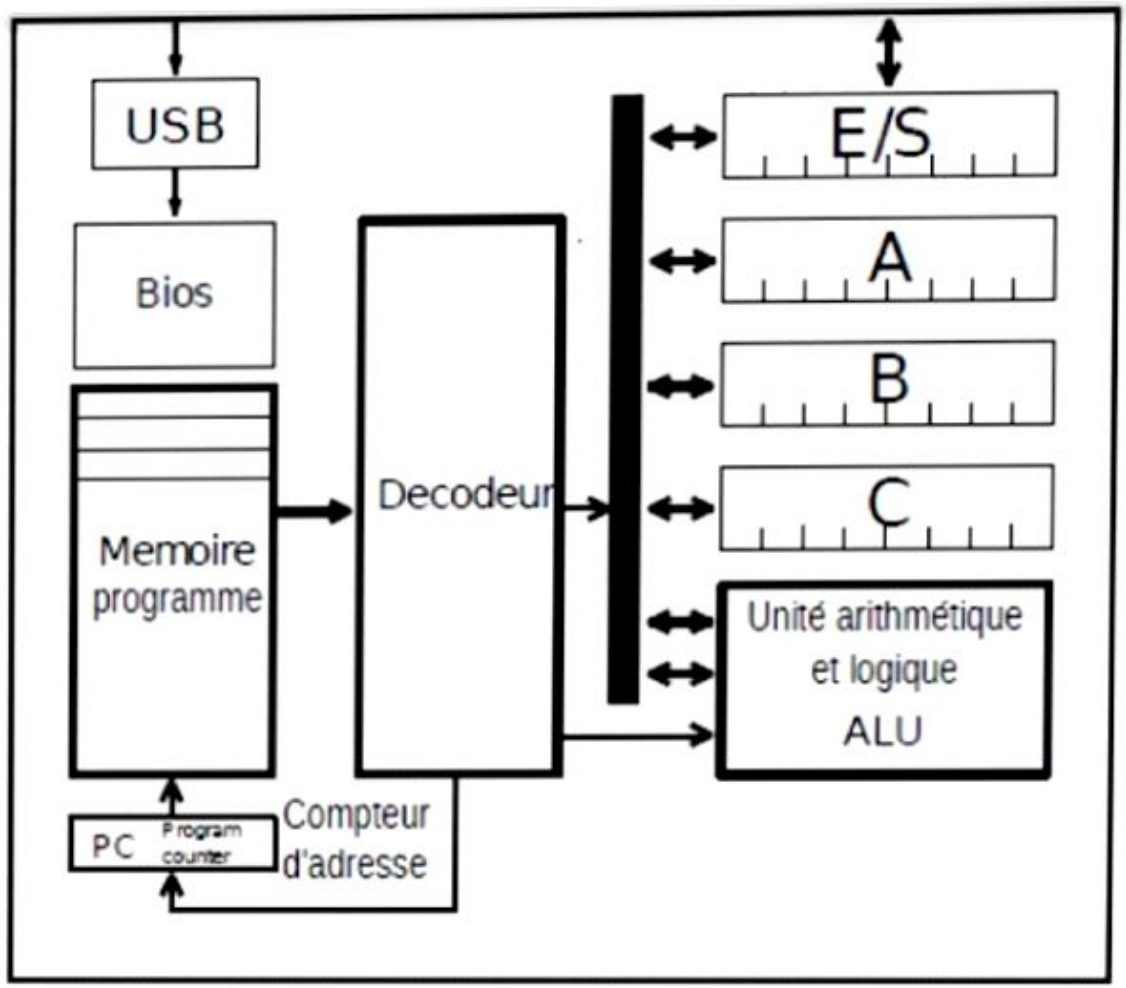
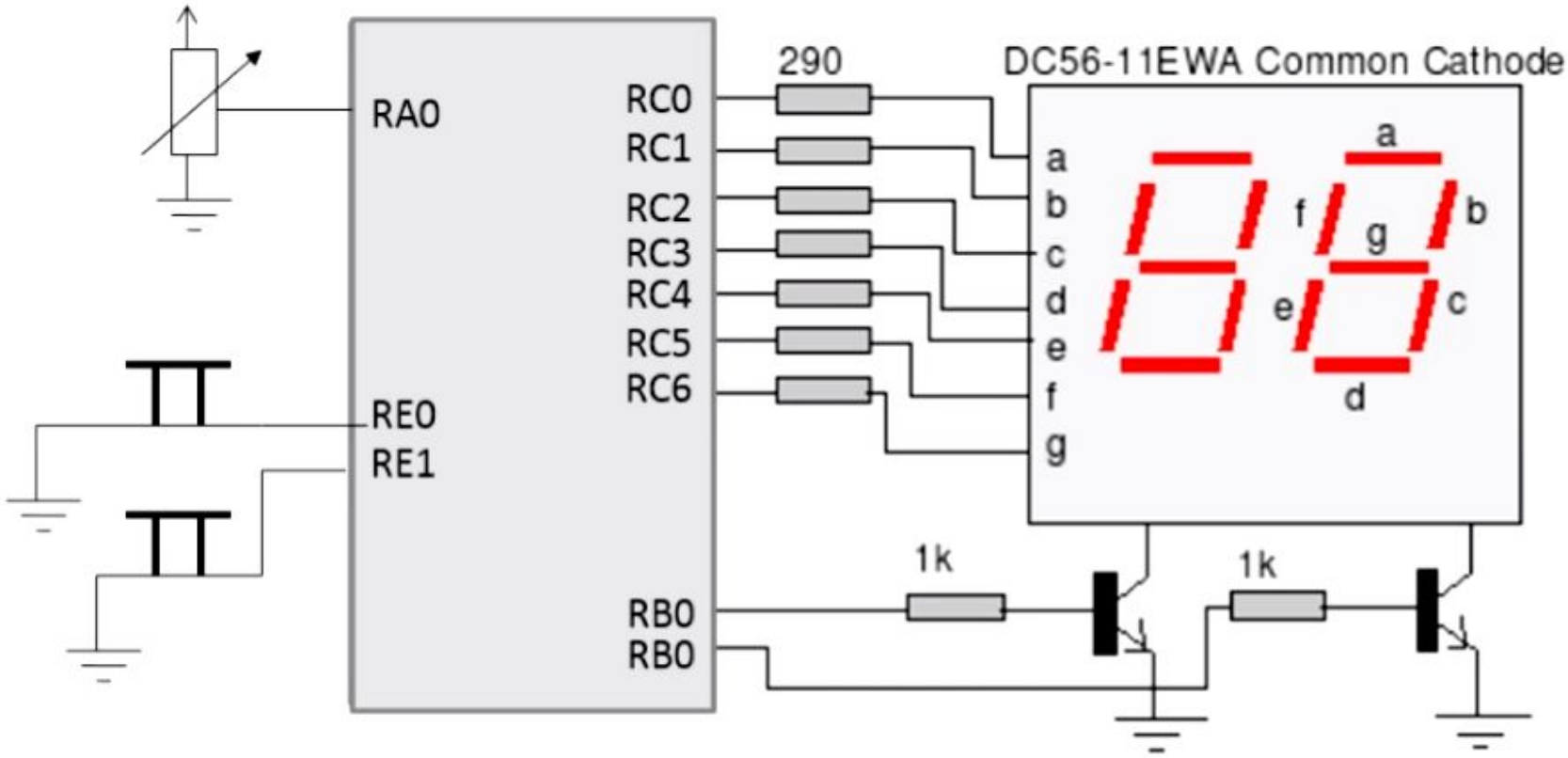
Microcontrôleur minimal



