comentarios se ha supuesto R12=a3a2a1a0, R13=b3b2b1b0 y R14=1010
        xor.w  r13, r12     ;R12=<a3⊕b3><a2⊕b2><a1⊕b1><a0⊕b0>
        bic.w  r14, r12     ;R12=<0><a2⊕b2><0><a0⊕b0>
        xor.w  r13, r12     ;R12=<0⊕b3><a2⊕b2⊕b2><0⊕b1><a0⊕b0⊕b0>=b3a2b1a0
```

d) Para acceder a un elemento de un vector dada su dirección inicial y su índice, se calculará el desplazamiento multiplicando el índice por el tamaño del elemento (4) y se calculará la posición en memoria sumando la dirección inicial. Ya sólo queda hacer la operación en cuestión. Al tratarse de un elemento de 32 bits, se necesitarán 22 instrucciones para tener el cuenta el posible desboradamiento.

```
        rla.w  R12          ;R12=2*i
        rla.w  R12          ;R12=4*i=offet del elemento i-ésimo
        add.w  R13, R12     ;R12=Dir. del elemento i-ésimo
        inc.w  0(R12)       ;Incrementar elemento del vector
        adc.w  2(R12)       ;...sumando el carry a la palabra alta
```

CRITERIO DE CORRECCIÓN

Cada apartado 25%

2.- **[4 puntos]** Se desea realizar una subrutina en ensamblador del MSP430 que seguirá el convenio de llamada tipo C y que atenderá al siguiente prototipo:

```
void DesUnivDer (void *p, int n, int modo);
```

donde `p` es un puntero a un número en memoria en formato *Little Endian* y `n` es el número de bytes que ocupa. En todos los casos el contenido del número es desplazado un bit a la

derecha y el *lsb* se guarda en el carry. El *msb* se actualiza según los dos bits menos significativos del parámetro `modo`. El resultado queda en el mismo sitio.

modo	Tipo de desplazamiento	Nomenclatura MSP	msb ←
0	LSR. Desplazamiento lógico a la derecha.	-	0
1	ASR. Desplazamiento aritmético a la derecha.	RRA	msb
2	ROR. Rotación a la derecha.	-	lsb
3	RRC. Rotación a la derecha por el carry	RRC	carry

a) Explique el algoritmo que usará para resolver el problema.
b) Escriba la subrutina en ensamblador.

SOLUCIÓN

Para desplazar varias palabras a la derecha hay que empezar siempre por el MSB. Como el MSP430 utiliza un convenio *Little Endian*, esto implica empezar por el último byte (ubicado en p+n-1). Otro detalle a tener en cuenta es que en cada iteración del bucle se desplaza un byte con RRC para incorporar el C de la iteración anterior y dejar el C saliente para la próxima. El problema es que hay que hacer aritmética con el puntero y con el contador de bytes, lo que supone perder el C. Para evitarlo, se guarda el SR justo después del desplazamiento en R15 y se recupera justo antes del siguiente. El primer desplazamiento es especial porque no hay SR guardado y porque el C es función del parámetro modo.

Los distintos desplazamientos se diferencian sólo en qué valor se carga en el carry. La solución habitual va preguntando por los distintos valores que puede tomar el parámetro modo y va saltando en caso de no serlo:

```
            and.w  #BIT1|BIT0, r14;Ignorar todo lo que no sean los 2 lsb de modo
            jnz    VerModo1       ;No es 0. Ver si es el 1
Modo0       clrc                  ;Modo=0. Hacer C=0
            jmp    Modo3
VerModo1    cmp.w  #1, r14        ;Es 1?
            jnz    VerModo2       ;...no es 1. Ver si es el 2
Modo1       bit.b  #BIT7, 0(r12) ;Copiar msb en C
            jmp    Modo3
VerModo2    cmp.w  2, r14         ;Es 2?
            jnz    Modo3          ;...no es 2. Es el 3
Modo2       bit.b  #BIT0, 0(r15) ;Copiar lsb en C
Modo3       ...                   ;En todos los casos se sigue por aquí
```

Una alternativa más interesante, sobre todo cuando el número de opciones es alta, es usar una tabla de saltos. El código queda más limpio, es más fácil de seguir y tarda el mismo tiempo en llegar al trozo específico independientemente de cuál sea el modo:

```
            and.w  #BIT1|BIT0, r14;Ignorar todo lo que no sean los 2 lsb de modo
            rla.w  r14            ;Modo por 2 para indexar tabla de saltos
            add.w  r14, pc        ;Saltar al código que prepara el bit de relleno
            jmp    Modo0
            jmp    Modo1
            jmp    Modo2
            jmp    Modo3

            ;Código específico de cada modo
Modo0       clrc                  ;C=0
            jmp    Modo3
Modo1       bit.b  #BIT7, 0(r12) ;Copiar msb en C
            jmp    Modo3
Modo2       bit.b  #BIT0, 0(r15) ;Copiar lsb en C
            jmp    Modo3
Modo3       ...                   ;En todos los casos se sigue por aquí
```

Finalmente, ya que hay que pasar por el modo 3 en todos los casos, se podría usar una subrutina específica con una tabla de saltos y volver para hacer la parte genérica:

```
            and.w  #BIT1|BIT0, r14;Ignorar todo lo que no sean los 2 lsb de modo
            rla.w  r14           ;Modo por 2 para indexar tabla de saltos
            call   TabSaltos(r14);Saltar al código que prepara el bit de relleno
            ;Código genérico de desplazamiento
            mov.w  sr, r15       ;Inicializar R15 para primera iteración del bucle

            ;Subrutinas específicas de cada modo
TabSaltos   .word  Modo0
            .word  Modo1
            .word  Modo2
            .word  Modo3

Modo0       clrc                 ;C=0
            ret
Modo1       bit.b  #BIT7, 0(r12) ;Copiar msb en C
            ret
Modo2       bit.b  #BIT0, 0(r15) ;Copiar lsb en C
Modo3       ret
```

Veamos el código usando la tercera técnica:

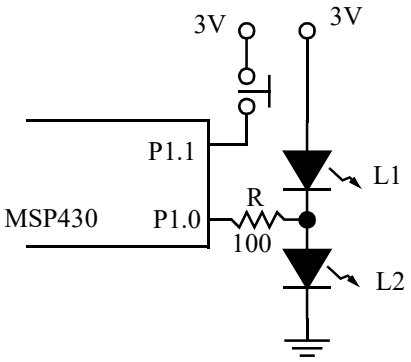
```
;-------------------------------------------------------------------------------
; void DesUnivDer (void *p, int n, int modo);                                v1.0
;
; Divide un número con signo por 2.
;-------------------------------------------------------------------------------
DesUnivDer mov.w  sr, r11       ;Salvar C (en realidad todoe el SR) en R11
           cmp.w  #1, r13       ;Comprobar precondiciones
           jl     DUDFin        ;n<1? Salir
           mov.w  r12, r15      ;R15=p
           add.w  r13, r12      ;R12=p+n (una posición más allá del último byte)
           dec.w  r12           ;R12 aputa al último byte
           and.w  #BIT1|BIT0, r14;Ignorar todo lo que no sean los 2 lsb de modo
           rla.w  r14           ;Modo por 2 para indexar tabla de saltos
           mov.w  r11, sr       ;Recuperar el carry (sólo últil en el modo=3)
           call   TabSaltos(r14);Preparar el bit de relleno. Se deja en el carry
           ;Código genérico de desplazamiento
           mov.w  sr, r11       ;Inicializar R11 para primera iteración del bucle
DUDBuc     mov.w  r11, sr       ;Recuperar bit de carry
           rrc.b  @r12          ;Desplazar un bit a la derecha
           mov.w  sr, r11       ;Salvar bit de carry
           dec.w  r12           ;Siguiente byte
           dec.w  r13           ;Contabilizar byte
           jnz    DUDBuc        ;Quedan más? Seguir
DUDFin     ret

