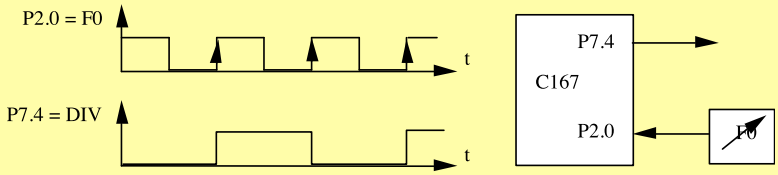
Chapitre 5 -
Le fonctionnement en régime d'interruption de programme

➡ Intérêt des interruptions

Exemple : la division par deux de la fréquence d'un signal externe F0.

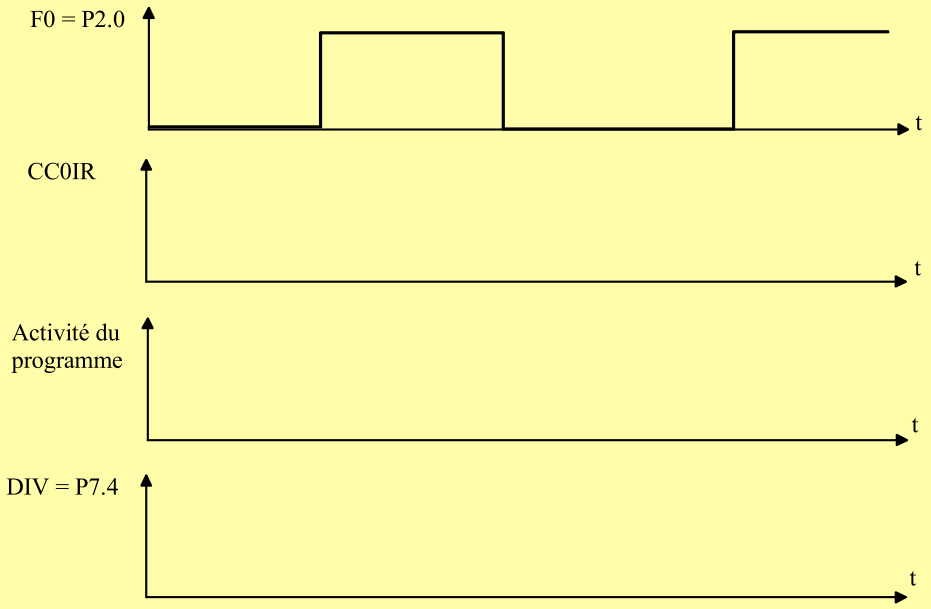


Programme proposé

```
void Init_ES (void)
{
  DP7 = 0x10 ;
  P7 = 0;
  CCM0 = 1 ;  // front montant sur P2.0
}
```

```
sbit        DIV = P7^4 ;
void main (void)
{
  Init_ES();
  while (1) {
     while ( ! CC0IR) ;
     CC0IR = 0 ;
     DIV = ~ DIV ;
  }
}
```

Déroulement temporel de la division de fréquence par 2 (après initialisation)



➡ Que conclure ?

Le programme passe son temps à tourner dans une boucle d'attente !

- ➡ Il serait plus judicieux de décharger le microprocesseur de ce test et de faire en sorte que ce soit **le circuit d'E/S qui prévienne le microprocesseur**, par un signal logique, lorsque la transition est détectée.
- ➡ A ce moment, il faudrait que le **microprocesseur interrompe ce qu'il est en train de faire** pour traiter la détection de la transition.

De ce fait :

- ➡ Le **déroulement** temporel des programmes devient, en partie, **guidé** par **l'environnement extérieur** du microprocesseur.
- ➡ On récupère les temps d'attente précédents pour **faire travailler** le microprocesseur à **d'autres tâches**.

➡ Deux questions

- Quelle est la **structure matérielle** permettant au microprocesseur de prendre en compte de telles demandes externes ?
- Quelle est la **structure** correspondante du **logiciel** ?

Remarque :

Un signal externe demandant "l'attention" du microprocesseur s'appelle :

- un signal d'interruption ou
- une demande d'interruption de programme ou encore
- une requête externe.

Par simplification, on la note **Demande d'IT** ou plus simplement **IT**.

➡ Eléments de base

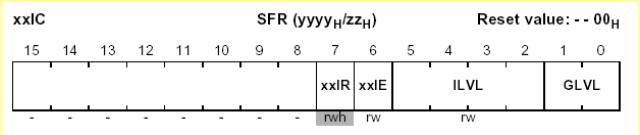
5.2.1. Les éléments de base (1)

L'origine des demandes d'interruption est :

- soit externe, par les lignes _NMI et EX0IN (P2.8) à EX7IN (P2.15)(non étudié ici)
- soit interne, par les nombreux éléments intégrés du microcontrôleur
  - entrées de capture, sorties de comparaison
  - liaisons séries, liaisons parallèles
  - PWM, etc…

Il y a **56 sources internes** de demandes d'interruptions dans le C167.