Exercice

On désire réaliser un voltmètre qui affiche la valeur de la tension lue sur une résistance variable sur deux digit en sept segments. Le voltmètre possède deux bouton pour la Marche/Arret et le reset selon le schéma suivant :



Exercice

Réponse :

PINs	configuration	valeur	Type
RC (0-6)	Sortie	**0**	Numérique
RA(0)	Entrée	**1**	Analogique
RE(1,0)	Entrée	**3**	Numérique
RB(1,0)	Sortie	**0**	Numérique

Configuration des I/O résumé

Etape 1)
Déterminer les entrées Analog ou numériques

Etape 2)
Choisir le port via lequel le microcontrôleur va communiquer avec l'éxterieur (en tenant compte de l'etape précédant)

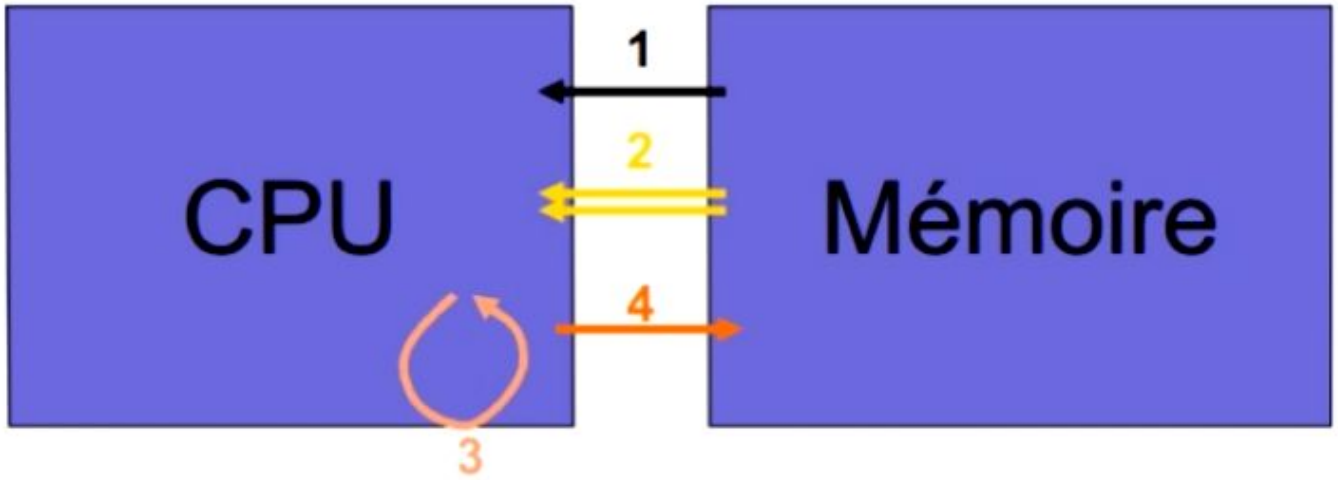
Etape 3)
Choisir la valeur des registres TRIS (A-E) avec 0 pour une sortie et 1 pour une entrée

Etape 4)
Choisir la valeur des registres ADCON pour la configuration des entrées analogiques

Etape 5)
Au cas ou vous utiliser des boutons ou switch il est préférables d'utiliser le PORTB puisqu'il contient des résistances de pull up et connecté au gestionnaire d'interruption

Etape 6)
Il ne faut pas oublier de configurer les registres en relation avec les ports au cas ou des pin spéciaux sont utilisées comme : OPTION_REG (RBPU,..)
INCON,..

Le fonctionnement de base d'une instruction



- (1) Charger une instruction depuis la mémoire
- (2) Charger les opérandes depuis la mémoire
- (3) Effectuer les calculs
- (4) Stocker le résultat en mémoire

