SISTEMAS BASADOS EN MICROPROCESADOR **Primer parcial 2023-2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3	P4

Duración 2:00 horas

1.- (2 puntos) Preguntas cortas:

a) Ponga un ejemplo de instrucción cuyo código máquina ocupe una palabra en el MPS430. Repita para dos y para tres palabras.

b) Indique los acciones que suceden en el procesador desde que se acepta una solicitud de interrupción.

c) Escriba un trozo de código en ensamblador que realice un bucle `for` en el que el registro de control recorra los valores pares del 2 al 100.

SOLUCIÓN

a)

```
        mov.w   r12, r13        ;Una palabra
        mov.w   #3, r13         ;Dos palabras: la extra para el 3
        mov.w   #3, 0(r13)      ;Tres palabras: las extras para el 3 y el 0
```

b)

c)

```
INI     .equ    2
FIN     .equ    100
        mov.w   #INI, r12     ;R12=2 (valor inicial del bucle)
for     ...                   ;cuerpo del bucle
        incd.w  r12           ;siguiente elemento
        cmp.w   #FIN+1, r12   ;Comparar con el último+1 para incluir FIN
        jl      for
```

CRITERIO DE CORRECCIÓN

a) Instrucciones de diversos tamaños: 30%

b) Acciones tras una IRQ: 40%

c) bucle for de 2 a 100 de 2 en 2: 30%

2.- (2 puntos) Analice el siguiente código e indique qué hace. Señale si sigue o no el convenio de llamada de C. En caso afirmativo, muestre el prototipo. En caso negativo, enumere los parámetros de entrada/salida, ubicación y tipo. Calcule el número de ciclos de ejecución, así como el tiempo total para f_{MCLK}=1MHz. Ensamble manualmente el programa. ¿Cuántas palabras ocupa el código? NOTA: Suponga que Var=0x1234.

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
nss     clr.w   r12             ;_____________|___|_____________________________
        bit.b   #BIT0, &Var     ;_____________|___|_____________________________
        jz      nssFin          ;_____________|___|_____________________________
        mov.w   #1, r12         ;_____________|___|_____________________________
nssFin  ret                     ;_____________|___|_____________________________
```

SOLUCIÓN

a) Análisis:

```
nss     clr.w   r12             ;R12=0
        bit.b   #BIT0, &Var     ;Es Var par?
        jz      nssFin          ;...sí, salir devolviendo 0
        mov.w   #1, r12         ;...no, devolver 1
nssFin  ret
```

La subrutina comprueba si el byte que hay en la posición 0x1234 es par. En ese caso devuelve un 0 por R12 y un 1 si es impar.

La salida es por R12, pero la entrada sería a través de la variable Var. Es discutible si sigue o no el convenio de llamada de C, ya que se está chequeando si una variable global es par o no. En C es posible hacer eso, en cuyo caso el prototipo sería:

```
int nss (void);
```

Aunque, estrictamente hablando, la entrada no es usando un parámetro.

Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
nss         clr.w  r12          ;mov.w #0, r12     | 1 |
            bit.b  #BIT0, &Var  ;BIT0=1            | 4 |
            jz     nssFin       ;+1                | 2 |
            mov.w  #1, r12      ;                  | 1 |
nssFin      ret                 ;mov.w @sp+,pc     | 3 |
```

Se ha comentado el código para expresar las instrucciones emuladas y el número de ciclos de cada instrucción. Si Var es par, la subrutina tarda 10 ciclos. En caso contrario, 11. Para una frecuencia de 1MHz, el tiempo total sería x*1μs, 10/11μs.

Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos), las de tipo II (las que tienen 1 operando) y las de salto relativo. En éstas sólo hay que codificar la condición y el desplazamiento, que se calcula como el número de palabras desde el origen (instrucción posterior al salto), al destino, es decir, +1, que en complemento a 2 es 0000000001. Cada instrucción ocupa una palabra, excepto la segunda que necesita una palabra extra para almancenar la dirección 0x1234, dando un total de 6 palabras o 12 bytes.

```
;                               Emulación           Código máquina
;                                                   OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                   000100 OpC BDf -Rd-(tipo II)
;                                                   001 Cnd Desplazami (salto)
;-------------------------------------------------------------------------------
nss         clr.w  r12          ;mov.w #0, r12      0100 0011 0000 1100 = 430C
            bit.b  #BIT0, &Var  ;BIT0=1             1011 0011 1101 0010 = B3D2 1234
            jz     nssFin       ;+1                 001 001 0000000001  = 2401
            mov.w  #1, r12      ;mov.w #1, r12      0100 0011 0001 1100 = 431C
nssFin      ret                 ;mov.w @r1+,r0      0100 0001 0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

- Análisis: 40%
- Tiempo de ejecución: 20%
- Codificación: 40%

3.- (3 puntos) Desarrolle la siguiente función en ensamblador del MSP430, siguiendo el convenio de llamada estilo C:

```
void strinv (char *s);
```

`strinv` invierte la cadena que se pasa como parámetro. Es decir, hace que el primer carácter pase a ser el último, el segundo el penúltimo,...

SOLUCIÓN

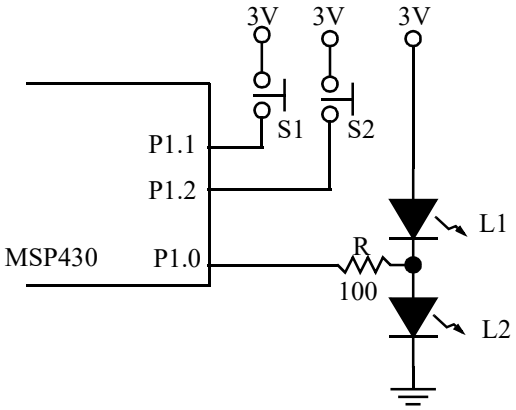
El algoritmo coloca R13 apuntando al último carácter (`strinvB1`) y hace un bucle (`strinvB2`) para intercambiar el carácter apuntado por R12 (principio) y R13 (final). Después avanza R12 y retrocede R13. El proceso se acaba cuando R12>=R13. Nótese que el caso R12=R13 se daría cuando el número de caracteres es impar y el R12>R13 si es par. En cualquiera de ellos se para el proceso de intercambio porque ya se ha completado. Obsérvese que se ha implementado un bucle *while* con la estructura de un *do-while* (la condición se comprueba al final), pero haciendo un salto la primera vez a la comprobación de la condición, para hacerlo más eficiente. Esta comprobación inicial se hace para tener en cuenta el caso de cadena vacía y el de cadena con sólo un carácter.

```
;-------------------------------------------------------------------------
; void strinv (char *s)                                                v1.0
;
; Invierte la cadena que se pasa como parámetro.
;-------------------------------------------------------------------------
strinv      mov.w     r12, r13  ;R13=s
            ;Mueve R13 al final de la cadena
strinvB1    mov.b     @r13+, r14;Leer carácter y avanzar puntero
            tst.b     r14       ;Fin de cadena?
            jnz       strinvB1  ;...no, seguir buscando
            decd.w    r13       ;...sí. R13 apunta al último carácter

            ;Intercambiar el byte del principio con el del final.
            ;El proceso acaba cuando R12>=R13 (puntero inicial alcanza al final)
            jmp       strinvPunt;Empezar comprobando condición de bucle
strinvB2    mov.b     @r12+, r14;Leer carácter del principio y avanzar puntero
            mov.b     @r13, -1(r12);Leer carácter del final y guardar al inicio
            mov.b     r14, 0(r13);Guardar carácter del principio en final
            dec.w     r13       ;Retroceder puntero final
strinvPunt  cmp.w     r13, r12  ;Punteros cruzados?
            jlo       strinvB2  ;...no, siguiente par de caracteres
            ret                 ;...sí, hemos terminado
```

4.- (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*** y en el modo de menor consumo el encendido/apagado de los leds en función de los pulsadores. Inicialmente ambos leds están apagados, pero el led activo es el L1 y el modo de pulsación. Cada vez que se pulsa S2, se cambia el led activo. En modo de pulsación, cada vez que se pulsa S1 se enciende el led activo y se apaga cuando se suelta. En modo de conmutación, cada pulsación de S1 hace que se invierta el estado del led activo. Se cambia de modo cuando se pulsa S2 estando S1 pulsado.

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



SOLUCIÓN

Dado que el pulsadores no tienen resistencia externa, hay que usar las internas. En este caso deben ser de *pull-down*. El control de los leds es con lógica positiva para L2 y negativa para L1 (1 enciende L2; 0 enciende L1; entrada, apaga ambos). En el programa principal se configuran los puertos: los leds como entrada para apagarlos, pero con salida 0 para que el L1 esté activo y los switches como entradas con resistencia de *pull-down* sensibles a flanco de subida. Para recordar el modo de trabajo de S1 (pulsación/conmutación) se usará una bandera. Basta con un bit de la variable Modo. Se inicializa este bandera a 0. Por último se habilitan las interrupciones y se entra en LPM4.

El resto del trabajo se hace por interrupciones. La lógica de S1 es relativamente sencilla: en modo pulsación cambia el estado del led en cada flanco y configura el flanco opuesto para la siguiente vez. En modo conmutación, conmuta el estado del led en los flancos de subida. Los de bajada no se usan. Para hacer el código más compacto, siempre se conmuta el estado del led L1 y, si estamos en pulsación, se cambia de flanco.

La lógica de S2 es un poco más compleja. En cualquier caso cambia el led activo. Para ello lo único que tiene que hacer el invertir el estado de la salida de P1.0. En caso de que S2 se haya pulsado estando pulsado S1, hay que gestionar el cambio de modo (invirtiendo el bit0 de Modo) y actuando sobre la salida de L1 para que no quede en un estado incoherente. Nótese que en este caso, S1 se encuentra pulsado siempre. Si el modo que ha quedado activo, hay que asegurarse de que el led activo está encendido (poniendo el puerto como salida) y dejando activo el flanco de bajada para que se pueda apagar cuando se suelte la tecla. En cambio, si se ha quedado activo el modo conmutación, sólo hay que configurar el flanco de subida, ya que en la pulsación de S1, estando en modo pulsación, se configuró el flanco de bajada que ahora hay que ignorar.