A chaque source est associé un registre de contrôle xxIC, par exemple CC0IC pour le canal 0 de CAPCOM1 :



➡ Champs de xxIC : exemple avec CC0IC

5.2.1. Les éléments de base (2)

CC0IC (FF78h/BCh)			SFR				Reset : 00h
7	6	5	4	3	2	1	0
CC0IR	CC0IE	ILVL				GLVL	
rw	rw	rw				rw	

CC0IR : CC0 Interrupt **Request** : bit (si à 1) indiquant une détection, soit d'une transition active (mode capture), soit d'une comparaison réussie (mode comparaison)

CC0IE : CC0 Interrupt **Enable** : bit jouant le rôle d'un masque local, autorisant (si à 1) la génération d'une requête d'interruption lorsque le bit CC0IR passe à 1.

ILVL : Interupt **Level** : C'est le niveau de priorité de l'interruption. Ceci est très important lorsqu'il y a des requêtes simultanées, car on peut traiter qu'une interruption à la fois.
ILVL est un champ de 4 bits
16 niveaux d'importance possibles pour les 56 IT (0 : le plus faible, 15 : le plus fort)

GLVL : **Group Level**
16 niveaux seulement — notion de groupe pour définir l'ordre interne à un niveau
(0 : niveau le plus faible, 3 : niveau le plus fort)

ILVL + GLVL donnent 64 combinaisons possibles pour 56 demandes (48 sont utilisables)

Toute les interruptions doivent avoir une combinaison (ILVL, GLVL) différente.

➡ Vecteur d'interruption

5.2.1. Les éléments de base (3)

A chaque interruption est associée une adresse mémoire (en fait 4 octets) dans le segment 0 :

le vecteur d'interruption

A chaque interruption est associé un numéro de « trappe » :

le Trap Number

Ce numéro multiplié par 4 donne l'adresse mémoire du vecteur d'interruption.

Tableau des paramètres pour quelques sources internes

Source	*Drapeau*	*Masque*	*Registre*	*adresse du vecteur*	*N° de trappe*
Canal 0 CAPCOM1	**CC0IR**	CC0IE	CC0IC	**0x00'0040**	**16**
Canal 4 CAPCOM1	CC4IR	CC4IE	CC4IC	0x00'0050	20
Canal 12 CAPCOM1	CC12IR	CC12IE	CC12IC	0x00'0070	28
Canal 13 CAPCOM1	CC13IR	CC13IE	CC13IC	0x00'0074	29
Timer 0	**T0IR**	T0IE	T0IC	**0x00'0080**	**32**
Timer 1	T1IR	T1IE	T1IC	0x00'0084	33

5.2.1. Les éléments de base (4)

➡ Le registre PSW (Processor Status Word)

PSW
Processor Status Word SFR ($FF10_H/88_H$) Reset value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	-	-	-	USR 0	MUL IP	E	Z	V	C	N
rw				rw	rw	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

IEN : Interrupt **Enable** (si à 1)

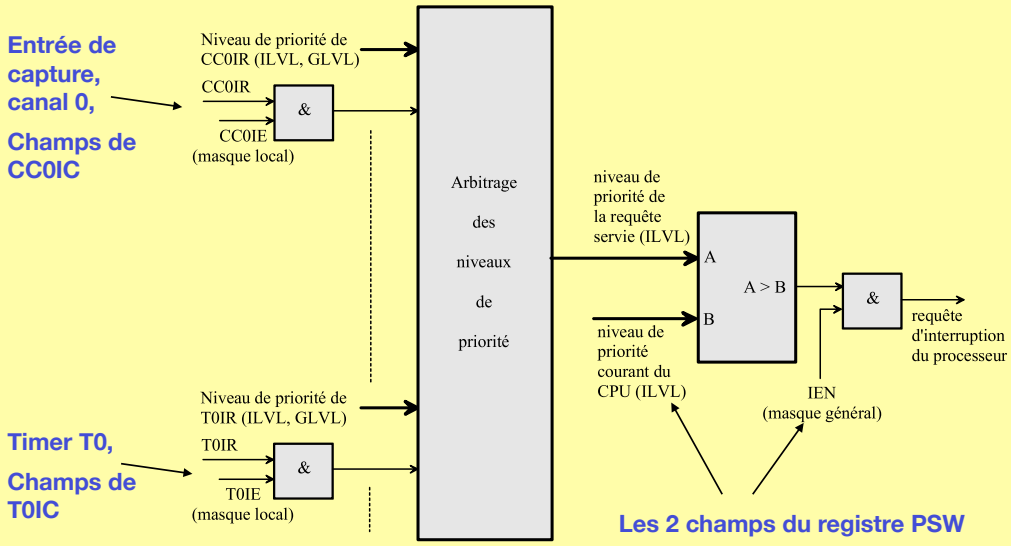
Masque global pour toutes les interruptions. Les autorise (si à 1) ou non.

ILVL : Interrup **Level**

Niveau courant d'interruption pour le processeur

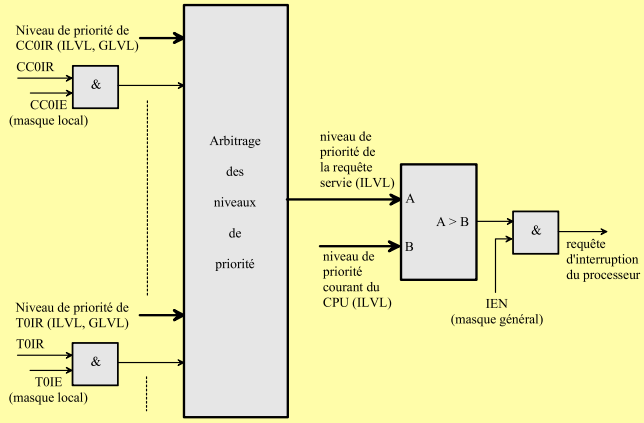
5.2.2. Cheminement requête (1)

➡ Cheminement d'une requête



Comment une requête progresse-t-elle ?

