

Evaluación del Laboratorio Número 3

Sistemas con Microprocesadores

6/11/2024

Instrucciones

- La presente evaluación es individual y podrá utilizar todo el material escrito o digital del que disponga.
- La misma consta de 1 ejercicios en total y la duración será de 90 minutos.
- En cada ejercicio se indica el tiempo estimado para resolverlo y el puntaje que otorga el mismo.
- No se permitirá intercambiar ni prestar ningún tipo de elemento entre los estudiantes.
- Cada ejercicio funcionando deberá ser aprobado por un docente para que se compute como válido antes de comenzar con el siguiente ejercicio.
- El código de los ejercicios aprobados por el docente debe ser enviados por correo electrónico a las direcciones *mgiori@herrera.unt.edu.ar* y *ltek@herrera.unt.edu.ar* antes de finalizar la evaluación.

Ejercicios para resolver

1) 90 Minutos - 30, 20, 30, 20 Puntos

Implemente un reloj de ajedrez de 2 jugadores. Este reloj tiene la característica de tener 2 cuentas regresivas, 1 por cada jugador y 2 botones. En nuestro caso y para simplificar la cuenta regresiva sera desde 99 a 0.

- a) Escriba un programa que muestre en el **Display 1 y Display 2** una cuenta descendente desde 99 hasta 0. Estos displays corresponden al jugador 1.
- b) Escriba un programa que muestre en el **Display 3 y Display 4** una cuenta descendente desde 99 hasta 0. Estos display corresponden al jugador 2.
- c) La tecla **Aceptar** corresponderá al jugador 1 y cada vez que se presione, comienza la cuenta regresiva en el **Display 3 y 4** y detiene la cuenta en el **Display 1 y 2**.
- d) La tecla **Cancelar** corresponderá al jugador 2 y cada vez que se presione, comienza la cuenta regresiva en el **Display 1 y 2** y detiene la cuenta en el **Display 3 y 4**.

Soluciones completas

```

1) a) /* Copyright 2016-2020, Laboratorio de Microprocesadores
    * Facultad de Ciencias Exactas y Tecnología
    * Universidad Nacional de Tucuman
    * http://www.microprocesadores.unt.edu.ar/
    * Copyright 2016-2020, Esteban Volentini <evolentini@herrera.unt.edu.ar>
    * All rights reserved.
    *
    * Redistribution and use in source and binary forms, with or without
    * modification, are permitted provided that the following conditions are met:
    *
    * 1. Redistributions of source code must retain the above copyright notice,
    *   this list of conditions and the following disclaimer.
    *
    * 2. Redistributions in binary form must reproduce the above copyright notice,
    *   this list of conditions and the following disclaimer in the documentation
    *   and/or other materials provided with the distribution.
    *
    * 3. Neither the name of the copyright holder nor the names of its
    *   contributors may be used to endorse or promote products derived from this
    *   software without specific prior written permission.
    *
    * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
    * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
    * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
    * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
    * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
    * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
    * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
    * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
    * POSSIBILITY OF SUCH DAMAGE.
    */

.cpu cortex-m4           // Indica el procesador de destino
.syntax unified          // Habilita las instrucciones Thumb-2
.thumb                  // Usar instrucciones Thumb y no ARM

#include "configuraciones/lpc4337.s"
#include "configuraciones/rutinas.s"

/*****
/* Definiciones de macros
*****/

// Recursos utilizados por el segmento A
// Numero de puerto de entrada/salida utilizado en el Segmento A
.equ SEG_A_PORT,    4
// Numero de terminal dentro del puerto de e/s utilizado en el Segmento A
.equ SEG_A_PIN,     0
// Numero de bit GPIO utilizado en el Segmento A
.equ SEG_A_BIT,     0
// Mascara de 32 bits con un 1 en el bit correspondiente al Segmento A
.equ SEG_A_MASK,    (1 << SEG_A_BIT)

// Recursos utilizados por el segmento B
.equ SEG_B_PORT,    4
.equ SEG_B_PIN,     1
.equ SEG_B_BIT,     1
.equ SEG_B_MASK,    (1 << SEG_B_BIT)

// Recursos utilizados por el segmento C
.equ SEG_C_PORT,    4