```
LEDS        .equ   BIT0
S1          .equ   BIT1
S2          .equ   BIT2

            .bss   Modo, 1      ;Controla el modo del S1: 0, pulsación; 1, conmutación

main        ;L1 salida, apagado
            bic.b  #LEDS, &P1OUT;L1 activo
            ;bic.b #LEDS, &P1DIR;...pero apagado
            ;S1 y S2 entrada, pulldown, sensible en flanco de subida, IRQ
            ;bic.b #S1|S2, &P1DIR;Entrada
            bis.b  #S1|S2, &P1REN;Habilitar resistencias
            bic.b  #S1|S2, &P1OUT;...de pullup
            bic.b  #S1|S2, &P1IES;Sensibles flanco subida
            bic.b  #S1|S2, &P1IFG;Borrar eventos de S1 y S2
            bis.b  #S1|S2, &P1IE ;Habilitar IRQ de S1 y S2
            ;Modo pulsación
            bic.b  #BIT0, &Modo  ;Pulsación

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

;-------------------------------------------------------------------------------
; P1ISR                                                                  v1.0
; Subrutina de servicio de la interrupción del puerto P1
;-------------------------------------------------------------------------------
P1ISR       add.w  &P1IV, PC     ;Saltar a ISR. Borra flag
            reti                 ;IV=0 No IRQ
            reti                 ;IV=2. Px.0
            jmp    P1_1          ;IV=4. Px.1
            ;jmp   P1_2          ;IV=6. Px.2

            ;Rutina de servicio de la IRQ del P1_2
P1_2        xor.b  #LEDS, &P1OUT;Cambiar led activo
            bit.b  #S1, &P1IN    ;S1 pulsado mientras se pulsa S2?
            jz     P1_2Fin       ;...no, salir
            xor.b  #BIT0, &Modo  ;...sí. Cambiar de modo
            bit.b  #BIT0, &Modo  ;Ha quedado modo pulsación?
            jz     Pulsa         ;...sí, S1 pulsado=>encender led y flanco bajada
            bic.b  #S1, &P1IES   ;...no. Sólo reaccionar a los flancos de subida
            jmp    P1_2Fin
Pulsa       bis.b  #LEDS, &P1DIR;Encender led activo
            bis.b  #S1, &P1IES   ;Reaccionar a los flancos de bajada
P1_2Fin     reti

            ;Rutina de servicio de la IRQ del P1_1
P1_1        xor.b  #LEDS, &P1DIR;Cambiar estado del led activo
            bit.b  #BIT0, &Modo  ;Modo pulsación?
            jnz    P1_1Fin       ;...no, salir. Sólo flancos de subida
            xor.b  #S1, &P1IES   ;...sí, cambiar flanco de S1
P1_1Fin     reti

            .intvecRESET_VECTOR, main
            .intvecPORT1_VECTOR, P1ISR
```

CRITERIO CORRECCIÓN

- Programa principal: 30%
- P1.1 ISR 40%
- P1.2 ISR 30%