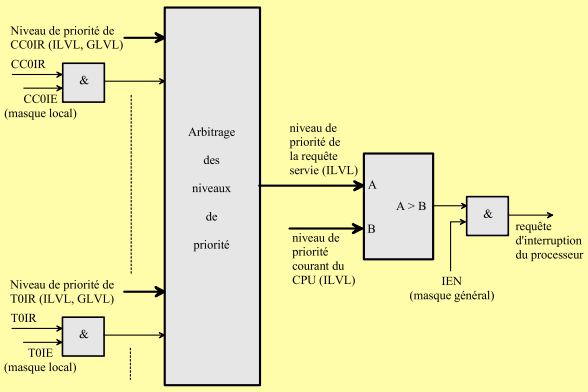
5.2.2. Cheminement requête (2)



Une requête (par exemple CC0IR à 1) ne peut progresser que si :

- elle est autorisée à engendrer une requête (masque local CC0IE à 1)
- son niveau d'interruption (ILVL dans CC0IC) est strictement supérieur à celui de toute autre requête présente ou bien s'il est égal à celui d'autres requêtes présentes, son niveau de groupe (GLVL dans CC0IC) est supérieur à celui des autres requêtes de même niveau
- son niveau d'interruption (ILVL) est strictement supérieur à celui du CPU (champ ILVL de PSW)
- les interruptions sont autorisées (masque général IEN à 1)

5.2.2. Cheminement requête (3)



Remarques :

- Sur **Reset**, PSW est mis à 0 => $(ILVL)_{CPU} = 0$
  => Une requête avec un niveau de priorité 0 n'est jamais prise en compte
  => Les interruptions ne sont pas acceptées (IEN à 0)
- Les **niveaux 14 et 15** sont utilisés pour un dispositif spécifique : le PEC (Peripheral Event Controller).
  => **utiliser** seulement les **niveaux 1 à 13** pour hiérarchiser vos requêtes
- Le programme peut contrôler, à tout moment, les demandes d'interruption qu'il veut accepter en fixant la valeur de ILVL dans PSW

5.2.3. Acceptation d'une requête

① Le processeur **termine** toujours **l'instruction** en cours (sauf MUL/DIV, voir plus loin)

② **Sauvegarde** sur la pile de PSW, puis de CSP (Code Segment Pointer) si on est en mode segmenté, puis IP (Instruction Pointer). On a ainsi sauvegardé le **contexte minimal** du CPU :

- le registre d'état de la machine (PSW)
- l'adresse de la prochaine instruction à exécuter (IP + CSP)

Ce sont les informations minimales pour pouvoir reprendre le traitement après l'interruption (de programme) ;

③ **Mise à jour** de PSW (champ ILVL) par le niveau d'interruption actuellement servi.

=> seules les requêtes de priorité supérieures à celle en cours pourront interrompre le processeur

④ **Remise à zéro** de la requête servie : le bit xxIR est **automatiquement** remis à 0 ;

⑤ **Chargement** dans le registre **IP** du vecteur d'interruption (n° de trappe x par 4) ;
Remise à zéro de CSP si on est en mode segmenté (les vecteurs sont dans le segment 0) ;

⑥ A l'adresse du vecteur d'interruption, l'utilisateur met en principe une instruction de saut (jmp cc_UC,…) à la routine de traitement de l'interruption :

=> **exécution** de ce saut et **le programme se continue dans la routine d'IT**

5.2.4. Format routine de traitement (1)

Forme générale valable pour une programmation en assembleur

Point d'entrée:

sauvegarde du contexte	**fait automatiquement par le compilateur C**
traitement de l'IT	**sera le corps de votre fonction**
restitution du contexte	**fait automatiquement par le compilateur C**
reti	**traduit la parenthèse fermante de la fonction (})**

➡ **Sauvegarde du contexte**

Pourquoi ? Si la routine fait des calculs utilisant les registres, ceux-ci vont être détruits (les valeurs qu'il y avait dans le programme interrompu vont être perdues)

=> il faut les sauvegarder avant de les utiliser

➡ **Restitution du contexte**

Pourquoi ? il faut rendre au programme interrompu les valeurs qu'il y avait dans les registres avant son interruption

➡ **reti : Return from Interrupt** (instruction assembleur)

Récupération dans la pile des valeurs de PSW, (CSP) et IP pour revenir au point d'interruption dans le programme interrompu : c'est la **restitution du contexte minimal** sauvegardé en acceptant l'interruption.

