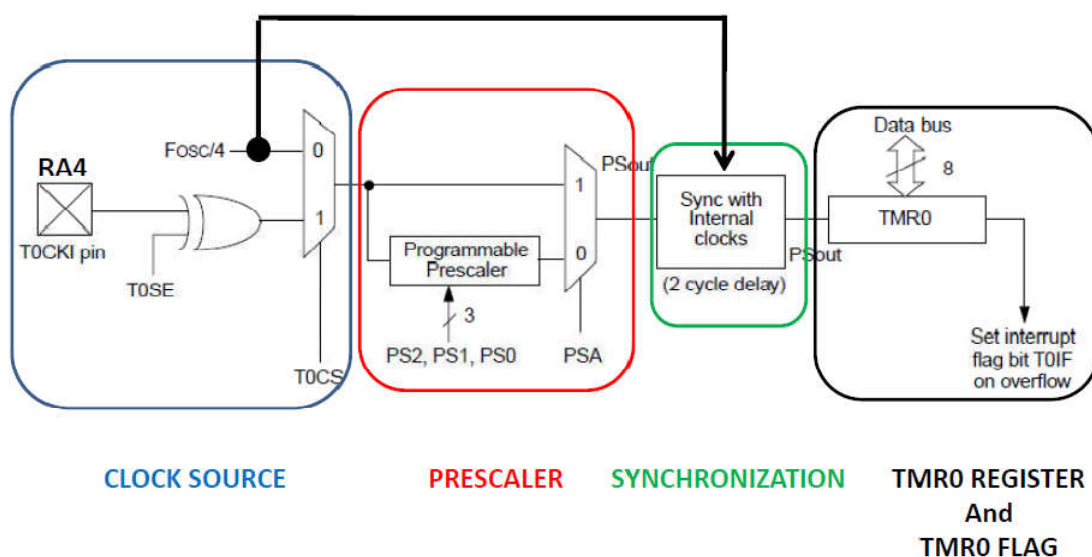


LAB4 (2^{ème} partie)

Le but du LAB est comment programmer une temporisation à base du SOFT et comment faire aussi avec du HARD TIMER0.

Dans cette partie, on va faire le même travail mais avec du HARD TIMER0.

Avant de commencer, il faut faire un petit tour sur le module TIMER0



Cette figure montre le schéma block fonctionnel de module TIMER0 qui se compose par 4 unités

- Unité de sélection de source d'incrémentations,
- Unité de pré-division,
- Unité de synchronisation,
- Le registre TMR0 de 8 bits et le flag T0IF.

Selon la source d'incrémentation, le module TIMERO travaille soit en mode temporisateur si la source est $F_{osc}/4$, soit en mode compteur d'événement si la source est la patte RA4.

Notre LAB demande de faire des temporisations pour créer un signal carré afin de clignoter une LED reliée à la patte RB0.

La fréquence de clignotement est de 2.5Hz (une période de 400ms), donc on garde la patte RB0 à l'état haut durant 200ms puis on passe vers l'état bas durant le reste de la période (200ms).

Dans la version SOFT on fait le code celui-ci :

```
BSF PORTB, RB0  
CALL DELAY1  
BCF PORTB, RB0  
CALL DELAY2  
GOTO $-4
```

Avec DELAY1 et DELAY2 se sont des sous-programmes de temporisation de grande précision.

Dans notre cas, les deux temporisations on va les changer leur code principal par un autre code gère le module TIMERO afin de temporiser le temps nécessaire pour créer correctement notre signal carré.