Exercice

Solution: La portion de code assembleur est le suivant

Configuration du Port C

```
MOVLW 0x00
MOVWF TRISC
```

Configuration du Port A

```
MOVLW 0x01
MOVWF TRISA
```

Configuration du Port E

```
MOVLW 0x03
MOVWF TRISE
```

Exercice

Solution: La portion de code assembleur est le suivant

Configuration du Port C

```
CLRW
MOVWF TRISC
CLRF PORTC
```

Configuration du Port A

```
MOVLW 0x01
MOVWF TRISA
```

Configuration du Port E

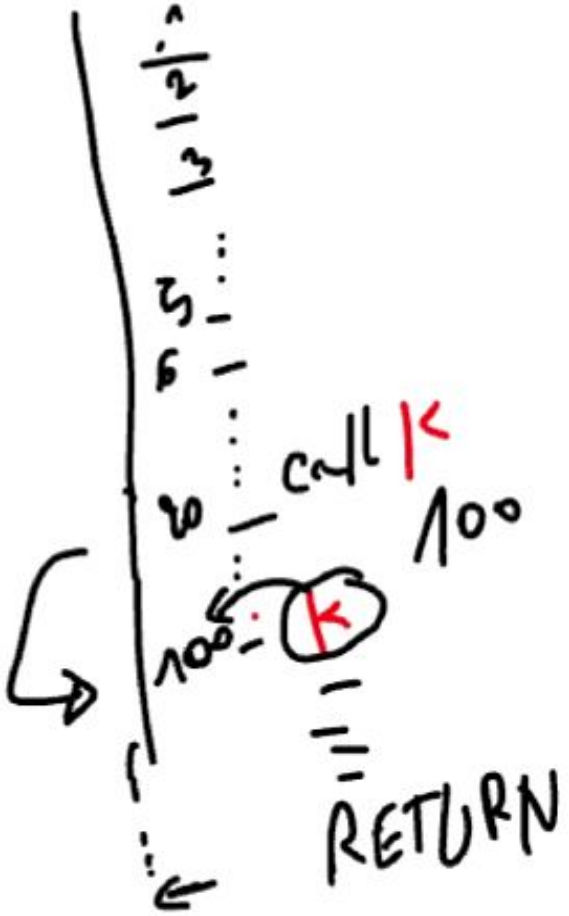
```
MOVLW 0x03
MOVWF TRISE
```

CALL k

PC : 20

R → 100

PC → 21



Exercice

On est dans le bank 1 => RP1=0 et RP0=1

Pour se mettre dans le bank 0 => RP1=0 et RP0=0

=> Il faut mettre le bit RP0 à zéro => **BCF STATUS,5**

Pour se mettre dans le bank 3 => RP1=1 et RP0=1

=

Directives

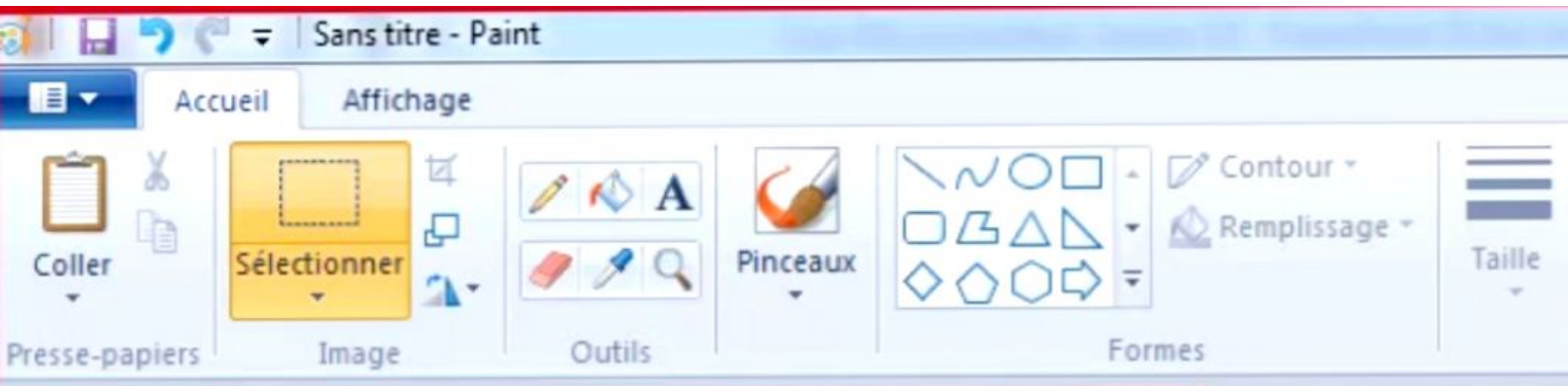
- List p=16f877A ; pour choisir le Microcontrôleur
- include "p16f877A.inc ; inclure la bibliothèque cible

-Déclaration de variable

```
CBLOCK   Adresse départ
    var1  :1
    Temps: 3
ENDC
```

- Choix du Bank

Banksel **Registre**



Premier Programme Hello World !!

Un programme qui inverse la sortie PORTB continuellement

```
list p=16f877A                  ; pour choisir le Microcontrôleur
include "p16f877A.inc"          ; inclure la bibliothèque cible
        Init                    ; Init est un label (peut être début,..)
          BSF  STATUS,5   ; mettre le bit RP0 à 1
          BCF  STATUS ,6  ; mettre le bit RP1 à 0 pour choisir le banc 1
          CLRF  TRISB     ; configure PORTB en sortie
          BCF  STATUS,5   ; Revenir au Bank 0
        Main                    ; label main
          MOVLW  b'01010101' ;  mettre une valeure dans Wreg
          MOVWF  PORTB       ;  initialiser PORTB
    Boucle                      ; label Boucle
          MOVLW  0xFF           ;  mettre à 1 Wreg
          XORWF  PORTB,1        ; inverser le PORTB
      goto boucle               ; revenir à Boucle
End
```


```
Boucle                  ; label Boucle
    MOVLW  0xFF         ; mettre à 1 Wreg
    XORWF  PORTB,1      ; inverser le PORTB
```