```

```

.equ SEG_C_PIN,      2
.equ SEG_C_BIT,      2
.equ SEG_C_MASK,     (1 << SEG_C_BIT)

// Recursos utilizados por el segmento D
.equ SEG_D_PORT,     4
.equ SEG_D_PIN,      3
.equ SEG_D_BIT,      3
.equ SEG_D_MASK,     (1 << SEG_D_BIT)

// Recursos utilizados por el segmento E
.equ SEG_E_PORT,     4
.equ SEG_E_PIN,      4
.equ SEG_E_BIT,      4
.equ SEG_E_MASK,     (1 << SEG_E_BIT)

// Recursos utilizados por el segmento F
.equ SEG_F_PORT,     4
.equ SEG_F_PIN,      5
.equ SEG_F_BIT,      5
.equ SEG_F_MASK,     (1 << SEG_F_BIT)

// Recursos utilizados por el segmento G
.equ SEG_G_PORT,     4
.equ SEG_G_PIN,      6
.equ SEG_G_BIT,      6
.equ SEG_G_MASK,     (1 << SEG_G_BIT)

// Recursos utilizados por los segmentos
// Numero de puerto GPIO utilizado por los segmentos
.equ SEG_GPIO,       2
// Desplazamiento para acceder a los registros GPIO de los segmentos
.equ SEG_OFFSET,     ( 4 * SEG_GPIO )
// Mascara de 32 bits con un 1 en los bits correspondiente a cada segmento
.equ SEG_MASK,       ( SEG_A_MASK | SEG_B_MASK | SEG_C_MASK | SEG_D_MASK | SEG_E_MASK
    | SEG_F_MASK | SEG_G_MASK)

// -----
// Recursos utilizados por el Dígito 1
.equ DIG_1_PORT,     0
.equ DIG_1_PIN,      0
.equ DIG_1_BIT,      0
.equ DIG_1_MASK,     (1 << DIG_1_BIT)

// Recursos utilizados por el Dígito 2
.equ DIG_2_PORT,     0
.equ DIG_2_PIN,      1
.equ DIG_2_BIT,      1
.equ DIG_2_MASK,     (1 << DIG_2_BIT)

// Recursos utilizados por el Dígito 3
.equ DIG_3_PORT,     1
.equ DIG_3_PIN,      15
.equ DIG_3_BIT,      2
.equ DIG_3_MASK,     (1 << DIG_3_BIT)

// Recursos utilizados por el Dígito 4
.equ DIG_4_PORT,     1
.equ DIG_4_PIN,      17
.equ DIG_4_BIT,      3
.equ DIG_4_MASK,     (1 << DIG_4_BIT)

// Recursos utilizados por los Dígitos
.equ DIG_N_GPIO,     0
.equ DIG_N_OFFSET,   ( DIG_N_GPIO << 2)
.equ DIG_N_MASK,     ( DIG_1_MASK | DIG_2_MASK | DIG_3_MASK | DIG_4_MASK )

```

```

// -----
// Recursos utilizados por el led 1
.equ LED_1_PORT,      2
.equ LED_1_PIN,       10
.equ LED_1_BIT,       14
.equ LED_1_MASK,      (1 << LED_1_BIT)

.equ LED_1_GPIO,      0
.equ LED_1_OFFSET,    ( LED_1_GPIO << 2)

// -----
// Recursos utilizados por el boton de aceptar
.equ ACCEPTAR_PORT,   3
.equ ACCEPTAR_PIN,    1
.equ ACCEPTAR_BIT,    8
.equ ACCEPTAR_MASK,   (1 << ACCEPTAR_BIT)

// Recursos utilizados por el boton de cancelar
.equ CANCELAR_PORT,   3
.equ CANCELAR_PIN,    2
.equ CANCELAR_BIT,    9
.equ CANCELAR_MASK,   (1 << CANCELAR_BIT)

// Recursos utilizados por los botones
.equ BOT_GPIO,        5
.equ BOT_OFFSET,      ( BOT_GPIO << 2)
.equ BOT_MASK,        ( ACCEPTAR_MASK | CANCELAR_MASK )

/*****
/* Vector de interrupciones
*****/

.section .isr          // Define una seccion especial para el vector
.word stack            // 0: Initial stack pointer value
.word reset+1          // 1: Initial program counter value
.word handler+1        // 2: Non mascarable interrupt service routine
.word handler+1        // 3: Hard fault system trap service routine
.word handler+1        // 4: Memory manager system trap service routine
.word handler+1        // 5: Bus fault system trap service routine
.word handler+1        // 6: Usage fault system tram service routine
.word 0                // 7: Reserved entry
.word 0                // 8: Reserved entry
.word 0                // 9: Reserved entry
.word 0                // 10: Reserved entry
.word handler+1        // 11: System service call trap service routine
.word 0                // 12: Reserved entry
.word 0                // 13: Reserved entry
.word handler+1        // 14: Pending service system trap service routine
.word systick_isr+1    // 15: System tick service routine
.word handler+1        // 16: Interrupt IRQ service routine

/*****
/* Definicion de variables globales
*****/

.section .data         // Define la seccion de variables (RAM)
jug1:
.byte 9,9              // Vector de 2 bytes para el contador del jugador 1
jug2:
.byte 9,9              // Vector de 2 bytes para el contador del jugador 2
turno:
.byte 0                // Variable compartida (0 = j1, 1 = j2)
divisor:
.hword 0               // Variable para el divisor del SysTick