           ;Subrutinas específicas de cada modo
; Modo0: C = 0 ------------------------------------------------------------------
Modo0      clrc                 ;C=0
           ret
; Modo1: C = lsb----------------------------------------------------------------
Modo1      bit.b  #BIT7, 0(r12) ;Copiar msb en C. R12 apunta a MSB
           ret
; Modo2: C = msb----------------------------------------------------------------
Modo2      bit.b  #BIT0, 0(r15) ;Copiar lsb en C. R15 apunta a LSB
; Modo2: C = C------------------------------------------------------------------
Modo3      ret
; Tabla de direcciones de subrutinas--------------------------------------------
TabSaltos  .word  Modo0
           .word  Modo1
           .word  Modo2
           .word  Modo3
```

CRITERIO DE CORRECCIÓN

3.- **[3 puntos]** Considere el circuito de la figura. La tensión directa de los leds es Vf=2V, por lo que no es posible encender los dos leds simultáneamente con una tensión de alimentación de 3V. Explique y haga un programa en ensamblador del MSP430 que gestione ***por interrupciones*** el encendido/apagado de los leds en función del pulsador. Inicialmente ambos leds están apagados. Mientras se mantiene pulsada la tecla, uno de los leds permanece encendido y se apaga cuando se suelta. En cada pulsación cambiará el led operativo (la primera pulsación enciende el L1, la segunda el L2, la tercera el L1,...).

Nota: Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados y la pila inicializada.



SOLUCIÓN

La lógica irá cambiando la configuración del P1.0 cada vez que se reciba la interrupción de P1.1:

P1DIR.0	P1OUT.0	L1	L2
0	-	OFF	OFF
1	0	ON	OFF
1	1	OFF	ON

Se podría usar una variable para guardar el turno de a qué led le toca encenderse. Bastaría con un bit. En lugar de eso, se usará el propio estado del puerto P1.0 para llevar la cuenta de qué led se enciende, lo que simplifica mucho el código.

En cada IRQ se comprobará primero qué flanco ha provocado la misma. Si es el de subida (IES=0), será la pulsación de la tecla. Si es el de bajada (IES=1), la liberación. En el primer caso se conmutará el puerto del led (lo que hace que cambie qué led se encenderá) y se pondrá como salida (lo que hará que se encienda). En el segundo se pone el puerto como entrada, provocando que se apaguen ambos leds.

```
LED         .equ   BIT0
SW          .equ   BIT1

main        bis.b  #SW, &P1REN   ;Activar resistencia del SWITCH
            bic.b  #SW, &P1OUT   ;Pulldowon para SW
            bis.b  #LED, &P1OUT  ;L2 activo (pero apagado). En la ISR se invierte
            ;bic.b #LED|SW, &P1DIR;SWITCH entrada y LEDS apagados

            bic.b  #SW, &P1IES   ;SW sensible flanco subida
            bic.b  #SW, &P1IFG   ;Borrar eventos de SW
            bis.b  #SW, &P1IE    ;Habilitar IRQ para SW

            bis.w  #LPM4|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------
; P1ISR                                                              v1.1
; Subrutina de servicio de la interrupción del puerto P1
; Control de cambios:
```

```
; v1.1: Se ha optimizado la lógica.
;-------------------------------------------------------------------------------
P1ISR       bit.b  #SW, &P1IN    ;Se ha pulsado la tecla?
            jz     SwOff         ;...no. Se ha soltado. No cambiar de led
            xor.b  #LED, &P1OUT  ;...sí. Cambiar led activo
SwOff       xor.b  #LED, &P1DIR  ;Cambiar estado del led (apaga/enciende)
            xor.b  #SW,  &P1IES  ;Detectar flanco contrario del SW
            bic.b  #SW,  &P1IFG  ;Borrar IRQ
            reti
            .intvec PORT1_VECTOR, P1ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

main: 30%

ISR: 70%