Soit le pseudo-code suivant (Q=4MHz) :

- T0CS \leftarrow 0 ($F_{OSC}/4$ ou T_{cycle} comme source d'incrémentement)
- PSA \leftarrow 0 (associer le pré-diviseur au TIMERO)
- PS2, PS1, PS0 \leftarrow B'001' (pré-diviseur 1 :4)
- T0IF \leftarrow 0 (effacement du flag)
- TMR0 \leftarrow 250 (initialisation de registre TMR0)

$$F_{OSC} = 4\text{MHz} \rightarrow$$

$$T_{OSC} = 1/F_{OSC} = 0.25\mu\text{S} \rightarrow$$

$$T_{cycle} = T_{OSC} * 4 = 1\mu\text{S}.$$

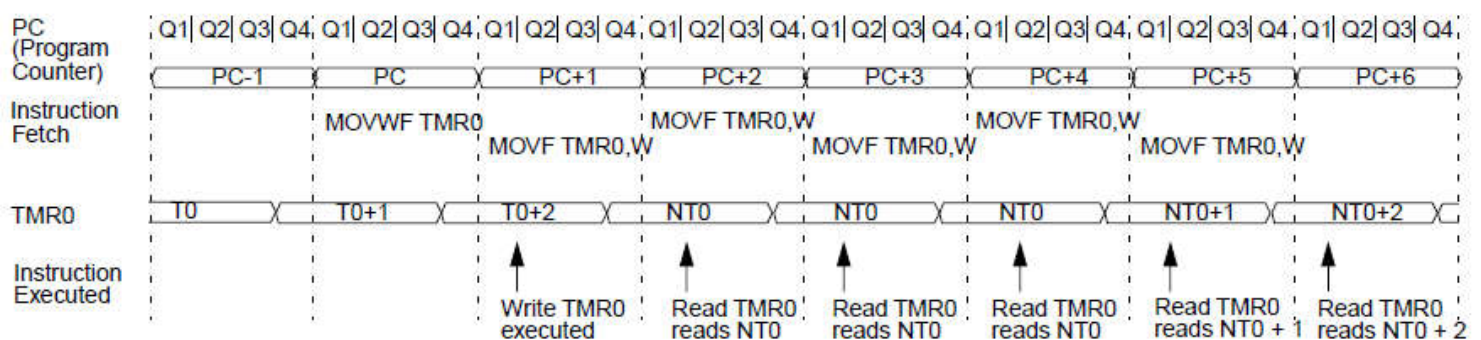
Après comme bien du temps le flag T0IF positionne à '1'.

La plus part des étudiants raisonnent comme suit :
 PSA est '0' donc le pré-diviseur est associé au module TIMER0 avec un rapport de 1 :4 selon les valeurs de PS2, PS1 et PS0, en plus le registre TMR0 commence par la valeur 250, donc à travers le mécanisme de positionnement du flag T0IF (positionne à 1 dès qu'il y a un passage de la valeur 255 vers 0) la réponse est la suivante :

$$T = (256-250) * \text{pré-diviseur} * T_{\text{cycle}} = 6 * 4 * 1\mu\text{s} = 24\mu\text{s}.$$