```

```

/*****
/* Programa principal
*****/

.global reset           // Define el punto de entrada del codigo
.section .text          // Define la seccion de codigo (FLASH)
.func main              // Indica al depurador el inicio de una funcion
reset:
// Mueve el vector de interrupciones al principio de la segunda RAM
LDR R1,=VTOR
LDR R0,=#0x10080000
STR R0,[R1]

// Llama a una subrutina para configurar el systick
BL systick_init

// Configura los pines de los Segmentos como gpio sin pull-up
LDR R1,=SCU_BASE
MOV R0,#(SCU_MODE_INACT | SCU_MODE_INBUFF_EN | SCU_MODE_ZIF_DIS | SCU_MODE_FUNC0)
STR R0,[R1,#(4 * (32 * SEG_A_PORT + SEG_A_PIN))]
STR R0,[R1,#(4 * (32 * SEG_B_PORT + SEG_B_PIN))]
STR R0,[R1,#(4 * (32 * SEG_C_PORT + SEG_C_PIN))]
STR R0,[R1,#(4 * (32 * SEG_D_PORT + SEG_D_PIN))]
STR R0,[R1,#(4 * (32 * SEG_E_PORT + SEG_E_PIN))]
STR R0,[R1,#(4 * (32 * SEG_F_PORT + SEG_F_PIN))]
STR R0,[R1,#(4 * (32 * SEG_G_PORT + SEG_G_PIN))]

MOV R0,#(SCU_MODE_INACT | SCU_MODE_INBUFF_EN | SCU_MODE_ZIF_DIS | SCU_MODE_FUNC0)
STR R0,[R1,(LED_1_PORT << 7 | LED_1_PIN << 2)]

// Se configuran los pines de los digitos 1 al 4 como gpio sin pull-up
MOV R0,#(SCU_MODE_INACT | SCU_MODE_INBUFF_EN | SCU_MODE_ZIF_DIS | SCU_MODE_FUNC0)
STR R0,[R1,(DIG_1_PORT << 7 | DIG_1_PIN << 2)]
STR R0,[R1,(DIG_2_PORT << 7 | DIG_2_PIN << 2)]
STR R0,[R1,(DIG_3_PORT << 7 | DIG_3_PIN << 2)]
STR R0,[R1,(DIG_4_PORT << 7 | DIG_4_PIN << 2)]

// Configura los pines de las teclas como gpio con pull-up
MOV R0,#(SCU_MODE_PULLUP | SCU_MODE_INBUFF_EN | SCU_MODE_ZIF_DIS | SCU_MODE_FUNC0)
STR R0,[R1,((ACEPTAR_PORT << 7 | ACEPTAR_PIN) << 2)]
STR R0,[R1,((CANCELAR_PORT << 7 | CANCELAR_PIN) << 2)]

// Apaga todos los bits gpio de los segmentos
LDR R1,=GPIO_CLR0
LDR R0,=SEG_MASK
STR R0,[R1,#SEG_OFFSET]
// Se apagan los bits gpio correspondientes a los digitos
LDR R0,=DIG_N_MASK
STR R0,[R1,#DIG_N_OFFSET]
LDR R0,=LED_1_MASK
STR R0,[R1,#LED_1_OFFSET]

// Configura los bits gpio de los segmentos como salidas
LDR R1,=GPIO_DIR0
LDR R0,[R1,#SEG_OFFSET]
ORR R0,#SEG_MASK
STR R0,[R1,#SEG_OFFSET]
LDR R0,=LED_1_MASK
STR R0,[R1,#LED_1_OFFSET]
// Se configuran los bits gpio de los digitos como salidas
LDR R0,[R1,#DIG_N_OFFSET]
ORR R0,#DIG_N_MASK
STR R0,[R1,#DIG_N_OFFSET]

// Configura los bits gpio de los botones como entradas
LDR R0,[R1,#BOT_OFFSET]

