INFO2 : INFORMATIQUE EMBARQUEE
TD8 : Mise en œuvre des interruptions du PIC18F4520

Remarque : Le mécanisme général des interruptions sur PIC18F4520 est décrit dans **`39631E.pdf`** (*datasheet* du PIC18F4520), chapitre 9. La mise en œuvre de ce concept en langage C avec le compilateur C18 est décrite dans **`MPLAB_C18_User_Guide_51288j.pdf`**.

1 Intérêt des interruptions

1.1 Les limites de la scrutation

Jusqu'à maintenant, pour savoir qu'un module périphérique était le siège d'un **évènement**, il fallait le faire par **scrutation (*polling*) d'un drapeau** comme en témoigne le tableau des trois exemples ci-dessous.

Quand … (exemples d'évènement)	**alors …**	**Détection de l'évènement en langage C : c'est quand on quitte la boucle `while` ci-dessous.**
… le timer 0 déborde (fin de comptage),	… le bit **`INTCONbits.TMR0IF`** passe à 1.	`while(INTCONbits.TMR0IF == 0);`

… le CAN a fini sa conversion,	… le bit **`PIR1bits.ADIF`** passe à 1.
	ou bien :
	…le bit **`ADCON0bits.GO`** retombe à 0.
		`while(PIR1bits.ADIF == 0);`
		ou bien :
		`while(ADCON0bits.GO);`

… un caractère a été reçu sur la liaison série,	… le bit **`PIR1.RCIF`** passe à 1.	`while(PIR1bits.RCIF == 0);`

Cette méthode fonctionne très bien mais son gros inconvénient est de sous-employer le microcontrôleur. Tant que le programme est dans la boucle **`while`**, le contrôleur ne fait rien de spécial alors qu'il pourrait vaquer à d'autres occupations (faire de l'affichage par exemple).

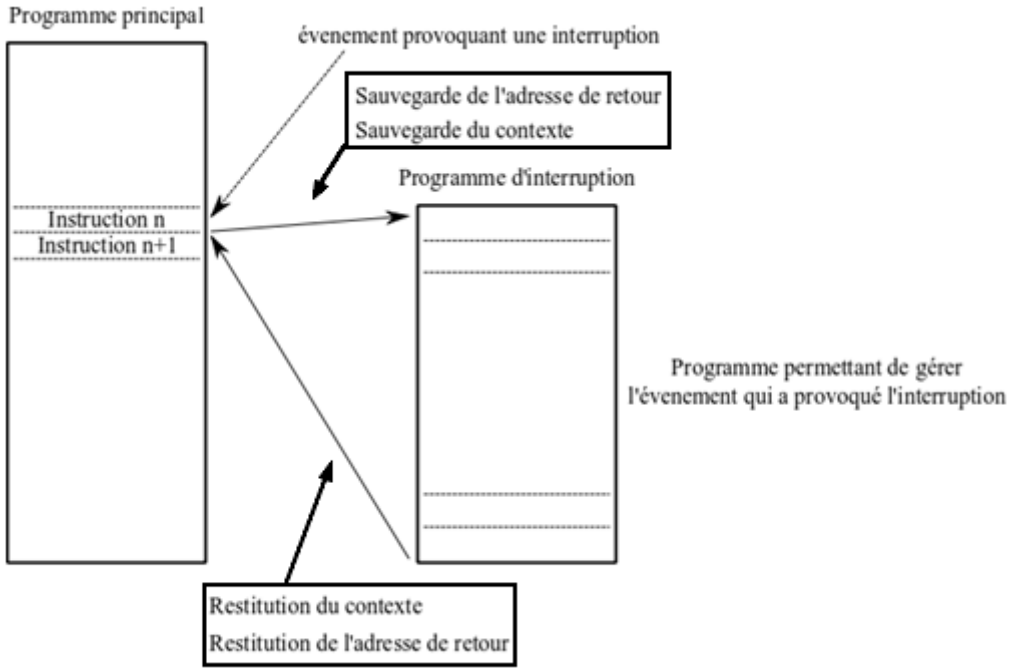
Il existe en outre des disciplines, comme par exemple le filtrage numérique[1], où il faut exécuter une tâche à période fixe (« la **période d'échantillonnage** »). Pour un filtre numérique, cette tâche consiste à calculer la sortie du filtre. Cette périodicité est assurée par un timer. Si cette tâche n'est pas faite au bon moment (par exemple : si on a détecté avec un peu de retard que le timer avait débordé) alors le filtre produit une sortie inadaptée. En effet, celle-ci dépend de l'entrée, des caractéristiques souhaitées du filtre mais aussi de la période d'échantillonnage. Cette dernière doit donc être impérativement respectée et il n'est pas possible de rater le débordement du timer au moment où celui-ci survient.

Pour ces deux considérations, on introduit le concept fondamental d'**interruptions**, à l'œuvre dans de nombreux aspects de l'informatique.

