Aix*Marseille université

Faculté des Sciences

Licence 3 SPI

Informatique industrielle

Travaux Dirigés

Création d'un chronomètre
Interruptions et gestion du lcd

Julien Marot

2017-2018

On souhaite programmer un chronomètre : le premier appui sur le bouton S2 déclenche le comptage, et l'affichage ; le second appui stoppe le chronomètre ; le troisième appui efface l'écran.

Dans le cadre du TP, l'appui sur un bouton ne déclenche pas d'interruptions. Par contre, vous programmerez une routine d'interruption de débordement du TIMER0 pour compter pendant une seconde.

1 Eléments fondamentaux

On présente en sous-section 1.1 les fonctions qui seront utilisées pour créer un chronomètre, et en sous-section 1.2 les registres qui sont utiles au fonctionnement du chronomètre. Ensuite, en sous-section 1.3, on montre comment insérer des lignes de code assembleur dans un programme C ; en sous-section 1.4 on montre comment placer une instruction à une case mémoire bien précise de la mémoire programme.

1.1 Les principales fonctions pour créer un chronomètre

Tout d'abord comme pour tout programme en C il existe un programme principal `void main()`.
Celui-ci initialise et configure le micro-contrôleur, initialise l'afficheur LCD, les variables de comptage. Ensuite le programme principal active le TIMER0 quand on appuie sur un bouton (RA4). Celui-ci se met à compter et, toutes les 1 seconde, une interruption de débordement a lieu. C'est le programme d'interruption qui permet d'afficher les minutes et les secondes. Un deuxième appui désactive le TIMER0 : c'est la fin de la séquence de comptage ; un troisième appui conduit à l'initialisation des variables de comptage, et ainsi de suite.

Les fonctions qui permettent de faire marcher le chronomètre correspondent d'abord au mode de fonctionnement intuitif d'un chronomètre : le micro-contrôleur détecte le début du comptage par un appui sur un bouton :

```
void bouton_appui (void);
```

Ce programme va attendre que le bouton change deux fois d'état, ce qui correspond à un appui puis un relâchement, pour pouvoir passer à l'instruction suivante qui sera de commencer à compter, s'arrêter, ou encore recommencer.

Voici la fonction qui écrit un nombre et la fonction qui affiche les variables MIN puis SEC sur l'écran LCD :

```
void lcd_ecrire_nombre(unsigned char nombre);
void lcd_affiche_chrono(void);
```

Voici la fonction 'vecteur reset d'interruption' et la fonction qui met en oeuvre le programme d'interruption :

```
void Vecteur_interruption_priorite_haute (void);
void TMR0_overflow(void);
```

Le programme d'interruption remet à zéro le bit de flag, réinitialise le TIMER0, incrémente une variable SEC (secondes) et une variable MIN (minutes) en remettant SEC à 0 lorsque MIN est incrémentée.

1.2 Les registres

En annexe, les registres qui permettent de configurer les interruptions sont décrits (extractions du datasheet du PIC) :
INTCON, INTCON2, RCON. Ces registres permettent de configurer les interruptions de cette façon-là : autoriser des interruptions venant du TMR0 ; placer l'interruption TIMER0 en priorité haute ; activer la gestion des priorités pour les interruptions.

1.3 De l'assembleur dans du C

Cet assembleur diffère néanmoins de MPASM sur les points suivants :
- commentaires rédigés dans la syntaxe du C,
- pas de valeur par défaut, i.e., les arguments d'une instruction doivent être spécifiés,

– la convention de notation en hexadecimal est 0xNN,
– les étiquettes (les labels) doivent comporter un « : »,
– pas d'adressage indexé.

Voici une exemple qui consiste à initialiser et décrémenter une variable :

```
_asm  /* User assembly code */
MOVLW 10  // Move decimal 10 to count
MOVWF count, 0
start: /* Loop until count is 0 */
DECFSZ count, 1, 0
GOTO done
BRA start
done:
_endasm
```

Cette façon de programmer est TRES pratique pour se déplacer vers une instruction munie d'un label, avec l'instruction goto :

```
 _asm
    goto TMR0_overflow    // instruction assembleur, cf. MPLAB C18 UserGuide, p.19
 _endasm
```

1.4 Directives pragma

On peut utiliser des instruction asm pour se déplacer vers un label, mais il est nécessaire de trouver une méthode pour atteindre une adresse précise. Par exemple, les interruptions sont associées à un vecteur d'interruption, qui DOIT être placé à une case mémoire bien précise de la mémoire PROGRAMME : 0×08 pour les interruptions de priorité haute, 0×18 pour les interruptions de priorité basse. Les directives pragma sont utiles dans ce sens. Pour plus d'informations voir MPLAB C18 UserGuide, pp. 20 et 29.