5.2.4. Format routine de traitement (2)

➡ **Format en C d'une routine d'interruption**

Obligatoire car c'est le matériel qui active, donc pas de paramètres

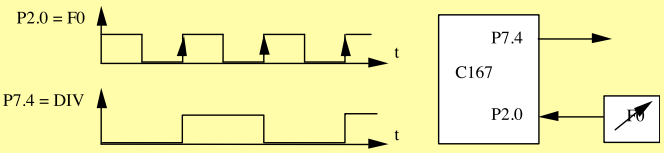
```
void  function_name (void)   interrupt   trap_number
```

Obligatoire car je ne peux rien renvoyer

Indique au compilateur que c'est une procédure d'interruption

Le n° de trappe de la requête d'interruption

Exemple pour le diviseur par 2




```
sbit        DIV= P7^4 ;

void Diviseur (void)  interrupt  16
{
        DIV = ~ DIV ;
}
```

Le n° de trappe de CC0IR est 16

A chaque front on bascule la sortie, puis on quitte la procédure

Sera traduit par « reti » pour revenir au programme interrompu

De plus le compilateur a généré l'instruction de saut qui sera placée, lors du chargement, à l'adresse 0x0040 (4 fois 16 en hexa), afin d'activer automatiquement la procédure.

5.2.4. Format routine de traitement (3)

➡ **Structure en C du programme principal**

```
void main (void)
{
    Initialisation_diverses (ports,timer) ;     <=   Ceci est hors interruptions
    Initialisations des interruptions;          <=   la validation des interruptions
    Travail de "fond" ;                         <=   Ceci peut être interrompu
}
```

Exemple pour le diviseur par 2

```
void Init_ES (void)
{
  DP7= 0x10 ;
   P7 = 0;
   CCM0 = 1 ;
}
```

```
void Diviseur (void)  interrupt  16
{
   DIV = ~ DIV ;
}
```

ILVL = 4 GLVL = 0

```
void main (void)
{
   Init_ES();
   CC0IC = 0x10 ;  // 0001 0000
   CC0IE = 1 ;       // masque local à CC0
   IEN = 1 ;         // masque général
   while (1)   {
       tache_de_fond ()
   }
}
```

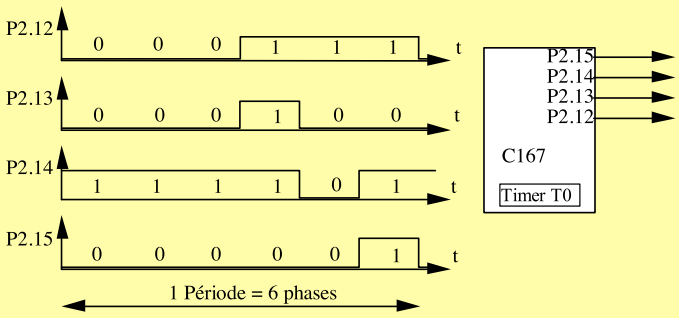
Interruptions, vecteurs, n° de trappes

5.2.4. Format routine de traitement (4)

Source	*Drapeau*	*Masque*	*Registre*	*adresse du vecteur*	*N° de trappe*
Canal 0 CAPCOM1	CC0IR	CC0IE	CC0IC	0x00'0040	16
Canal 1 CAPCOM1	CC1IR	CC1IE	CC1IC	0x00'0044	17
Canal 2 CAPCOM1	CC2IR	CC2IE	CC2IC	0x00'0048	18
Canal 3 CAPCOM1	CC3IR	CC3IE	CC3IC	0x00'004C	19
Canal 4 CAPCOM1	CC4IR	CC4IE	CC4IC	0x00'0050	20
Canal 5 CAPCOM1	CC5IR	CC5IE	CC5IC	0x00'0054	21
Canal 6 CAPCOM1	CC6IR	CC6IE	CC6IC	0x00'0058	22
Canal 7 CAPCOM1	CC7IR	CC7IE	CC7IC	0x00'005C	23
Canal 8 CAPCOM1	CC8IR	CC8IE	CC8IC	0x00'0060	24
Canal 9 CAPCOM1	CC9IR	CC9IE	CC9IC	0x00'0064	25
Canal 10 CAPCOM1	CC10IR	CC10IE	CC10IC	0x00'0068	26
Canal 11 CAPCOM1	CC11IR	CC11IE	CC11IC	0x00'006C	27
Canal 12 CAPCOM1	CC12IR	CC12IE	CC12IC	0x00'0070	28
Canal 13 CAPCOM1	CC13IR	CC13IE	CC13IC	0x00'0074	29
Canal 14 CAPCOM1	CC14IR	CC14IE	CC14IC	0x00'0078	30
Canal 15 CAPCOM1	CC15IR	CC15IE	CC15IC	0x00'007C	31
Timer 0	T0IR	T0IE	T0IC	0x00'0080	32
Timer 1	T1IR	T1IE	T1IC	0x00'0084	33
Timer 7	T7IR	T7IE	T7IC	0x00'00F4	61
Timer 8	T8IR	T8IE	T8IC	0x00'00F8	62
A/D Conv. Complete	ADCIR	ADCIE	ADCINT	00'00A0	40
A/D Overrun Error	ADEIR	ADEIE	ADEINT	00'00A4	41
PWM channel 0..3	PWMIR	PWMIE	PWMINT	00'00FC	63

Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (1)

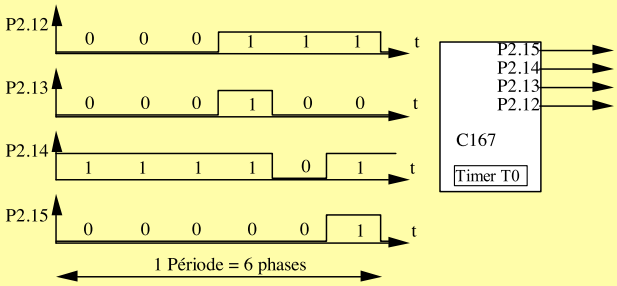


On utilise le timer T0. La durée d'une phase = une période du timer = **1ms**.