1.2 Qu'est-ce qu'une interruption ?

Une interruption est un évènement qui va détourner le programme de son fonctionnement normal afin d'exécuter un « **programme d'interruption** » (ISR – *Interrupt Service Routine*, c'est une fonction C un peu spéciale à écrire) permettant de gérer l'évènement. Une fois le programme d'interruption fini, le système retourne dans le programme principal à l'instruction suivante.

[1] Les asservissements numériques présentent la même problématique.



Quand l'interruption se produit :

- Le microcontrôleur empêche les interruptions de niveau inférieur ou égal à celle en cours (voir un peu plus bas pour cette notion de niveau d'interruption).
- Le microcontrôleur finit l'instruction n en cours (l'instruction en assembleur, pas celle en C !).
- Il sauvegarde dans la pile (empilement) l'adresse de l'instruction n+1 du programme principal (**adresse de retour**).
- Il sauvegarde dans la pile (empilement) le **contexte** :
    - le registre de travail **`W`**,
    - le registre d'état **`STATUS`**,
    - le registre **`BSR`** (*Bank Shift Register*) qui spécifie sur laquelle des banques de 256 octets de la mémoire de donnée on est en train de travailler.
- Il exécute le programme lié au périphérique demandant l'interruption.
- Il restitue le contexte (dépilement).
- Il exploite l'adresse de retour (dépilement) pour retourner au programme principal à l'adresse de l'instruction n+1.
- Le microcontrôleur réautorise les interruptions de niveau inférieur ou égal à celle qui vient de se terminer.

Remarque : Les sauvegardes de registres et d'adresse de retour sont effectuées dans une zone particulière de la mémoire appelé la **pile** (*stack*) : Cf. FIGURE 5-1 de **`39631E.pdf`** (*datasheet* du PIC18F4520), reproduite à la partie **2.3.1**.

Remarque : Cette pile sert aussi lors de l'appel de fonctions en C afin de sauvegarder l'adresse de retour, passer d'éventuels paramètres à la fonction et récupérer une éventuelle valeur retournée par la fonction.

2 Mise en œuvre des interruptions en langage C (compilateur C18)

Remarque sur les priorités d'interruption : Les PIC18F possèdent **deux niveaux d'interruption**, c'est-à-dire qu'il est possible de rendre une interruption **prioritaire** par rapport à une autre. Ainsi par exemple, si l'on est dans l'ISR de réception d'un caractère et qu'un timer déborde à ce moment-là et qu'il faut traiter impérativement cet évènement, le débordement du timer va interrompre l'ISR de réception du caractère à condition d'avoir dit préalablement que l'interruption du timer était de **priorité haute** et que celle de réception d'un caractère était de **priorité basse**. Dans le cadre de cette présentation, nous n'exploiterons pas cette possibilité (toutes les interruptions seront de priorité haute) rendant ainsi le système interruptif calqué sur celui de la famille PIC16F. Pour information, il y a plus de 16 niveaux d'interruptions dans un ordinateur de type PC.

2.1 Les bits individuels d'autorisation d'interruption et les flags de témoin d'interruption

Le PIC18F4520 dispose de 20 sources d'interruptions. Chaque source d'interruption possède :

- un bit permettant de l'autoriser individuellement. Son nom se termine par IE (*Interrupt Enable*), par exemple **`INTCONbits.TMR0IE`**,
- un *flag* (un bit) témoignant de la demande d'interruption[2]. Son nom se termine par IF (*Interrupt Flag*), par exemple **`INTCONbits.TMR0IF`**.

Ces 40 bits sont accessibles dans les registres **`INTCON`**, **`INTCON3`**, **`PIE1`**, **`PIE2`**, **`PIR1`** et **`PIR2`** : Cf. pages 93 à 99 de **`39631E.pdf`** (*datasheet* du PIC18F4520). Une synthèse est donnée ci-dessous :

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	49, 93
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	49, 95
PIR2	OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	00-0 0000	52, 97
PIE2	OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	00-0 0000	52, 99
PIR1	PSPIF(2)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	52, 96
PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	52, 98

Remarque : Au *reset*, tous ces bits sont placés à zéro (sauf le flag **`RBIF`**) ce qui implique qu'aucune interruption n'est validée et qu'aucun *flag* n'est levé.

1. Compléter le tableau ci-dessous qui recense une partie des sources d'interruptions :

	Périphérique	**Bit d'autorisation de l'int.**		***Flag* témoin**	
		nom du bit	**nom du registre**	**nom du bit**	**nom du registre**
1	Timer 0	`TMR0IE`	`INTCON`	`TMR0IF`	`INTCON`
2	Timer 1				
3	Timer 2				
4	INT0				
5	PORTB				
6	CAN (convertisseur analogique)				
7	UART (RS232 réception)				
8	UART (RS232 transmission)				
9	Port parallèle (module SSP)				
10	Transmission série de type I2C ou SPI				
11	Collision de BUS				
12	Ecriture dans la mémoire EEPROM				
14	Module CCP1				
15	Module CCP2				

[2] *interruption request*

2.2 Bit GIE d'autorisation globale des interruptions

Après avoir autorisé ou pas individuellement les interruptions, il faut régler un bit d'autorisation globale de toutes les interruptions : *Global Interrupt Enable bit* (**GIE**) du registre **INTCON**.

