Les fonctions et les interruptions du PIC16F877

cours n°3
LI326

Directive CBLOCK

Dans le fichier picmips.inc

```
variable BANK0 = 0x20       ; first free byte in bank0
variable BANK1 = 0xA0       ; first free byte in bank1
variable BANK2 = 0x110      ; first free byte in bank2
variable BANK3 = 0x190      ; first free byte in bank3
variable BANK4 = 0x70       ; first free byte in shared bank

CBLOCK BANK0
      _W0                   ; working register
      _W1                   ; working register
      _M1BAK                ; macro register backup
      _WBAK                 ; W     backup
      _PCLATHBAK            ; PCLATH backup
ENDC
variable BANK0 = _PCLATHBAK+1

CBLOCK BANK4
      _M1                   ; macro register
      _STATUSBAK            ; STATUS backup
      A0                    ; arguments et resultat
      A1                    ; des fonctions
      A2                    ; ...
      A3                    ; ...
ENDC
variable BANK4 = A3+1
```

Dans votre fichier .asm

Pour réserver un registre de 1 octet dans le bank0

```
CBLOCK BANK0
      var1
      var2
ENDC
variable BANK0 = var2+1
```

Si on veut réserver plusieurs octets

```
CBLOCK BANK1
      tab   : 8         ; réserve 8 octets
      var3
ENDC
variable BANK1 = var3+1
```

L'adresse de var3 est tab+8

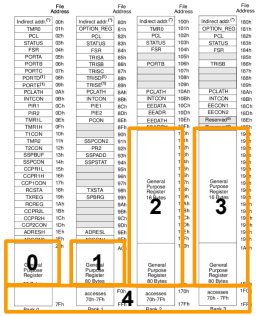
Nommage des registres

On a vu que les registres spéciaux (SFR) étaient associés à des symboles dans le fichier p16f877.inc grace à la directive **EQU**

```
STATUSEQU     H'03'
```

Pour les registres spéciaux, nous allons distinguer 2 cas:
Les registres n'ayant qu'une adresse

BANK0 de 0x20 à 0x6F
BANK1 de 0xA0 à 0xEF,
BANK2 de 0x110 à 0x16F
BANK3 de 0x190 à 0x1EF



Les registres partagés ayant 4 adresses (n'utilisent donc pas les bits RP0 et RP1)

BANK4 de 0x70 à 0x7F
de 0xA0 à 0xFF
de 0x170 à 0x17F
de 0x1F0 à 0x1FF

Ces registres vont être utilisés pour le passage des paramètres aux fonctions parce qu'on n'a pas à se soucier de la valeur des bits RP0 RP1

Définition d'une fonction *(wikipedia)*

En informatique, une **fonction** est une portion de code représentant un sous-programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme et qui renvoie une valeur (elle se distingue ainsi de l'instruction).

[...]

Anatomie d'une fonction

Éléments constitutifs d'une fonction :

- Nom de la fonction ➜ symbole
- Noms ~~et/ou types~~ des paramètres ➜ par W ou les registres partagés
- ~~Type de la valeur de retour~~
- ~~Délimitation du bloc de code de la fonction~~
- Définition de la valeur de retour ➜ dans W ou les registres partagés
- Les variables locales ➜ dans les registres standards

Illustration sur un exemple

La moyenne de deux nombres

```
char moy (char a0, char a1) {
    return (a0+a1) / 2;
}
```

On suppose que les registres a0 et a1
ont été remplis avec les arguments.
Le résultat sera mis dans a0

```
f_moy:  movf    a1,w
        addwf   a0,f
        bcf     STATUS,c
        rrf     a0,f
        return
```

Pour l'usage si on veut faire la moyenne
de v0 et v1 dans v2 deux registres du BANK1

```
CBLOCK BANK1
    v0
    v1
    v2
ENDC
variable BANK1=v1+1
```

```
; v2 = moy(v0,v1)
movf    v0,w
movwf   a0
movf    v1,w
movwf   a1
call    f_moy
movf    a0,w
movwf   v2
```

Pour améliorer la lisibilité on peut gérer
le passage des arguments par macro.

```
moy macro @d, @s0, @s1
        movf    @s0,w
        movwf   a0
        movf    @s1,w
        movwf   a1
        call    f_moy
        movf    a0,w
        movwf   @d
    endm
```

AInsi on écrit

```
moy v2, v1, v0
```

Excercice pour comprendre

Ecrire la fonction qui fait un décalage à gauche parametrable encapsulé dans une macro de telle sorte que nous aurons le prototype suivant:

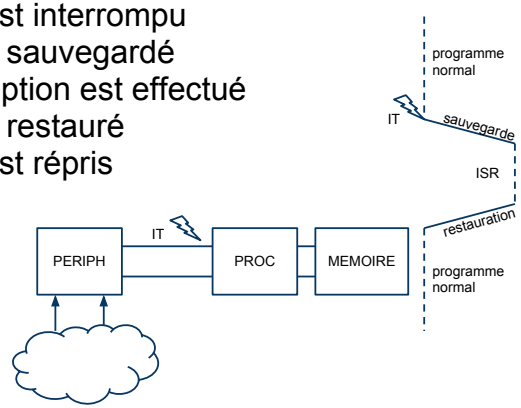
```
sllv r2, r1, r0          ; r2 = r1 << r0  (r2, r1, r0 sont des registres quelquonque)

sllv macro   @d, @s1, @s0                    f_sllv    ; a0 = a1 << a0
```

Qu'est-ce qu'une interruption

Une interruption est une demande d'un périphérique matériel de traitement d'un événement par le processeur.

- Le programme normal est interrompu
- L'état du processeur est sauvegardé
- Le traitement de l'interruption est effectué
- L'état du processeur est restauré
- Le programme normal est répris

Le périphérique a "volé" des cycles au programme normal



Acquittement d'une interruption

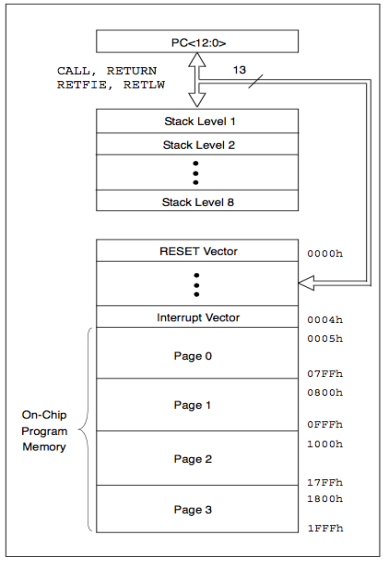
- Le programme normal n'est pas toujours interruptible. Il peut être en train de modifier l'état de ses données et demander au processeur de différer le traitement des interruptions.
- Un périphérique doit maintenir sa ligne d'interruption jusqu'à ce que le traitement demandé soit réalisé.
- Le processeur peut "masquer" ses lignes d'interruptions.
- Lors du traitement d'une interruption, il faut acquitter l'interruption en écrivant dans le périphérique pour lui demander de "baisser" sa ligne.

Routine d'interruption : ISR

- Le traitement d'une interruption est effectué par une ISR
  Interrupt Service Routine.
- En général, on a autant d'ISR que de type d'interruption.
- Un périphérique peut produire plusieurs types d'interruption.
  par exemple: le terminal peut lever une interruption si une touche est tapé au clavier et une interruption si l'écran attend un nouveau caractère à afficher.
- En général, l'objectif d'une ISR est de lire ou d'écrire des donnée dans le périphérique concerné, puis d'acquitter l'interruption pour que le périphérique baisse la ligne.

Cas du p16f877 : Vecteurs

- Le vecteur d'interruption est : 4 pour les interruptions normales
- Le vecteur du reset est 0
  Le reset peut être considéré comme une interruption non masquable.
- L'adresse de retour est copié dans la pile des adresses.

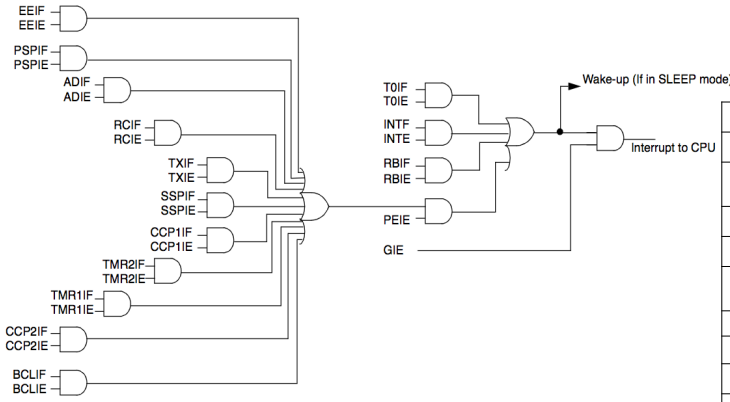


Gestionnaire d'interruption

- Le morceau de programme qui gère le déroutement du programme normal vers les routines ISR se nomme: gestionnaire d'interruption (interrupt handler).
- Le gestionnaire fait 4 choses.
  1. Il sauve le contexte pour ne pas perturber le programme interrompu (il ne doit se rendre compte de rien)
  2. Il analyse la cause d'interruption pour choisir la bonne ISR
  3. Il appelle l'ISR
  4. Il restaure le contexte
- L'exécution du gestionnaire est toujours faîte avec les interruptions masquées.
- L'adresse du gestionnaire est nommé vecteur d'interruption