Codage	3 2 1 0			
	0 1 1 0	soit	6	Nombre de phases dans la table
	0 1 0 0	soit	4	Valeur de la phase 1
	0 1 0 0	soit	4	Valeur de la phase 2
	0 1 0 0	soit	4	Valeur de la phase 3
	0 1 1 1	soit	7	Valeur de la phase 4
	0 0 0 1	soit	1	Valeur de la phase 5
	1 1 0 1	soit	0x0d	Valeur de la phase 6

➡ Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (2)



Travail à effectuer :

- trouvez les **initialisations** pour les **ports** et le **timer**
- trouvez les **initialisations** pour les **interruptions**
- pour le "**travail de fond**" on fera un **signal carré** sur P7.0 (Led L1 et borne PWM0 de la carte).
  Absolument aucun intérêt mais cela permet de "cadrer" les structures.
- donnez le **programme principal** et la **procédure d'interruption**, **dessinez** un **chronogramme temporel** montrant ce qui peut se passer.

➡ Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (3)

Initialisations pour les ports et le timer

Timer T0 : **T0REL** : 1ms de période => 2500 tops => 63036 pour **T0REL**

T01CON : 0000 0000 0100 **0000** 0x0040

Port P2 : **DP2** ppv 0xF000
P2 ppv 0

Port P7 : **DP7** ppv 1
P7 ppv 1

```
void Init_ES (void)
{
  DP2 = 0xF000 ;
  P2 = 0 ;
  DP7 = 1 ;
  P7 = 1 ;
}
```

```
void Init_Timer (void)
{
  T0REL = 63036 ;
  T01CON = 0x40 ;
  T0IR = 0 ;
}
```

➡ Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (4)

Initialisations pour les interruptions

On veut qu'à chaque période du timer il y ait une interruption. C'est T0IR qui fait la demande. Il faut choisir son niveau d'interruption. La requête doit être validée, ainsi qu'au niveau global.

on choisit le niveau 4 (il doit être entre 1 et 13) pour ILVL et le groupe 0 dans GLVL.

T0IC :	00 **01 00 00**		T0IC = 0x10 ;
T0IE	ppv	1	T0IE = 1 ;
IEN	ppv	1	IEN = 1 ;

On ne fait pas de routine pour cela

Travail de fond : signal carré sur P7.0 (Led L1 et borne PWM0 de la carte)

```
P7 = P7 ^ 1;
```


➡ Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (5)

```
void Init_ES (void)
{
  DP2 = 0xF000 ;
  P2 = 0 ;
  DP7 = 1 ;
  P7 = 1 ;
}
```

```
void Init_Timer (void)
{
  T0REL = 63036 ;
  T01CON = 0x40 ;
  T0IR = 0 ;
}
```

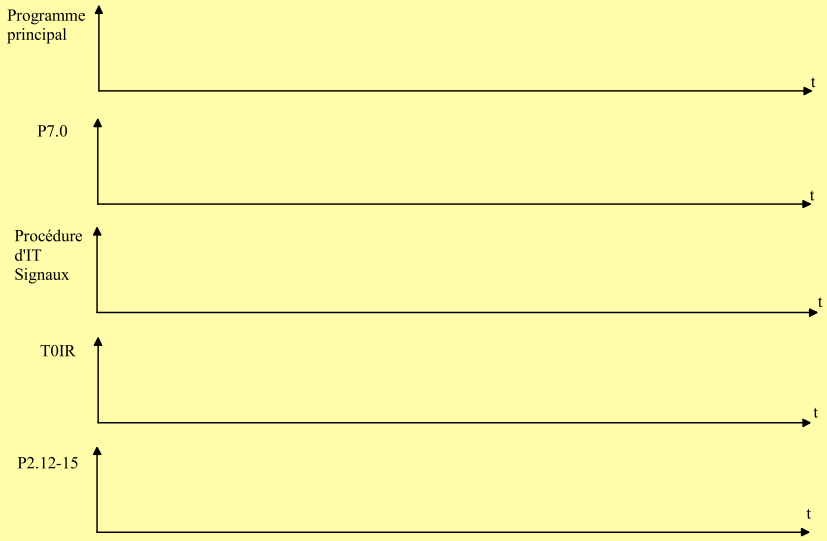
```
void main (void)
{
    Init_ES();
    Init_Timer() ;
    T0IC = 0x10 ;          // Timer 0 sur le niveau 4
    T0IE = 1 ;             // Timer 0 Interrupt Enable
    IEN = 1 ;              // Global Interrupt Enable
    while (1) {
        P7 = P7 ^ 1 ;
    }
}

void Signaux (void) interrupt 32
{
    static char I = 1, Table[7] = {6, 4, 4, 4, 7, 1, 0xD};
    P2 = Table[I] << 12 ;
    I++ ;
    if (I > Table[0]) I = 1 ;
}
```

➡ Exemple 1 : générateur de signaux

5.3.1. Exemple 1 (6)

Exemple de chronogramme temporel



Le processeur ne fait qu'une seule chose à la fois. Il est, soit dans le programme principal (dans la tâche de fond), soit dans la procédure d'interruption.