```

```

BIC R0,#BOT_MASK
STR R0,[R1,#BOT_OFFSET]
MOV R4,#0

refrescar:
WFI
LDR R2,=jug2
LDRB R0,[R2],#1           // Guardo el digito a mostrar en R0
CMP R4,0
BNE dig1
BL segmentos
MOV R1,#1                 // Prender el LED1
BL mostrar

dig1:
CMP R4,5
BNE dig2
LDRB R0,[R2]             // Guardo el digito a mostrar en R0
BL segmentos
MOV R1,#2                 // Prender el LED2
BL mostrar

dig2:
LDR R2,=jug1
LDRB R0,[R2],#1          // Guardo el digito a mostrar en R0
CMP R4,10
BNE dig3
BL segmentos
MOV R1,#4                 // Prender el LED3
BL mostrar

dig3:
CMP R4,15
BNE endref
LDRB R0,[R2]             // Guardo el digito a mostrar en R0
BL segmentos
MOV R1,#8                 // Prender el LED4
BL mostrar
endref:
CMP R4,20
ITE EQ
MOVEQ R4,#0
ADDNE R4,#1
B refrescar
.pool                      // Almacenar las constantes de codigo
.endfunc

/*****
*/
/* Rutina de inicializacion del SysTick
*/
/*****
*/

.func systick_init
systick_init:
CPSID I                  // Se deshabilitan globalmente las interrupciones

// Se configura prioridad de la interrupcion
LDR R1,=SHPR3            // Se apunta al registro de prioridades
LDR R0,[R1]              // Se cargan las prioridades actuales
MOV R2,#2                // Se fija la prioridad en 2
BFI R0,R2,#29,#3         // Se inserta el valor en el campo
STR R0,[R1]              // Se actualizan las prioridades

// Se habilita el SysTick con el reloj del nucleo

```

```

LDR R1,=SYST_CSR
MOV R0,#0x00
STR R0,[R1]                // Se quita el bit ENABLE

// Se configura el desborde para un periodo de 1 ms
LDR R1,=SYST_RVR
LDR R0,=#(96000 - 1)
STR R0,[R1]                // Se especifica el valor de RELOAD

// Se inicializa el valor actual del contador
LDR R1,=SYST_CVR
MOV R0,#0
// Escribir cualquier valor limpia el contador
STR R0,[R1]                // Se limpia COUNTER y flag COUNT

// Se habilita el SysTick con el reloj del nucleo
LDR R1,=SYST_CSR
MOV R0,#0x07
STR R0,[R1]                // Se fijan ENABLE, TICKINT y CLOCK_SRC

CPSIE I                    // Se habilitan globalmente las interrupciones
BX LR                     // Se retorna al programa principal
.pool                    // Se almacenan las constantes de codigo
.endfunc

/*****
*/
/* Rutina de servicio para la interrupcion del SysTick
*/
/*****
*/

.func systick_isr
systick_isr:
PUSH {LR}

LDR R0,=turno              // Cargo la dir de turno en R0
LDRB R1,[R0]              // Cargo el valor de R0 en R1
LDR R2,=GPIO_PIN0         // Cargo la dir de GPIO_PIN0 en R2
LDR R3,[R2,#BOT_OFFSET]   // Carga el estado actual de las teclas
TST R3,#ACEPTAR_MASK      // Verifica el estado del bit correspondiente a la tecla
                          Aceptar
IT EQ                      // Si la tecla esta apretada
MOVEQ R1,#1               // Cambia el turno a jugador 2
TST R3,#CANCELAR_MASK     // Verifica el estado del bit correspondiente a la tecla
                          Cancelar
IT EQ                      // Si la tecla esta apretada
MOVEQ R1,#0               // Cambia el turno a jugador 1
STRB R1,[R0]              // Guarda el valor de R1 en memoria

LDR R2,=divisor            // Cargo la dir de divisor en R2
LDRH R3,[R2]              // Cargo el valor de R2 en R3
ADD R3,#1                 // Agrego 1 al contador divisor

CMP R3,1000               // Comparo si el divisor llego a 1000
STRH R3,[R2]              // Guardo divisor en memoria
BNE fin                   // Si no llego a 1000, vuelve a ejecutar
MOV R3,#0                 // Si llego a 1000, divisor va a 0
STRH R3,[R2]              // Guardo divisor en memoria

CMP R1,#0                 // Comparo si el turno es del jugador 1
BNE j2                    // Si no es del jugador 1, salta a j2
LDR R0,=jug1              // Cargo la dir de jug1 en R0
BL restarReloj            // Llamo a la funcion de restar reloj
B fin
j2:
LDR R0,=jug2              // Cargo la dir de jug2 en R0