L'instruction assembleur

```
org h'0000'
goto init
```

est 'traduite' dans un programme C par les lignes de code suivantes :

```
#pragma code Vecteur_RESET = 0x0000
// Directive pour spécifier l'adresse memoire ou debute la fonction
// 0000 est l'adresse de la case mémoire

void Vecteur_RESET (void)
{
  // va à init
  _asm
    goto init
  _endasm
}
#pragma code
```

2 Exercice 1 : Algorigrammes

Ecrire les algorigrammes associés au mode de fonctionnement du chronomètre : le programme principal, et la routine d'interruption du TIMER0, la fonction appui sur bouton, la fonction `void lcd_affiche_chrono(void);` (sans détailler `void lcd_ecrire_nombre(unsigned char nombre);`).

3 Exercice 2 : Programmes

3.1 Programme d'interruption en C

La fonction `lcd_affiche_chrono` affiche les minutes et les secondes avec les fonctions du lcd. Pour cela nous devons programmer une interruption.

Placement du vecteur d'interruption à la case mémoire 0x08

On suppose que le label qui indique le début du programme d'interruption est `TMR0_overflow`.

Complétez le code pragma ci-dessous :

```
// Directive pour spécifier l'adresse memoire ou debute la fonction

#pragma code Vecteur_interruption_priorite_haute = _ _ _ _

void Vecteur_interruption_priorite_haute (void)
{
  // va a la routine de gestion des interruptions de priorite haute
  _asm
    goto  _ _ _ _
  _endasm
}

#pragma code
```

Routine de gestion de l'interruption de débordement du TIMER0 :

Complétez le code ci-dessous : ce code est censé remettre à 0 le bit de flag, initialiser le registre TMR0 avec

```
#pragma interrupt TMR0_overflow
void TMR0_overflow(void)
{
// RAZ du flag d'interruption venant de TMR0
INTCONbits.TMR0IF  = 0;

// RAZ TIMER 0
TMR0H =  _ _ _ _ ;
TMR0L =  _ _ _ _ ;

// Increment d'une seconde ajustement des minutes
  _ _ _ _  ;

// Ajustement des minutes (cas où on atteint 60)
  _ _ _ _
    _ _ _ _
      _ _ _ _

lcd_affiche_chrono(); // Affiche MIN SEC avec les fonctions lcd
}
```

3.2 Début du fichier principal : directives, déclarations, variables globales

Le fichier principal main.c, avant de lancer le programme principal, doit inclure les 'directives au préprocesseur' sous la forme de fichiers .h (définition de l'adresse des ports, etc.). Le fichier main doit aussi déclarer les fonctions utilisées dans le programme principal, et les variables globales. Dans cette sous-section, vous n'avez rien à compléter.

```
//================================================================================
// Filename: chronometre.c
//================================================================================
// Author:   M. Allain
// Company:  Universite Paul Cezanne
// Revision: 1.0
// Date de creation :   2013/09
// Derniere modification :
//================================================================================
// programme implémentant une base de temps et affichant sur l'afficheur LCD
// le temps écoulé (MIN:SEC) depuis que l'interrupteur RA4 a été actionné.
//
// Notes :
// * utilisation du TIMER0
// * utilisation des interruptions pour le TIMER0
// * utilisation de RA4 pour entree bouton poussoir
// * utilisation de la librairie lcd_lib.lib pour la gestion
//   d'un controleur d'affichage HD44780U à partir d'un PIC 18F4520.
// * liaison PIC / controleur d'affichage par PORTD (ie. celui
//    de la carte PICDEM2+ en version rohs (carte "verte")
// * Liaison PIC / controleur d'affichage basée sur un quartz cadencé à 4 MHz.
//================================================================================

//--------------------------------------------------------------------------------
// Directives au préprocesseur
//--------------------------------------------------------------------------------
#include <delays.h>
#include <p18f4520.h>
#include "lcd_lib.h"

// Definition concernant le bouton poussoir permettant d'enclencher / arrêter le chrono
#define PORTX_bouton PORTA
#define PORTXbits_bouton PORTAbits.RA4
#define TRISXbits_bouton TRISAbits.RA4

// Valeur initiale TMR0 = 3035 (0x0BDB) => debordement chaque seconde (quartz 4MHz)
#define MSB  0x0B
#define LSB 0xDB

// Offset dans la table ascii pour transmettre les unsigned char correspondants au caractère
#define offset  48

//--------------------------------------------------------------------------------
// Prototype de fonctions
//--------------------------------------------------------------------------------
void Vecteur_interruption_priorite_haute (void);
void TMR0_overflow(void);
void bouton_appui (void);
void lcd_ecrire_nombre(unsigned char nombre);
void lcd_affiche_chrono(void);

//--------------------------------------------------------------------------------
// variables globales
```

```
//-----------------------------------------------------------------------------
unsigned char MIN, SEC;
```