Cblocle 0x90
temp0 : 1
temp1 : 1
endc

boucle1
boucle0
INCFSZ Temp0,1
goto boucle0
INCFSZ Temp1,1
goto boucle1

0 → 255 → 0 → 255

Temp1 : 2 → 2 → 255 → 0

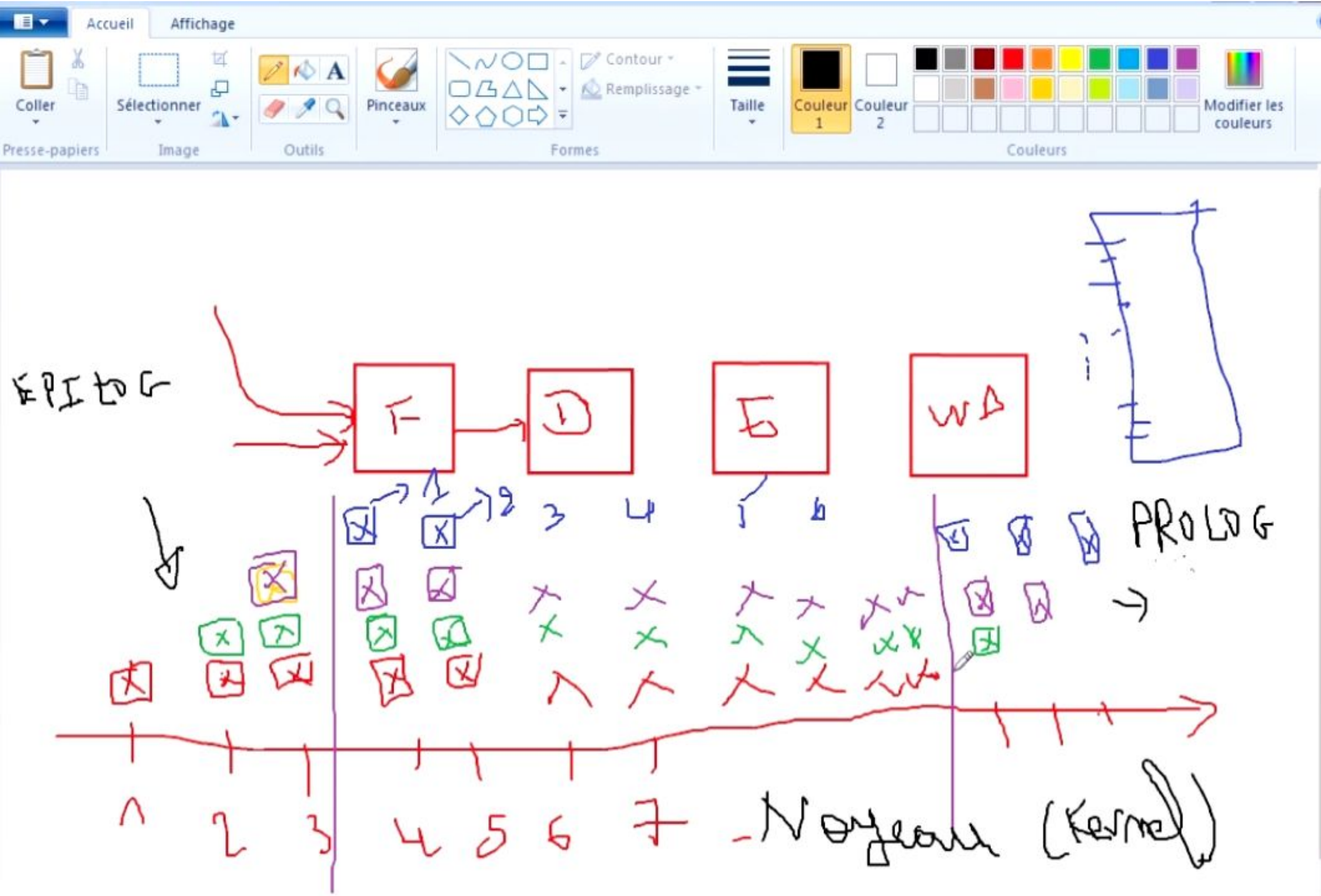
1 instr → 50 ns
4x instr → 200 ns

1384 x temp

97600000 ns

2,76 ms

```
goto boucle             ; revenir à Boucle
```



Exercice

On désire programmer la suite de fibunacci par le microcontrolleur PIC16F877. les éléments de la suite doivent être affecté au port B. Ecrire les instructions assembleur nécessaire en se basant sur la structure suivante et les variables Temp0 et Temp1 déjà déclaré.

Configuration et initialisation

Instructions

Boucle

Instructions

Aller à boucle

Configuration et initialisation

```
CLRF TRISTA // configurer le PORTA en sortie
CLRF Temp0          // mettre à zéro Temp0
MOVLW  0X01         // mettre 1 dans W
MOVWF  Temp1        // initialiser la variable Temps par 1
```

Boucle :

```
MOVF    Temp0,0
ADDWF   Temp1,0
MOVWF Temp0
MOVWF PORTA
MOVF    Temp1,0
ADDWF   Temp0,0
MOVWF Temp1
MOVWF PORTA
```

Registre W	1 0 1 1 1 1 2 2 2 2 1 3 3 3 2 5 5 5 5
Temp0	0 0 0 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3
Temp1	1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 5 5 5
PORTA	0 0 0 0 1 1 1 1 2 2 2 2 2 3 3 3 3 5 5