```

```

BL restarReloj          // Llamo a la funcion de restar reloj
fin:
POP {LR}
BX LR

.endfunc

.func segmentos
// Recibe en R0 el num a convertir, devuelve en R0 el num convertido
segmentos:
PUSH {R4}              // Guardo el valor de R4
LDR R4,=tabla          // Apunta R4 al bloque con la tabla
LDRB R0,[R4,R0]        // Cargar en R0 el elemento convertido
POP {R4}              // Devuelvo R4 a su valor original
BX LR                 // Vuelta a la llamada

.pool                 // Almacenar las constantes de código

tabla:                // Define la tabla de conversión
.byte 0x3F,0x06,0x5B,0x4F,0x66
.byte 0x6D,0x7D,0x07,0x7F,0x67
.endfunc

.func mostrar
mostrar:
PUSH {R4-R5}
LDR R4,=GPIO_PIN0
MOV R5,#0
STRB R5,[R4,#DIG_N_OFFSET]
STRB R5,[R4,#SEG_OFFSET]

STRB R0,[R4,#SEG_OFFSET]
STRB R1,[R4,#DIG_N_OFFSET]
POP {R4-R5}
BX LR

.endfunc
/*****
*/
/* Rutina de servicio generica para excepciones
*/
/* Esta rutina atiende todas las excepciones no utilizadas en el programa.
*/
/* Se declara como una medida de seguridad para evitar que el procesador
*/
/* se pierda cuando hay una excepcion no prevista por el programador
*/
/*****
*/
.func handler
handler:
LDR R1,=GPIO_SET0      // Se apunta a la base de registros SET
MOV R0,#LED_1_MASK    // Se carga la mascara para el LED 1
STR R0,[R1,#LED_1_OFFSET] // Se activa el bit GPIO del LED 1
B handler             // Lazo infinito para detener la ejecucion
.pool                // Se almacenan las constantes de codigo
.endfunc

.func restarReloj
// Funcion recibe R0 <- Dirección de memoria de los digitos a reducir
restarReloj:
PUSH {R4-R5}          // Push de R4 y R5

LDRH R4,[R0]          // Cargo digito[0-1] como halfword en R4
CMP R4,0              // Comparo si digito[0-1] = 0
BEQ endresta          // Si es asi, termina la funcion

```

```
LDRB R4,[R0]           // Cargo el valor de digito[0] en R4
SUB R4,#1              // Resto 1 a digito[0]
LDRB R5,[R0,#1]        // Cargo digito[1] como byte en R5

CMP R4,0               // Comparo si digito[0] < 0
ITT LT                 // Si es asi, digito[0] toma 9 y digito[1] reduce en uno
MOVL R4,#9
SUBL R5,#1

STRB R4,[R0]           // Guardo digito[0] en memoria
STRB R5,[R0,#1]        // Guardo digito[1] en memoria

endresta:
POP {R4-R5}            // Pop de R4 y R5
BX LR
.endfunc
```