Cas du p16f877 : Sources

Le p16f877 dispose de 14 sources d'interruptions.

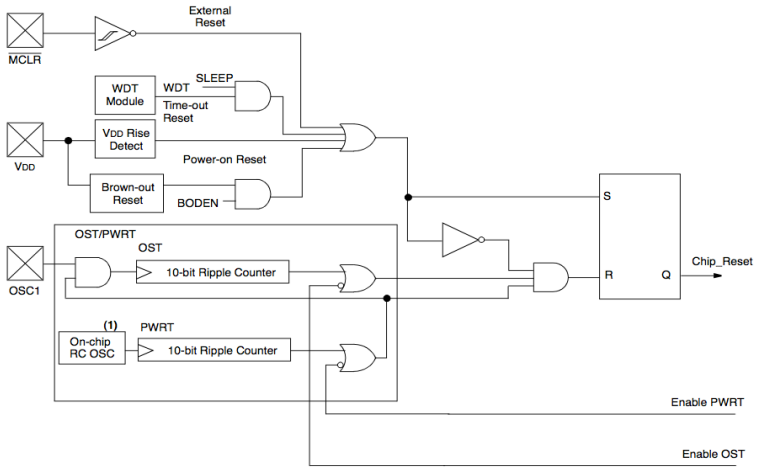


Pour chaque source, il y a 2 bits de contrôle : IF et IE
Il y a 2 masques globaux : GIE et PIEE

T0	Timer 0
INT	Broche d'interruption
RB	Changement d'état des broches
EE	EEprom
PSP	Port parallèle
AD	Convertisseurs analogique
RC	Entrée série
TX	Sortie série
SSP	I2C / SPI
CCP	Capture 1
TMR2	Timer 2
TMR1	Timer 1
CCP2	Capture 2
BCL	Collision de bus

Cas du p16f877 : Source du reset

4 sources de reset qui peuvent se gérer comme les interruptions mais on ne va pas écrire de gestionnaire de reset en TME



Sauvegarde et restauration

Les registres à sauver au minimum sont :

- W, STATUS et PCLATH : obligatoire
- FSR : optionnel

```
MOVWF     W_TEMP            ;Copy W to TEMP register
SWAPF     STATUS,W          ;Swap status to be saved into W
CLRF      STATUS            ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
MOVWF     STATUS_TEMP       ;Save status to bank zero STATUS_TEMP register
MOVF      PCLATH, W         ;Only required if using pages 1, 2 and/or 3
MOVWF     PCLATH_TEMP       ;Save PCLATH into W
CLRF      PCLATH            ;Page zero, regardless of current page
:
:(ISR)                      ;(Insert user code here)
:
MOVF      PCLATH_TEMP, W    ;Restore PCLATH
MOVWF     PCLATH            ;Move W into PCLATH
SWAPF     STATUS_TEMP,W     ;Swap STATUS_TEMP register into W
                            ;(sets bank to original state)
MOVWF     STATUS            ;Move W into STATUS register
SWAPF     W_TEMP,F          ;Swap W_TEMP
SWAPF     W_TEMP,W          ;Swap W_TEMP into W
```

- La sortie du gestionnaire est : retfie

Cas du p16f877 : Registres de contrôle

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h		190h	
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h	General Purpose Register 16 Bytes	197h	General Purpose Register 16 Bytes
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh / F0h	accesses 70h-7Fh	16Fh / 170h	accesses 70h-7Fh	1EFh / 1F0h	accesses 70h - 7Fh
		FFh		17Fh		1FFh	

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

PIR2 REGISTER (ADDRESS 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

Analyse de la cause

- L'analyse de la cause consiste à lire les drapeaux non masqués des périphériques susceptibles de lever une interruption et d'appeler les ISR correspondantes.
- Le plus simple est que les ISR soient interruptibles donc courtes.
- L'ISR doit toujours acquitter l'interruption en : baissant le drapeau ou en utilisant le périphérique
- On peut ne tester qu'une seule cause en faisant l'hypothèse que les interruptions sont courtes et ne sont pas simultanées généralement.
- Les ISR doivent utiliser des registres propres pour ne pas écraser des données du programme interrompu.