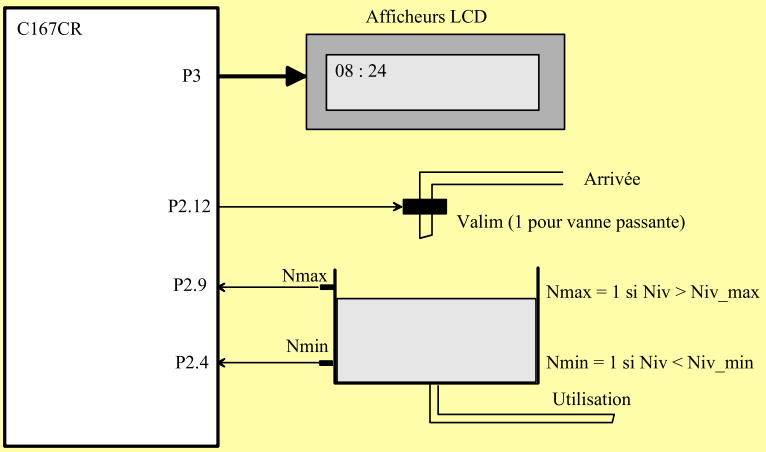
C'est sa grande vitesse d'exécution qui nous donne l'illusion du parallélisme d'exécution.

➡ Exemple 2 : régulation de niveau dans un bac - Version 1

5.3.2. Exemple 2 (1)

On veut en "même temps" :

- Indiquer sur une pendule (minutes et secondes sur un afficheur LCD), le temps écoulé depuis le lancement de l'application. Le temps sera calculé à partir d'un timer et on utilisera une interruption périodique, toutes les secondes ;
- Gérer le niveau du bac entre Nmin et Nmax. Ce niveau sera géré dans la tâche de fond, donc pas en régime d'interruption.

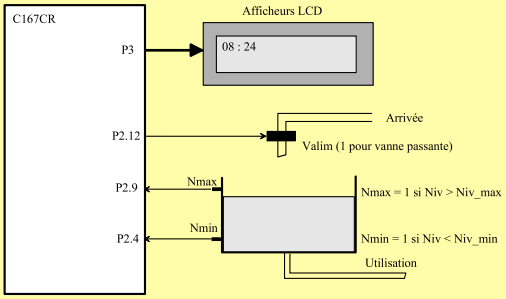


➡ Exemple 2 : régulation de niveau dans un bac - Version 1

5.3.2. Exemple 2 (2)

Pour utiliser l'afficheur, vous disposez des procédures suivantes :

void Init_LCD (**void**)	qui initialise le système d'affichage (y compris P3)
void PutStringLine1 (**char** *Str)	qui affiche la chaîne de caractères pointée par Str sur la ligne 1 de l'afficheur, à partir de la colonne 1
void PutChar (**char** CAR)	qui affiche le caractère CAR à la position courante



Travail à effectuer :

- trouvez les **initialisations** :

  pour les **ports**, le **timer, interruptions**

- donnez le **programme principal** et la **procédure d'interruption**

➡ Exemple 2 : régulation de niveau dans un bac - Version 1

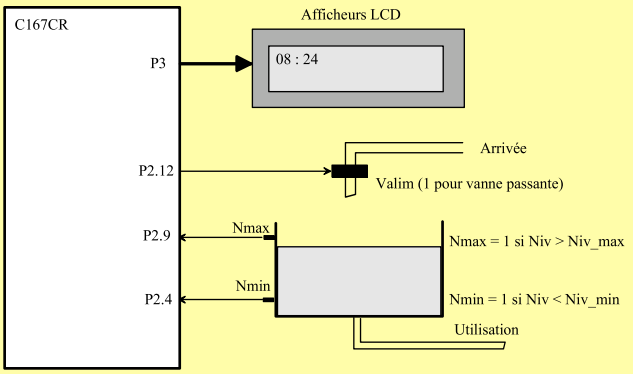
5.3.2. Exemple 2 (3)

Ports : on ne s'occupe pas de P3

DP2	ppv	0x1000
P2	ppv	0

```
sbit      Valim    = P2^12;
#define   Ouverte  1
#define   Fermee   0

void Init_ES (void)
{
  DP2 = 0x1000;
  Valim = Fermee;
}
```



➡ Exemple 2 : régulation de niveau dans un bac - Version 1

5.3.2. Exemple 2 (4)

Timer T0 résolution : de 25,6 µs(n=512), max : 1,68s.