3.3 Programme principal en C

```
//-----------------------------------------------------------------------------
// Programme Principal
//-----------------------------------------------------------------------------
void main()
{
// -----------------------------------------------------------------------------
// Initialisation
// -----------------------------------------------------------------------------

// Activation/configuration de l'afficheur LCD
// (inclut le PORTD utilise pour communiquer avec le controleur d'affichage)
lcd_init();

// Configuration des ports E/S (autre que celui de l'afficheur)
PORTX_bouton  = 0; // RAZ PORT sur lequel le bouton poussoir est connecté
TRISXbits_bouton  = 1;  // broche du port ES du bouton poussoir est une ENTREE

// Configuration des interruptions (Activation des it avec priorité)
INTCONbits.TMR0IE  =   _ _ _ _ ; // Autorisation des it venant de TMR0. TMR0IE
                                                                    // est le bit Enable pou
INTCON2  = 0x84;    // 10000100            // Place l'interruption TIMER0 en priorité haute
RCONbits.IPEN  =   _ _ _ _ ;             // Active la gestion des priorités pour les interruptions

// Configuration du TIMER0 (TMR0 off/16bits/Prescaler 1:16)
T0CON = 0b    _ _ _ _ ;   // 8 bits à spécifier

// Active les interruptions
INTCONbits.GIEH  = 1;

// -----------------------------------------------------------------------------
// Boucle principale
// -----------------------------------------------------------------------------

// --> Boucle Tant Que : On boucle indéfiniement...
while(1){

// Initialisation des variables (globales) du chrono et affichage
MIN = 0;
SEC = 0;
lcd_affiche_chrono(); // Affiche MIN SEC avec les fonctions lcd

// Attente du bouton poussoir...
bouton_appui();

// On lance le comptage après que PORTXbits_bouton soit actionné...
_  _  _  _  _  _ =  _  _  _  _  _  ;  // Active le TIMER0

// Attente du bouton poussoir PORTXbits_bouton...
bouton_appui();

// On arrete le comptage...
_  _  _  _  _  _ =  _  _  _  _  _  ;  // Desactive le TIMER0

// Attente du bouton poussoir PORTXbits_bouton...
bouton_appui();

}
```

```
// --> Fin de boucle TantQue
```

4 Annexe : Registres

PIC18F2420/2520/4420/4520

REGISTER 9-2: INTCON2 REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0**: External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1**: External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2**: External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented**: Read as '0'

bit 2 **TMR0IP**: TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented**: Read as '0'

bit 0 **RBIP**: RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

FIGURE 1 –

REGISTER 9-10: RCON REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(1)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN**: Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16XXX Compatibility mode)

bit 6 **SBOREN**: Software BOR Enable bit(1)
For details of bit operation, see Register 4-1.

Note 1: Actual Reset values are determined by device configuration and the nature of the device Reset. See Register 4-1 for additional information.

bit 5 **Unimplemented**: Read as '0'

bit 4 $\overline{\textbf{RI}}$: RESET Instruction Flag bit
For details of bit operation, see Register 4-1.

bit 3 $\overline{\textbf{TO}}$: Watchdog Time-out Flag bit
For details of bit operation, see Register 4-1.

bit 2 $\overline{\textbf{PD}}$: Power-down Detection Flag bit
For details of bit operation, see Register 4-1.

bit 1 $\overline{\textbf{POR}}$: Power-on Reset Status bit
For details of bit operation, see Register 4-1.

bit 0 $\overline{\textbf{BOR}}$: Brown-out Reset Status bit
For details of bit operation, see Register 4-1.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURE 2 –

REGISTER 9-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH**: Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts

When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL**: Peripheral Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE**: INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF**: INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURE 3 –