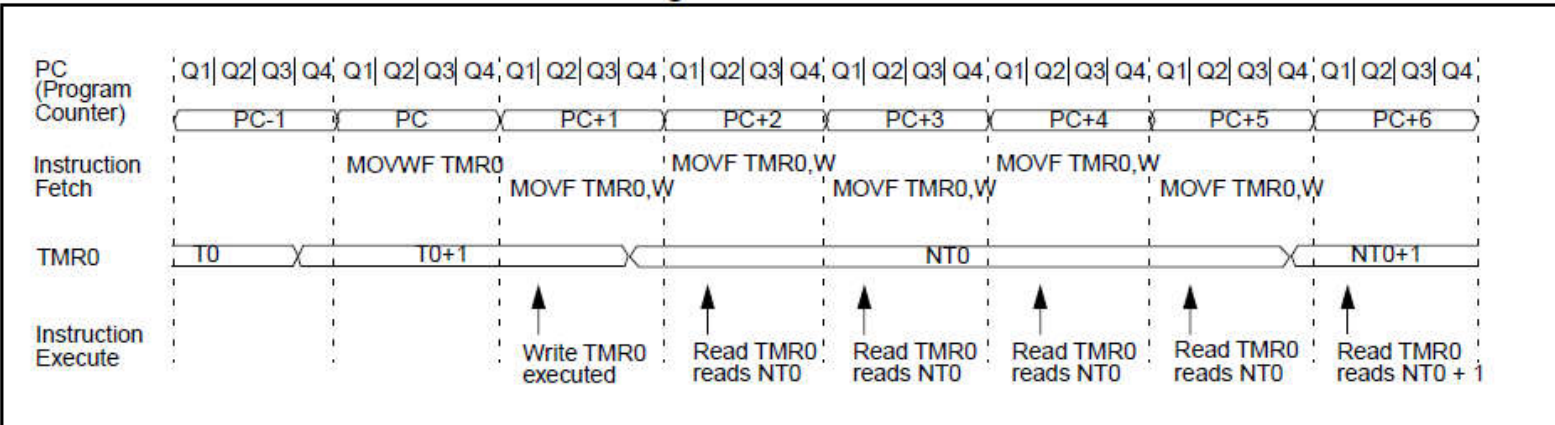
Mais la réalité, c'est une autre histoire ! L'écriture dans le registre TMR0 non seulement fait l'effacement de pré-diviseur, mais aussi elle ne fait pas un chargement immédiat de la nouvelle valeur dans le registre TMR0. Soit les chronogrammes suivants (source : MID-RANGE MCU FAMILY REFERENCE MANUAL):

Timer0 Timing: Internal Clock/No Prescale



Que remarquez-vous ?

Timer0 Timing: Internal Clock/Prescale 1:2



Que remarquez-vous ?

Ce problème apparaît seulement au moment d'un pré-diviseur 1:1 (absence de pré-diviseur) et un pré-diviseur 1:2. Donc, faites attention au moment de calculer la valeur qu'on va mettre dans le registre TMR0.

Les formules exactes :

Pour un pré-diviseur 1:1

$$T = (256 - \text{TMR0_INIT}) * 1 * T_{\text{CYC}} + 4 * T_{\text{CYC}}$$

Pour un pré-diviseur 1:2

$$T = (256 - \text{TMR0_INIT}) * 2 * T_{\text{CYC}} + 5 * T_{\text{CYC}}$$

Pour un pré-diviseur 1:4 jusqu'à 1:256

$$T = (256 - \text{TMR0_INIT}) * \text{pré-diviseur} * T_{\text{CYC}} + 2 * T_{\text{CYC}}$$

D'après l'étude qu'on a faite dans la première partie le premier délai épuise 199999 cycles et le deuxième délai épuise 199997 cycles.

Si on utilise la formule du temps passé par le TIMER0, on va trouver le suivant :

$$T = (256 - \text{TMRO_REG}) * \text{PRESCALER} * T_{\text{CYC}}$$

Le temps max qu'on peut faire avec un Q=4MHz par le module TIMER0 est :