Suite de fibunacci

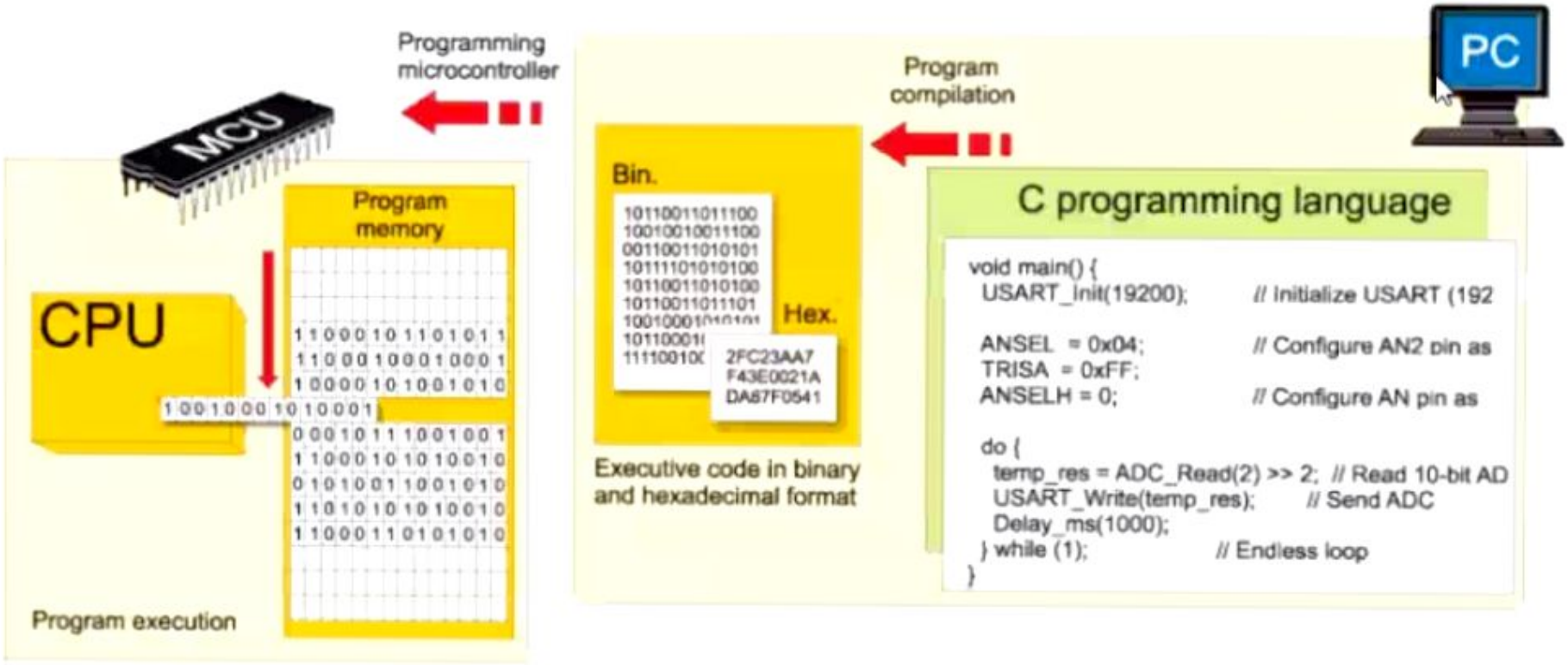
0	1	2	3	5

Aller à boucle

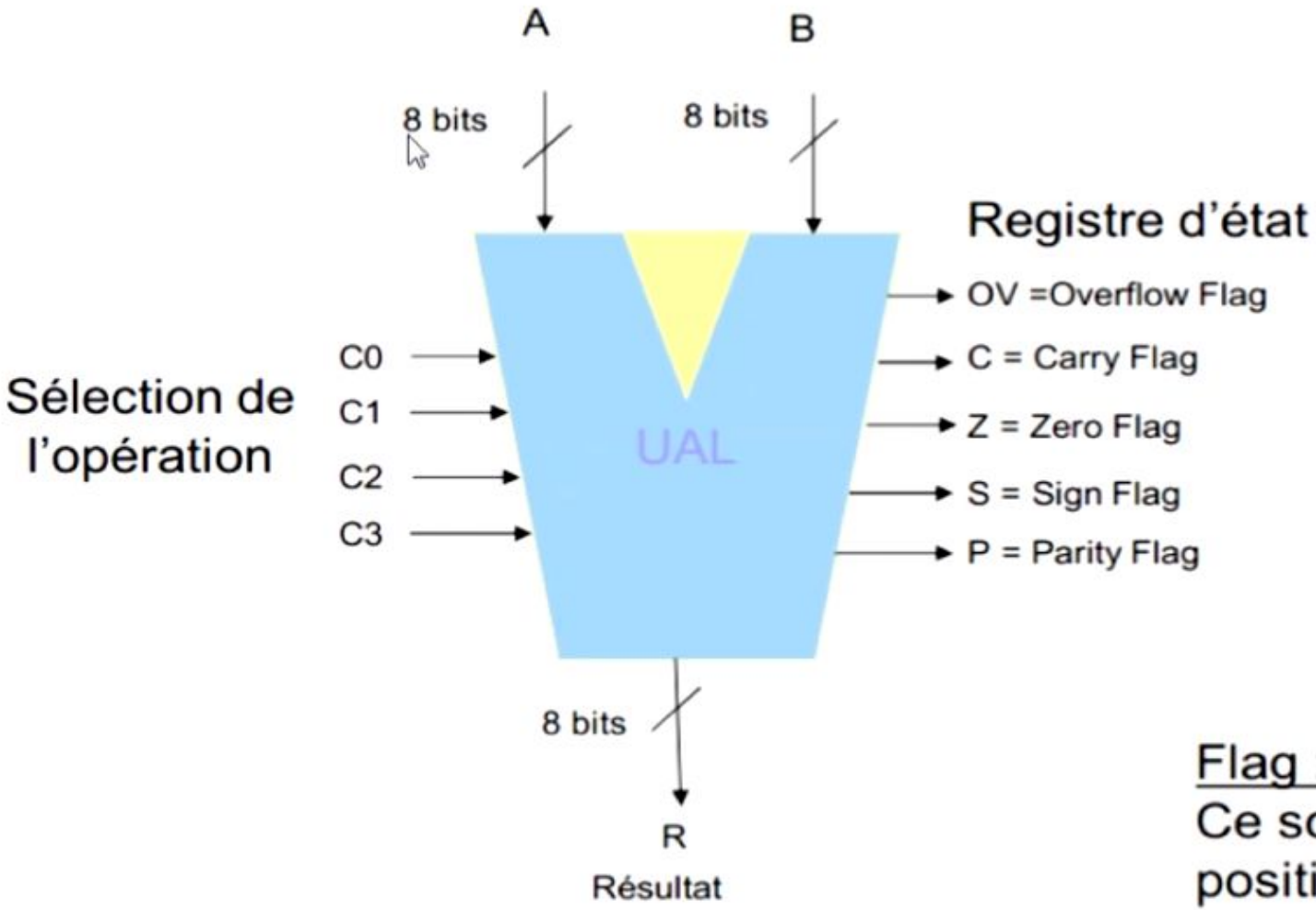
Programme ASM complet

```
List p= 16F877A
Include <p16F877A.inc>
CBLOCK 0x20
      Temp0
      Temp1
ENDC
Init
      CLRF PORTA       // configurer le PORTA en sortie ( CLRF 0x05)
      CLRF Temp0           // mettre à zéro Temp0
      MOVLW  0X01          // mettre 1 dans W
      MOVWF  Temp1     // initialiser la variable Temps par 1
Boucle
      MOVF    Temp0,0
      ADDWF  Temp1,0
      MOVWF Temp0
      MOVWF PORTA
      MOVF    Temp1,0
      ADDWF  Temp0,0
      MOVWF Temp1
      MOVWF PORTA
Goto Boucle
Goto Init
END
```