1s. donne 39062,5 impulsions arrondies à 39063 (pour être en retard)

T0REL sera chargé à 26473

T01CON 0000 0000 0100 0110 0x0046

T0IR à 0

T0IC 00 01 00 00 level 4

T0IE à 1

```
void Init_Timer (void)
{
  T0REL = 26473 ;
  T01CON = 0x0046 ;
  T0IR = 0 ;
}
```


➡ Exemple 2 : régulation de niveau dans un bac - Version 1

5.3.2. Exemple 2 (5)

```
sbit      Nmax     =P2^9 ;
sbit      Nmin     =P2^4 ;

void main (void)
{
    Init_ES() ;
    Init_LCD() ;
    PutStringLine1("00 : 00") ;
    Init_Timer() ;

    T0IC=0x10 ;  // Timer 0 sur le niveau 4
    T0IE=1 ;     // Timer 0 Interrupt Enable
    IEN=1;       // Global Interrupt Enable

    while (1)  {
      if (Nmax)          Valim = Fermee ;
      if (Nmin)          Valim = Ouverte ;
    }
}
```

```
void Pendule (void) interrupt 32
{
   static unsigned int Secondes=0 ;
   static char Heure[8]= {' ',' ',' ',':',' ',' ',' ',0};
   unsigned char  Min, Sec ;

    Secondes++ ;
    Min = Secondes / 60 ;
    Sec = Secondes % 60 ;
    Heure[0]=Min/10 + 0x30 ;
    Heure[1]=Min%10 + 0x30 ;
    Heure[5]=Sec/10 + 0x30 ;
    Heure[6]=Sec%10 + 0x30 ;
    PutStringLine1(Heure) ;
}
```

➡ Objectifs

Obtenir une description du programme qui soit plus lisible qu'un algorithme et montre les liens entre les différentes entités supposées s'exécuter en parallèle, chacune réalisant une fonction dans le système.

- La structure fonctionnelle ne remplace pas l'algorithme, elle le complète en apportant un autre niveau de description.

➡ Que représente-t-on dans cette structure ?

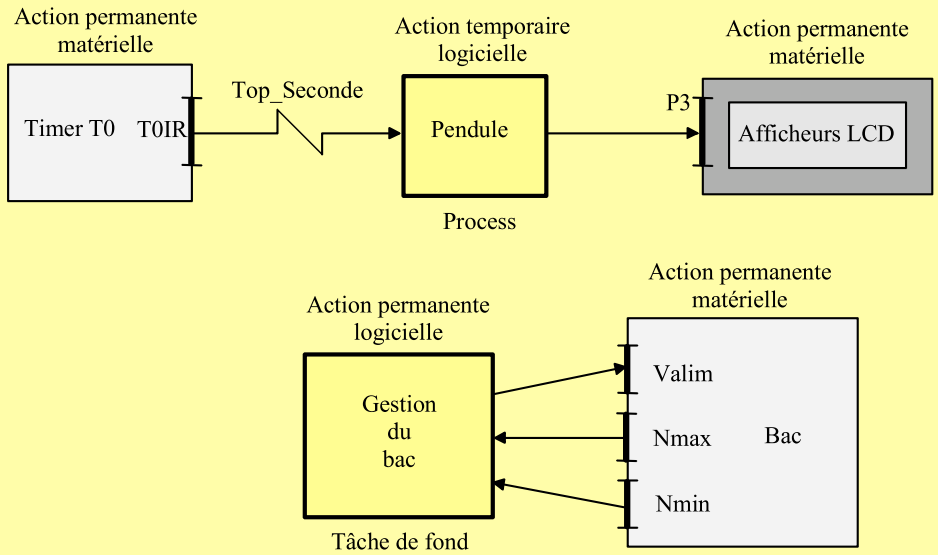
•Les **actions** supposées s'exécuter en parallèle :

- les actions **matérielles** réalisées par des circuit
- les actions **logicielles** (**temporaires**) réalisées par les **Process**
- l'action **logicielle permanente** (la tache de fond)

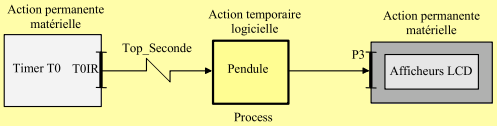
Les **liens** entre actions par l'intermédiaire de :

- *variables* partagées
- *événements* (matérialisés par des IT)

➡ structure fonctionnelle de l'exemple précédent



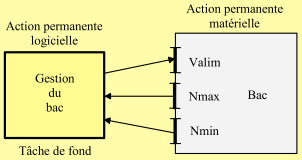
➡ structure fonctionnelle de l'exemple précédent



Actions permanentes matérielles (les circuits physiques) :

- le **timer**
- **l'afficheur** LCD
- les **capteurs** / **actionneur** du bac

Tout cela travaille en réel parallélisme avec le processeur



Action permanente logicielle :

la **tâche de fond** qui s'exécute tout le temps, sauf lorsqu'elle est interrompue.
L'algorithme de cette action est la partie du programme principal se trouvant **après** l'autorisation des IT.

Avant, il ne peut y avoir « **pseudo-parallélisme** » d'exécution, et donc cette partie d'initialisation n'est pas représentée par l'action « Tâche de fond ».

Action temporaire logicielle :

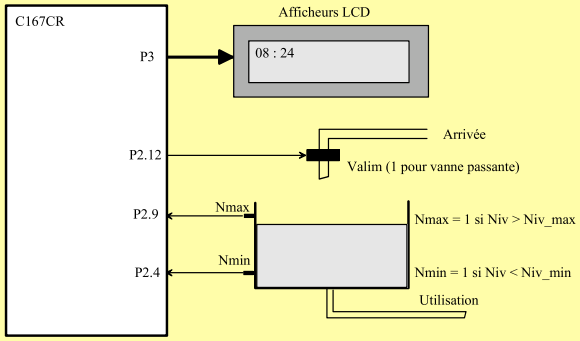
le **process**, procédure activée par le matériel, qui « **se réveille** » au moment de l'interruption **et interrompt** la tâche de fond, puis « **s'endort** » après avoir fait son travail (action de RETI qui « rend la main » au programme interrompu).