$$T = (256-0) * 256 * 1\mu\text{s} = 65536\mu\text{s} = 65.536\text{ms}$$

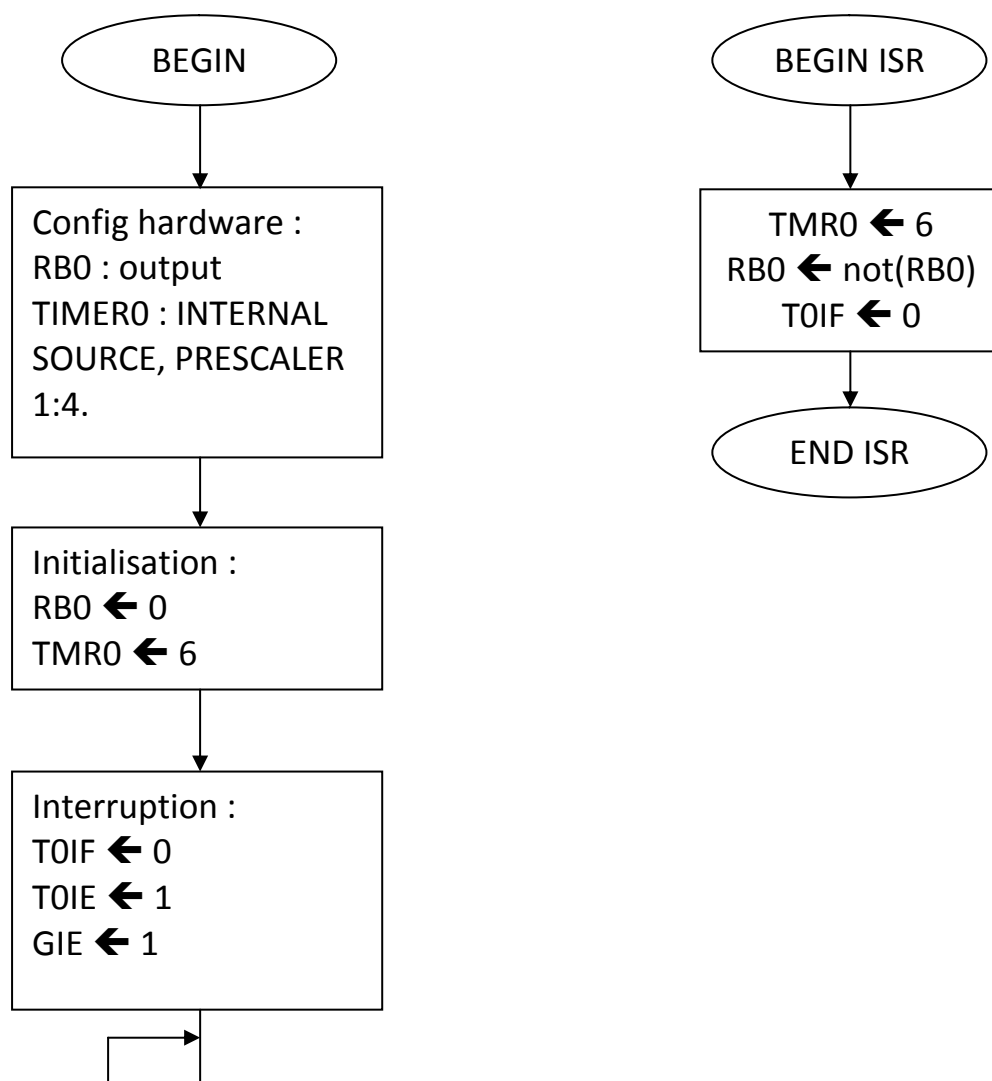
Par rapport à 199.999ms ou 199.997ms nous sommes loin !!! Comment faire ???

En plus, pour assurer une telle précision par le module TIMER0 reste un travail dur et difficile à faire si on garde le même code de la première partie!!! Comment faire ???

Apparemment, on est obligé de changer notre raisonnement ! Tout le monde connait que le module TIMERO génère une demande d'interruption au moment de passage de registre TMRO de la valeur 255 vers 0 avec l'activation de sa source par TOIE.