Chaine de compilation



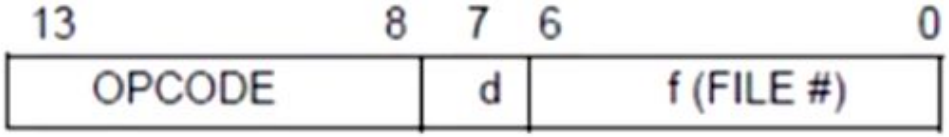
OP Code (code d'opération)



Flag :
Ce sont des Bits positionnées dans un registre

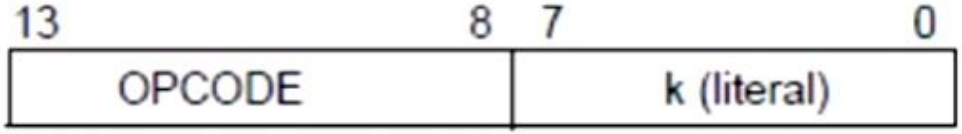
OP Code (code d'opération)

Opération qui manipule un Byte



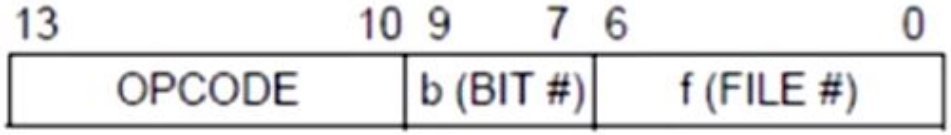
d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Opération littéral et de control



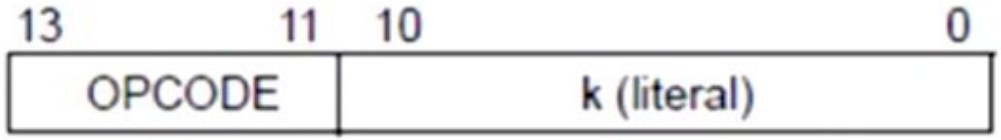
k = 8-bit immediate value

Opération qui manipule un bit



b = 3-bit bit address
f = 7-bit file register address

Les instructions Call et goto



k = 11-bit immediate value

Opcode (Exemple)

Instruction	Opcode	Exemple	
		Instruction	Opcode
ADDWF f,d	00 0111 dfff ffff	ADDWF PORTA,1	00 0111 1000 0101
ANDWF f,d	00 0101 dfff ffff	ANDWF STATUS,0	00 0101 0000 0011
BTFSS f,b	01 11bb bfff ffff	BTFSS OPTIONREG,3	01 1101 1000 0001

Exercice

```
void main() {

   TRISB = 0;                  // All port B pins are configured as
                               // outputs
   PORTB = 0b01010101;         // Logic state on port B pins
}
```

- Remplacer les deux lignes du programme C ci dessus par le code assembleur correspondant

- Donner le code machine correspondant à chaque instruction trouvée

Solution

```
Void main ()
{
TRISB=0;
PORTB=0b01010101;
}
```

```
BCF STATUS,6
BSF STATUS,5
MOVLW  0x00
MOVF    TRISB
```

```
BCF STATUS,6
BCF STATUS,5
MOVLW 0x55
MOVF   PORTB
```

```
Main
    BCF STATUS,6
    BSF STATUS,5
    MOVLW   0x00
    MOVWF  TRISB
    BCF STATUS,6
    BCF STATUS,5
    MOVLW   0x55
    MOVWF  PORTB
END
```

```
Main
    01 0011 0000 0011
    01 1010 1000 0011
    11 00xx  0000 0000
    00 0000 1000 0110
    01 0011 0000 0011
    01 0010 1000 0011
    11 00xx  0101 0101
    00 0000 1000 0110
END
```