➡ Exemple 3 : régulation de niveau dans un bac - Version 2

On veut en "même temps" :

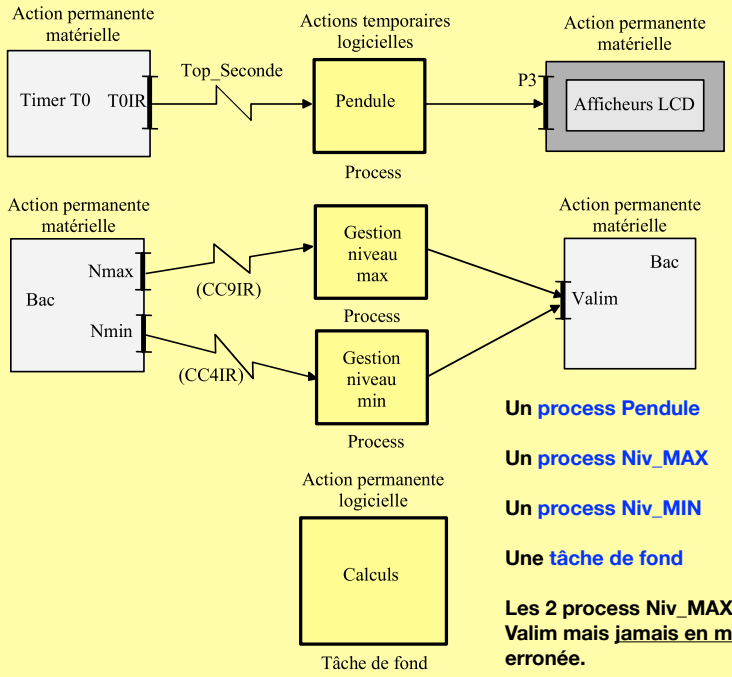
- Indiquer sur une pendule (minutes et secondes sur un afficheur LCD), le temps écoulé depuis le lancement de l'application. Le temps sera calculé à partir d'un timer et on utilisera une interruption périodique, toutes les secondes ;
- Gérer le niveau du bac entre Nmin et Nmax. Ce niveau sera géré en régime d'interruption.
- faire des calculs (non explicités pour ne pas surcharger) dans la tâche de fond

On a la même structure matérielle que précédemment mais on utilise la détection de transitions sur les lignes P2.4 et P2.9 (en entrées en capture) afin d'engendrer des interruptions.



- faire la structure fonctionnelle
- donnez tous les algorithmes

➡ Structure fonctionnelle proposée



Un process Pendule activé par l'IT liée à T0IR

Un process Niv_MAX activé par l'IT liée à CC9IR

Un process Niv_MIN activé par l'IT liée à CC4IR

Une tâche de fond

Les 2 process Niv_MAX et Niv_MIN gèrent la même variable Valim mais jamais en même temps. Pas de risque de gestion erronée.

➡ Initialisations

```
sbit      Valim     =P2^12;
#define   Ouverte   1
#define   Fermee    0

void Init_ES (void)
{
    DP2 = 0x1000;
    Valim = Fermee;
    CCM1 = 1 ;
    CC4IR = 0 ;
    CC4IC = 0x14 ;
    CCM2 = 0x10 ;
    CC9IR = 0 ;
    CC9IC = 0x15 ;
}
```

Calcul du temps de 1 seconde : **IDEM**

Ports : on ne s'occupe pas de P3 -
P2 IDEM pour la sortie

Config. des entrées de capture

CCM1	ppv	1	front montant pour CC4IO	Nmin
CC4IR	ppv	0		
CC4IC	ppv	00 01 01 00	0x14	level 5, groupe 0
CCM2	ppv	0x10	front montant pour CC9IO	Nmax
CC9IR	ppv	0		
CC9IC	ppv	00 01 01 01	0x15	level 5, groupe 1

➡ Programme

```
#include <reg167.h>
sbit        Valim       =P2^12;
#define     Ouverte     1
#define     Fermee      0


void main (void)
{
   Init_ES();
   Init_LCD() ;
   PutStringLine1("00 : 00") ;
   Init_Timer();

   T0IC=0x10;              // Timer 0 sur le niveau 4
   T0IE=1;                 // Timer 0 Interrupt Enable
   CC4IE=1 ;// Capture canal 4 Enable
   CC9IE=1 ;// Capture canal 9 Enable
   IEN=1;                  // Global Interrupt Enable

   while (1)   Calculs() ;
}
```

```
void Init_ES (void)
{
    DP2 = 0x1000;
    Valim = Fermee;
    CCM1 = 1 ; CC4IR = 0 ; CC4IC = 0x14 ;
 CCM2 = 0x10 ; CC9IR = 0 ; CC9IC = 0x15 ;
}

void Init_Timer (void)
{
                IDEM
}


void Pendule (void) interrupt 32
{
                IDEM
}

void Niv_Max (void) interrupt 25
{
                Valim = Fermee ;
}

void Niv_Min (void) interrupt 20
{
                Valim = Ouverte ;
}
```