Routine d'interruption : ISR

- La durée d'une ISR doit être bornée puisque non interruptible.
- L'acquittement de l'interruption dépend du périphérique:
  - pour le timer 0, il faut mettre le bit T0IF à 0.
  - pour l'entrée du port série, il faut lire la donnée reçue.
- L'echange avec le programme principal se fait par des canaux de communications simples constitués de :
  - un buffer : ça peut être un registre ou plusieurs
  - un drapeau : un bit dans un registre réservé pour ça
- Les canaux de communications sont nommés des événements

Exemple d'échange : sortie écran

- On suppose que l'on a un périphérique de sortie fonctionnant de la manière suivante:
  - on l'initialise pour définir, par exemple, le débit.
  - on dispose d'un registre de sortie tel qu'une écriture dans ce registre provoque l'émission de la donnée écrite à faible débit.
  - Une interruption est levée lorsque le registre d'échange est vide.
- On veut envoyer un message complet:
  - on réserve un buffer de 16 caractères (p.ex.)
  - on reserve une variable index qui indique le n° du caractère à envoyer
  - on reserve un bit drapeau initialisé à 0
    - BUFFER[16]
    - INDEX
    - FLAG

Echanges par événement

- Les événements sont un couple drapeau + buffer
  - Le drapeau permet de connaitre l'état du buffer: p.ex. 0/1 (vide/plein)
  - Le buffer permet de transporter de l'information
- Un événement permet à deux *"tâches"* de communiquer: ici les tâches sont: les ISR et le programme principale main.
- L'état du drapeau permet de savoir qui a le droit d'utiliser le buffer entre l'ISR et le main.

Exemple d'échange : sortie écran

Le périphérique écran est controlé grâce à 3 registres
- IE_ecran
- IF_ecran
- data_ecran

```
main
   tant que (FLAG !=0);
   copy (BUFFER, "hello");
   INDEX = 0;
   DEMASQUER IE_ecran ;
   FLAG = 1;
```

```
isr
   c = BUFFER[INDEX] ;
   si c == 0 alors
      MASQUER IE_ecran ;
      FLAG = 0;
   sinon
      INDEX++ ;
      data_ecran = c ;
   fsi
```

On suppose que l'écrire dans data_ecran acquite IF_ecran

Forme général du programme

Nous allons vous proposer une forme général pour les programmes qui vous guidera pour l'ensemble de TME

```
    org 0
    goto init

    org 4
    SAVE_CONTEXT
    ANALYSE_CAUSE
        call isr1
    REST_CONTEXT
    retfie

init
    PROGRAM_PERIPH
    DEMASQUE_PERIPH
    bsf    INTCON,PEIE
    bsf    INTCON,GIE
    goto main
```

```
isr1
    TRAITEMENT
    ACQUITEMENT
    return

isr2
    TRAITEMENT
    ACQUITEMENT
    return

main
    BOUCLE_SANS_FIN
```

un parfum de multi-tâches

Le programme principal peut être décrit comme une super-boucle qui appelle à tour de rôle des tâches. Chaque tâche traite un événement.

```
main
   call tache1
   call tache2
   ....
   call tache3
   goto main
```

```
tache1
   si drapeau event1 != 1
      return
   usage du buffer
   drapeau event1 = 0
   return
```

Echange avec ou sans perte

- L'échange décrit dans l'exemple est un échange sans perte.Toutes les données écrites par le main (l'écrivain du buffer) sont envoyées par l'ISR (le lecteur du buffer).
- On peut aussi décrire un échange à perte, il suffit que l'écrivain ne tiennent pas compte de l'état du drapeau pour reremplir le buffer.
- C'est utilise lorsque les données à écrire ont une durée de validité limitée

Cas particulier du timer

- Les tâches d'un micro-contrôleur sont souvent contrôlées par le temps. Par exemple il faut lire les boutons toutes les 20ms ou prendre la température toutes les minutes...
- On peut utiliser un timer qui va être programmer pour définir une base de temps périodique (p. ex. 10ms) et programmer des timers multiples de cette période dans l'ISR du timer.
- Pour chaque période dérivée, il y aura un drapeau.
  Ce drapeau est levé par l'ISR du timer et baissé par la tâche en attente de la période.