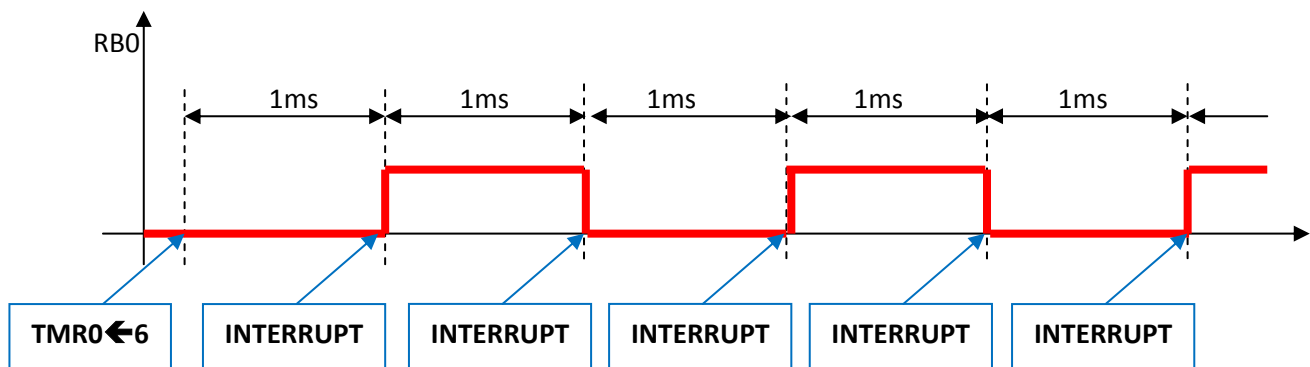
Comment exploiter cette propriété pour résoudre notre problème ?

Soit l'organigramme suivant :



Quel est le temps réalisé par le module TIMER0 si $Q=4\text{MHz}$? Et dessiner le signal généré sur la patte RB0.

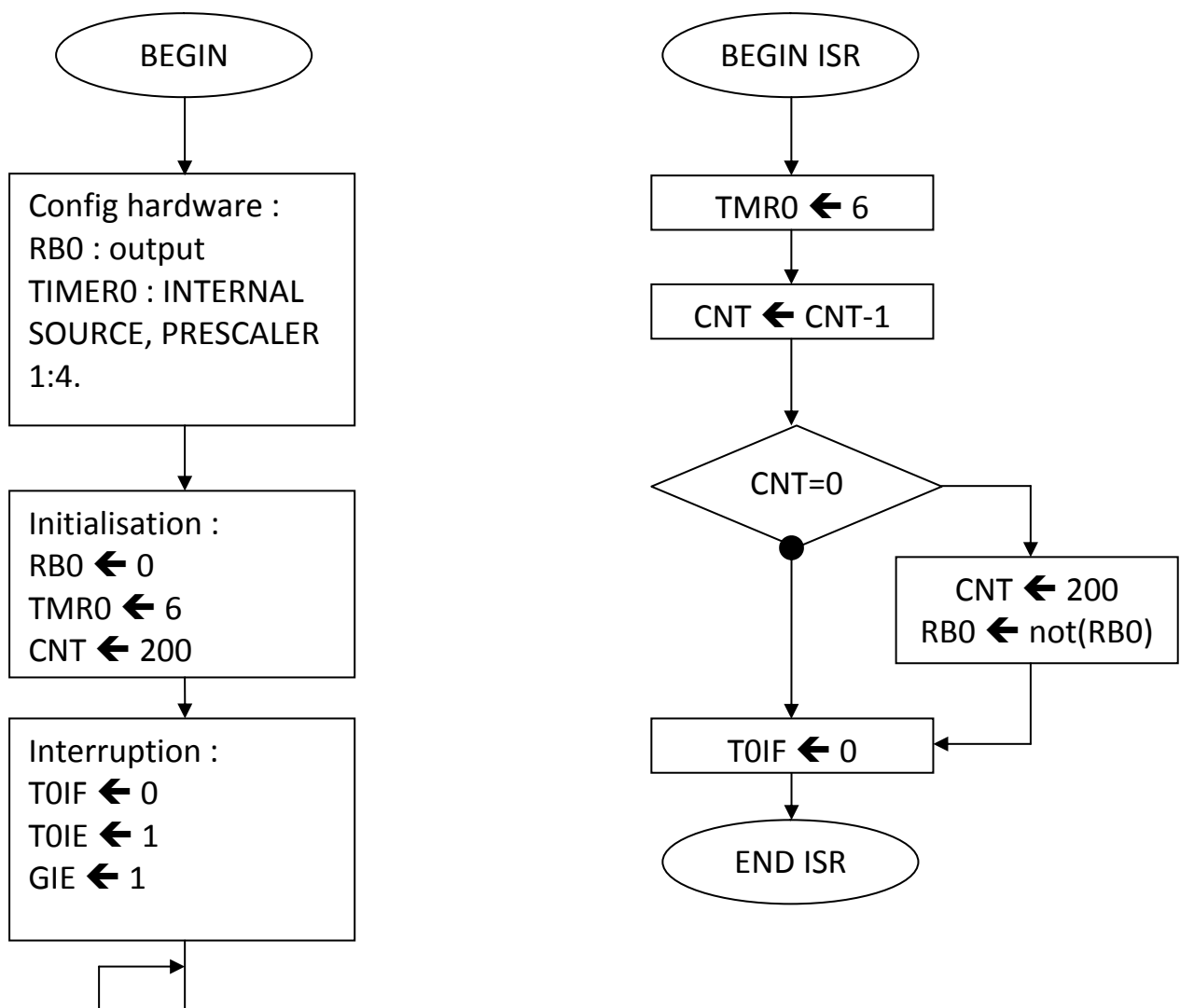
$$T = (256-6) * 4 * 1\mu\text{s} = 1000\mu\text{s} = 1\text{ms}.$$