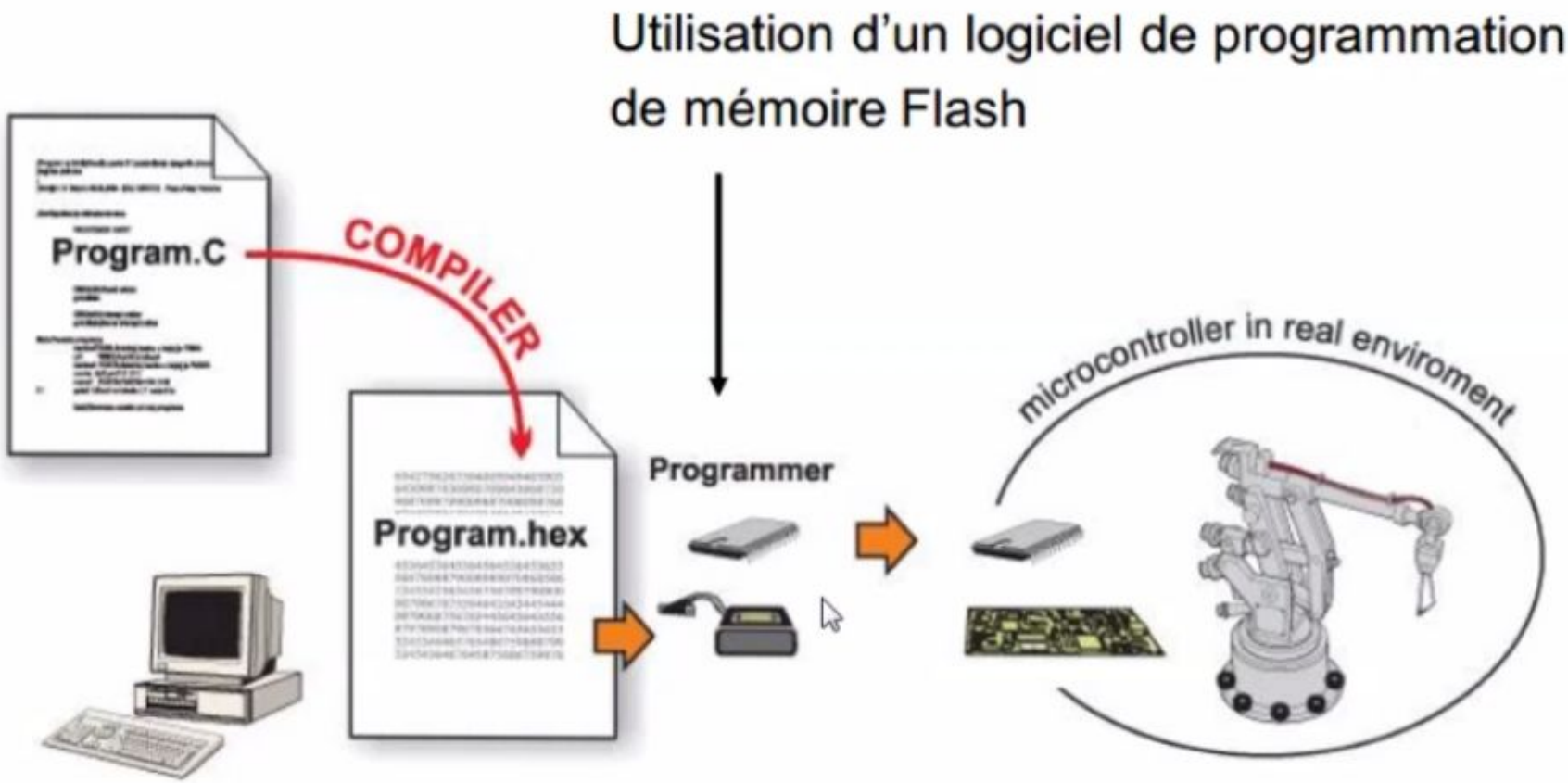
Chaine de compilation

```
void main() {
TRISC=0x00;
while(1) {
        PORTB=0xFF;
        PORTB=0x00;
        }
}
```

```
;Address Opcode          ASM
0x0000        0x2803              GOTO        3
_main:
;test.c,1 ::                      void main() {
;test.c,2 ::                      TRISC=0x00;
0x0003        0x1683              BSF         STATUS, 5
0x0004        0x1303              BCF         STATUS, 6
0x0005        0x0187              CLRF        TRISC
;test.c,3 ::                      while(1) {
L_main0:
;test.c,4 ::                      PORTB=0xFF;
0x0006        0x30FF              MOVLW       255
0x0007        0x1283              BCF         STATUS, 5
0x0008        0x0086              MOVWF       PORTB
;test.c,5 ::                      PORTB=0x00;
0x0009        0x0186              CLRF        PORTB
;test.c,6 ::                      }
0x000A        0x2806              GOTO        L_main0
;test.c,7 ::                      }
L_end_main:
0x000B        0x280B              GOTO        $+0
; end of _main
```

Le compilateur génère aussi un fichier listing (en assembleur), représentant le code et les emplacements mémoire qui seront utilisés.