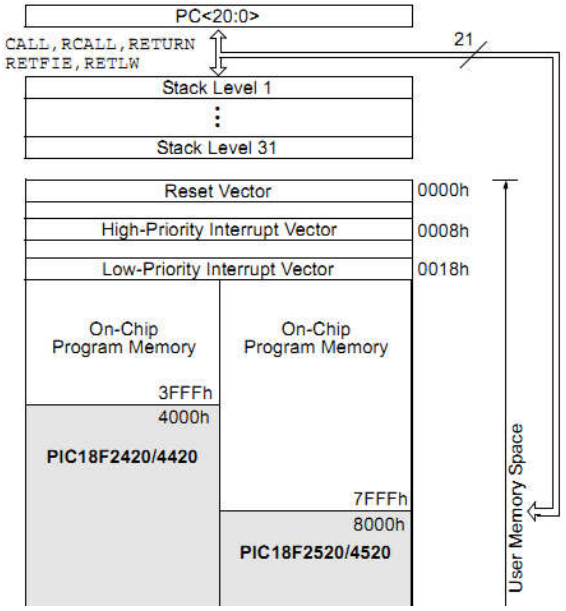
2. Quelle est la valeur au *reset* du bit **GIE** ? Les interruptions sont-elles autorisées au *reset* ?

2.3 Que se passe-t-il lorsqu'une demande d'interruption survient ?

2.3.1 Vecteur d'interruption

Comme pour le *reset* où le PIC se dirige automatiquement vers l'adresse 0x0000 de la mémoire de programme, le PIC se dirige automatiquement vers l'adresse 0x0008 de la mémoire de programme lorsqu'une interruption de priorité haute survient (et qu'elle est autorisée). Cette adresse s'appelle **vecteur d'interruption** :

FIGURE 5-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F2420/2520/4420/4520 DEVICES



Remarque : En général, on n'écrit pas l'ISR à partir de l'adresse 0x0008 mais une brève fonction de saut vers une fonction implémentant l'ISR. En effet, on voit sur la cartographie ci-dessous que 0x0018 correspond au vecteur d'interruption des priorités basses. Il ne faut surtout pas qu'en 0x0008 on ait écrit une fonction trop longue qui comporte encore des instructions en 0x0018 !

2.3.2 Sans gestion des priorités d'interruption, une interruption ne peut interrompre une ISR en cours

Lorsqu'une interruption autorisée survient, le système interruptif du PIC met automatiquement le bit **GIE** à 0, **empêchant ainsi une éventuelle interruption d'interrompre l'ISR de l'interruption en cours**. Quand l'ISR se termine, ceci est traduit en assembleur par l'instruction **RETFIE** (*REturn From IntErrupt*). Cette instruction a pour effet, outre la restitution de l'adresse de retour et du contexte, de repositionner le bit **GIE** à 1, autorisant les nouvelles interruptions à venir.

Remarque : La fin d'une fonction C se traduit par l'instruction assembleur **RETURN**. Cette instruction restitue l'adresse de retour mais ne gère pas le bit **GIE** à la différence de l'instruction **RETFIE**.

2.3.3 Que faut-il écrire dans l'ISR ?

Quand on arrive dans l'ISR, c'est qu'une interruption a eu lieu. Il convient d'abord de chercher quel module périphérique l'a provoquée. En effet, vus nos choix, il n'y a qu'un seul vecteur pour toutes les sources d'interruption (0x0008).

Attention : Une interruption étant un phénomène complétement **asynchrone** (on ne sait pas quand elle arrive), la fonction C implémentant l'ISR ne peut pas admettre de paramètres ni retourner de valeur de sortie.

Un exemple d'implémentation d'ISR est donc :

```
void ISR(void) {
    if (INTCONbits.TMR0IF) { // débordement du timer 0 ?
        // Tâches à effectuer suite au débordement du timer 0 à écrire ici
        bla bla bla
        // Puis remise à 0 du flag TMR0IF :
        INTCONbits.TMR0IF = 0;
    }
    if (PIR1bits.RCIF) {    // caractère reçu sur l'UART ?
       // Tâches à effectuer suite à la réception d'un caractère sur l'UART à écrire ici
       // Puis ne pas oublier de remettre à 0 le drapeau RCIF
    }
    etc.
}
```

3. À quoi sert la remise à 0 du flag **`TMR0IF`** ? Quelle instruction équivalente doit être écrite au niveau de la réception d'un caractère ?

3 Cahier des charges

Le programme principal consiste à numériser en continu un signal analogique comme dans le TD6.

En outre, toutes les 200 ms, la led connectée à la broche RD0 change d'état par un mécanisme d'interruption (timer 0, quartz à 8 MHz).

4. Quelle durée maximum peut durer l'ISR si on ne veut pas rater d'interruption ?
5. Écrivez le programme permettant de respecter le cahier des charges.