Comment modifier cet organigramme pour changer l'état de la patte RB0 chaque 200ms ?

La première idée qui apparaisse à notre esprit, c'est d'utiliser une variable compte ou décompte 200 interruptions de TIMER0 puis on change l'état de la patte RB0.

Organigramme modifié :



Pour diminuer le stress sur le programme principal (une interruption chaque 1ms, donc 200 interruptions pour former 200ms), on change la valeur de pré-diviseur par une autre valeur interrompe moins le programme principal et reste un diviseur de 200ms.

Le plus grand diviseur qu'on peut utiliser et reste un diviseur de 200ms est 1 :32. Donc, la valeur qu'on va charger dans la variable CNT est 25.

$$T = (256 - 6) * 32 * 1\mu s = 8000\mu s$$
$$200000 / 8000 = 25$$

Donc, au lieu d'interrompre le programme principal 200 fois par un pré-diviseur 1 :4; avec un pré-diviseur de 1 :32 on arrive à interrompu 25 fois seulement.

Il existe une autre solution par le biais de module TIMERO sans passé par une demande d'interruption. Si on prend le code optimal de la première partie qui utilise une seule temporisation, la modification à faire, c'est de changer le code interne de délai DELAY_199997 par un autre gère le module TIMERO qui passe à peu près 199997 cycles.

Algorithme :

```
; Initialisation
RB0 ← 0;

; Clignotement de la LED
While (True)
    RB0 ← not(RB0);
    Call DELAY;
End While

;=====
; sous-programme DELAY
; on configure le TIMERO par :
; source interne, 1 :256
; positionne le flag T0IF chaque ~ 50000 cycles
DELAY:
Configuration TIMERO;
CNT ← 4;
Do
    TMR0 ← 61;
    T0IF ← 0;
    While (T0IF==0)
        // Rien (attendre le positionnement T0IF)
    End While
    CNT ← CNT-1;
While (CNT≠0)
```

Assembleur :

```
; Initialisation
        BCF PORTB, RB0
; Clignotement
        MOVLW B'00000001'
AGAIN:   XORWF PORTB, F
        CALL DELAY
        GOTO AGAIN

DELAY:
        ; Config TIMER0
        ; INTR-SRC, 1 :256
        BSF STATUS, RP0
        MOVLW B'10000111'
        MOVWF OPTION_REG
        ;=====
        BCF STATUS, RP0
        MOVLW D'4'
        MOVWF CNT
        ;=====
        MOVLW D'61'
        MOVWF TMR0
        BCF INTCON, T0IF
        BTFSS INTCON, T0IF
        GOTO $-1
        DECFSZ CNT, F
        GOTO $-6
        RETURN
```