Université Djilali BOUNAAMA, Khemis Miliana
مليانة خميس بونعامة جيلالي جامعة
Faculté des Sciences et de la Technologie
والتكنولوجيا العلوم كلية

Série d'exercices 5: Les Interruptions

Exercice 1 : Polling vs Interruption (Conceptuel)

Énoncé :
Un capteur de température envoie une donnée toutes les 10 ms. Deux approches sont envisagées :

1. Attente active (polling) d'un flag.
2. Utilisation d'une interruption.

Comparez la charge CPU et la réactivité dans chaque cas si le microcontrôleur doit aussi exécuter une boucle principale complexe

Exercice 2 : Vectorisation des interruptions

Soit un microcontrôleur avec 5 sources d'interruption :

- INT0 (broche externe) → vecteur 0x0008
- INT1 (broche externe) → vecteur 0x0018
- Timer1 overflow → vecteur 0x0020
- UART RX → vecteur 0x0028
- ADC completion → vecteur 0x0030

Écrivez un extrait de code assembleur ou C (avec attributs) plaçant l'adresse de la fonction ISR_Timer1 dans le bon vecteur.

Exercice 3 : Gestion des flags et priorité

Vous avez deux interruptions :

- Interruption A (priorité haute) déclenchée par un bouton poussoir.
- Interruption B (priorité basse) déclenchée par un timer.

L'ISR de B met à jour une variable globale partagée avec la boucle principale.
L'ISR de A modifie la même variable.

Questions :

1. Que se passe-t-il si l'ISR de A interrompt l'ISR de B ?
2. Comment protéger l'accès à la variable ?

Exercice 4 : Configuration d'une interruption externe

Sur un STM32 (ou ARM Cortex-M), configurez la broche PA0 pour générer une interruption sur front montant. Activez l'interruption dans le NVIC. Donnez les registres principaux à configurer.

Exercice 5: Interruption de timer pour échantillonnage périodique

Vous devez échantillonner un capteur sur ADC toutes les 1 ms. Proposez une solution avec un timer qui génère une interruption périodique. L'ISR lance la conversion ADC et stocke le résultat dans un buffer circulaire. La boucle principale traite les données.

Exercice 6: **Bonnes pratiques – ISR ultra-courte**

Énoncé :
Voici une mauvaise ISR. Réécrivez-la correctement.

```
void __interrupt bad_ISR(void) {
   delay_ms(100);              // interdit
   printf("Interruption\n");     // interdit (lent)
   int x = 0;
   for(int i=0; i<10000; i++) {  // trop long
      x += i;
   }
   LED_toggle();               // OK mais perdu dans le long code
}
```

Solution Série 5

Exercice 1 :

- **Polling** : Le CPU vérifie en permanence le flag (ex: while(!flag);). Pendant l'attente, il ne fait rien d'utile → forte charge CPU inutile, mais réactivité immédiate dès que le flag est levé.
- **Interruption** : Le CPU exécute la boucle principale normalement. Quand l'événement survient, l'exécution est suspendue, l'ISR est appelée → charge CPU faible (pas d'attente active), mais réactivité légèrement dépendante de la latence d'interruption.

Conclusion : L'interruption est bien plus efficace pour des événements rares ou périodiques avec des traitements longs en tâche de fond.

Exercice 2 :

Exemple avec GCR AVR :

```
#include <avr/interrupt.h>

// Déclaration de l'ISR pour Timer1 overflow
ISR(TIMER1_OVF_vect) {
    // code court
}

// Le compilateur place automatiquement l'adresse dans le vecteur 0x0020
```

Explication : La vectorisation signifie que chaque source a un emplacement fixe en mémoire (table des vecteurs) pointant vers l'ISR. Le matériel charge directement le PC depuis cet emplacement

Exercice 3 :

1. Si priorités activées, A interrompt B → risque de corruption si la variable est en cours de modification par B (accès non atomique, ex: écriture 16 bits sur 8 bits).
2. Solutions :

    - Désactiver temporairement les interruptions (avant modification dans B) → mais augmente la latence.
    - Utiliser des variables volatile et un mécanisme de verrouillage simple (ex: __disable_irq() / __enable_irq()).
    - Mieux : dans l'ISR de B, ne pas modifier directement la variable partagée mais poster un événement vers la boucle principale (file d'attente).

Le code :

```
volatile int shared_var;
volatile int pending_update;

ISR(TIMER_B) {
```

```
    pending_update = new_value;  // ISR rapide
}

void loop() {
    if (pending_update) {
        __disable_irq();
        shared_var = pending_update;
        __enable_irq();
        pending_update = 0;
    }
}
```

Exercice 4

Solution (pseudo-code) :

```
// 1. Activer l'horloge du GPIOA
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

// 2. Configurer PA0 en entrée
GPIOA->MODER &= ~(3 << 0);  // mode input

// 3. Choisir le front montant (EXTI)
EXTI->RTSR1 |= (1 << 0);     // rising trigger

// 4. Désactiver le front descendant
EXTI->FTSR1 &= ~(1 << 0);

// 5. Activer l'interruption sur ligne 0
EXTI->IMR1 |= (1 << 0);

// 6. NVIC : activer l'IRQ pour EXTI0
NVIC_EnableIRQ(EXTI0_IRQn);
```

ISR correspondante :

```
void EXTI0_IRQHandler(void) {
    if (EXTI->PR1 & (1 << 0)) {  // flag en attente
        // Traitement très court (ex: allumer une LED)
        EXTI->PR1 |= (1 << 0);   // effacer flag
    }
}
```

Exercice 5

```
#define BUFFER_SIZE 100
volatile uint16_t buffer[BUFFER_SIZE];
volatile uint8_t index = 0;

// Configuration timer pour 1 ms (ex: TIM2 sur STM32)
void setup_timer(void) {
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    TIM2->PSC = 7999;    // prés caler pour 1 kHz (exemple)
    TIM2->ARR = 9;       // overflow toutes les 1 ms
    TIM2->DIER |= TIM_DIER_UIE;  // update interrupt enable
    TIM2->CR1 |= TIM_CR1_CEN;
    NVIC_EnableIRQ(TIM2_IRQn);
}

void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_UIF) {
        TIM2->SR &= ~TIM_SR_UIF;
        // Démarrer conversion ADC
        ADC1->CR2 |= ADC_CR2_SWSTART;
        // Attendre fin (polling très court ou interruption ADC)
        while(!(ADC1->SR & ADC_SR_EOC));
        buffer[index] = ADC1->DR;
        index = (index + 1) % BUFFER_SIZE;
    }
}
```

Remarque : Pour respecter la règle « ISR courte », mieux vaut ne pas attendre la fin de l'ADC dans l'ISR. On utiliserait une seconde interruption ADC.

Exercice 6

```
volatile uint8_t event_flag = 0;

void __interrupt good_ISR(void) {
   event_flag = 1;         // flag atomique (1 octet)
   LED_toggle();           // très rapide
}

void main_loop(void) {
   while(1) {
      if (event_flag) {
         event_flag = 0;
         delay_ms(100);          // long traitement ici
         printf("Interruption\n");
         // calculs longs
      }
      // autres tâches
   }
}
```

Explication :

- Pas de fonction lente (printf, delay) dans l'ISR.
- Pas de boucle longue.
- L'ISR ne fait que lever un flag et éventuellement des actions rapides (changer une sortie, incrémenter un compteur).
- Le traitement long est déporté dans la boucle principale (ou un RTOS).