Chaine de compilation



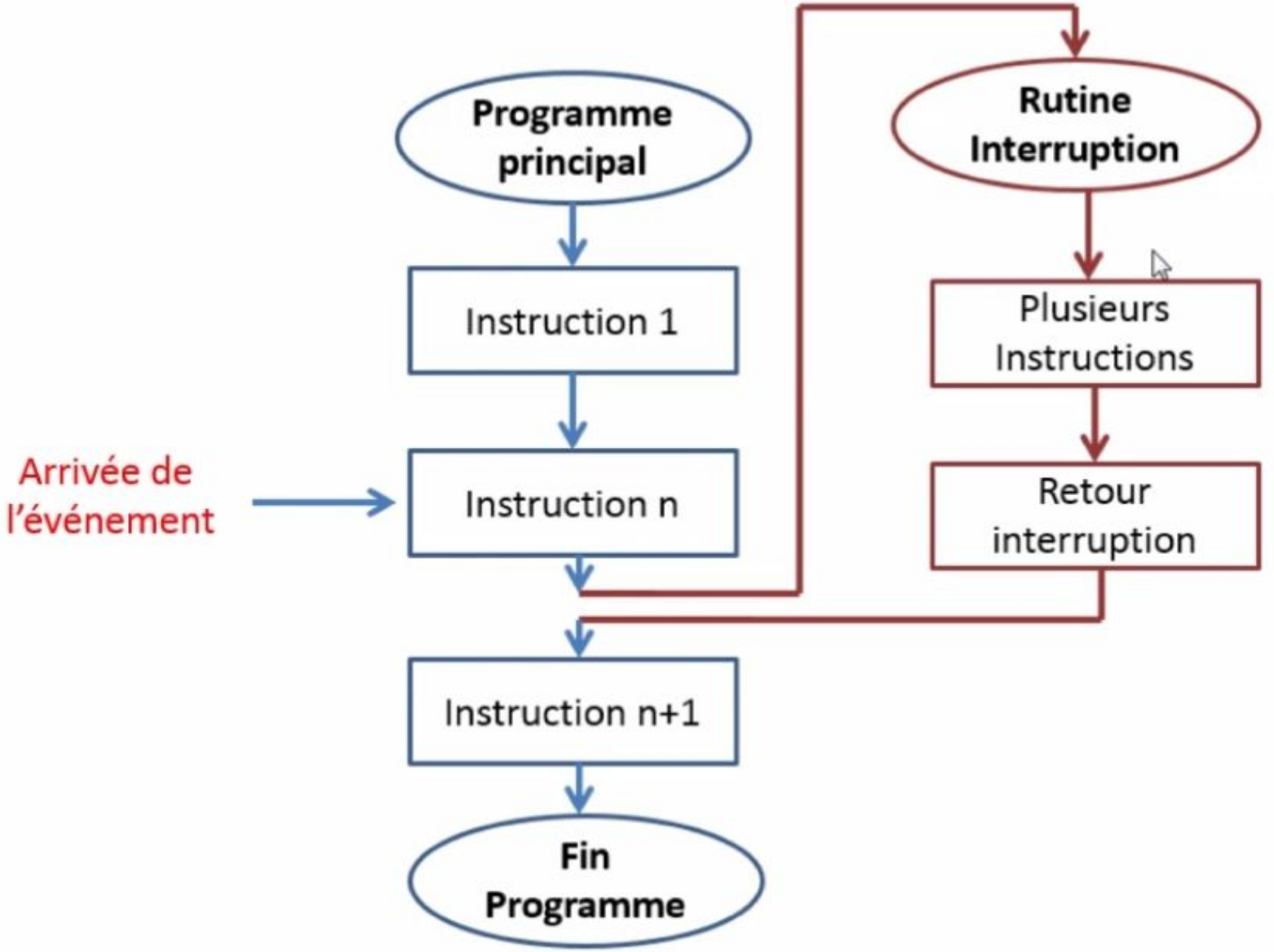
Les Interruptions

L'interruption est une RUPTURE DE SEQUENCE ASYNCHRONE, c'est à dire non synchronisée avec le déroulement normal du programme.

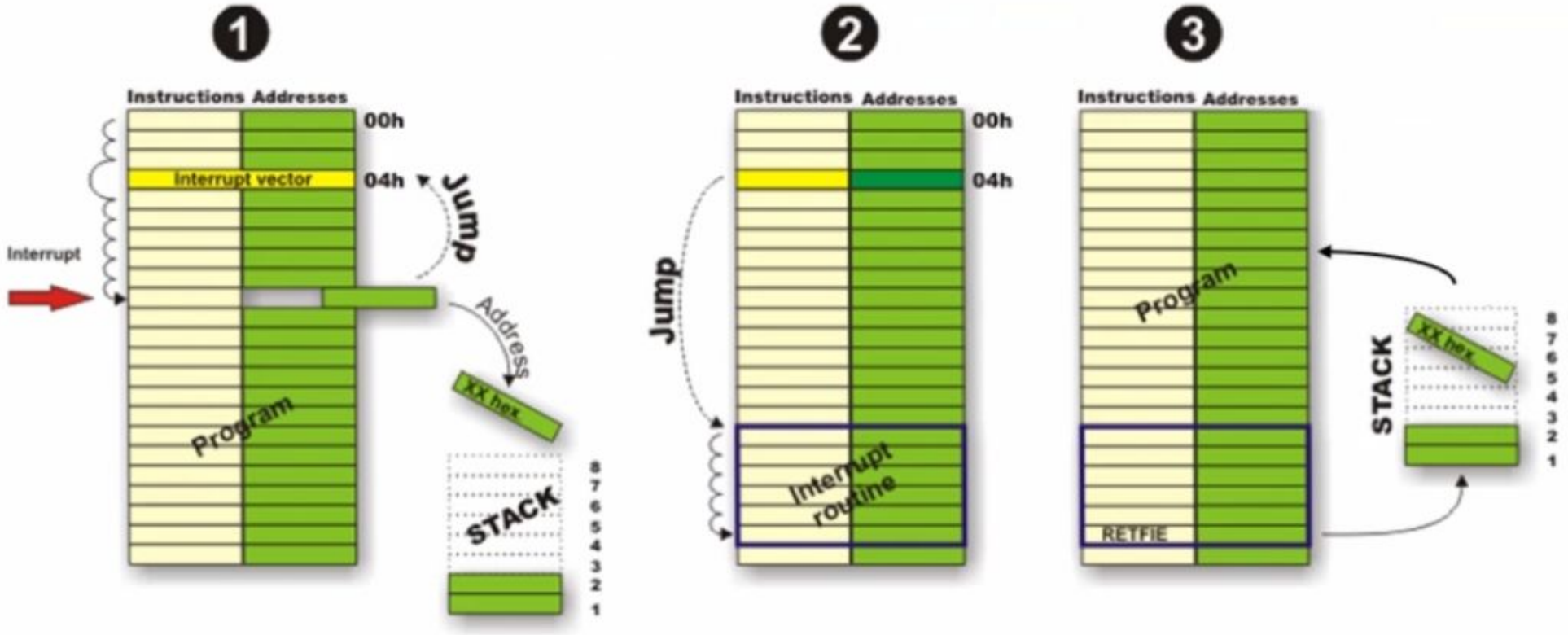
Exemple :

Appel téléphonique reçu lorsque vous jouez avec votre Smartphone

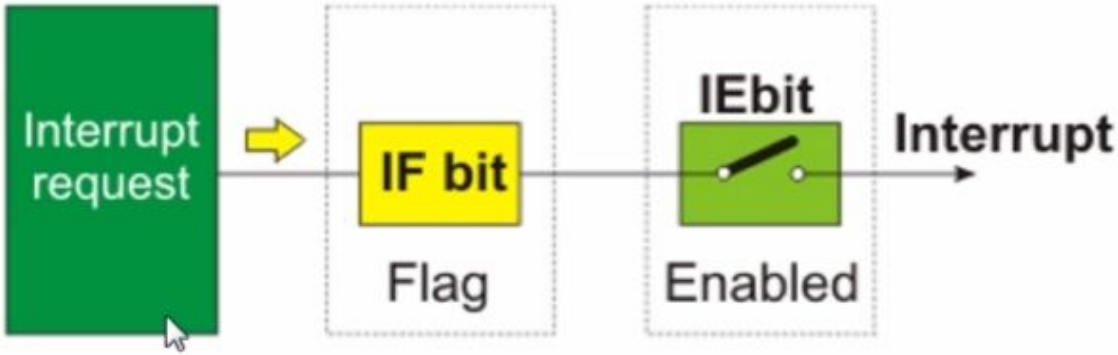
Les Interruptions



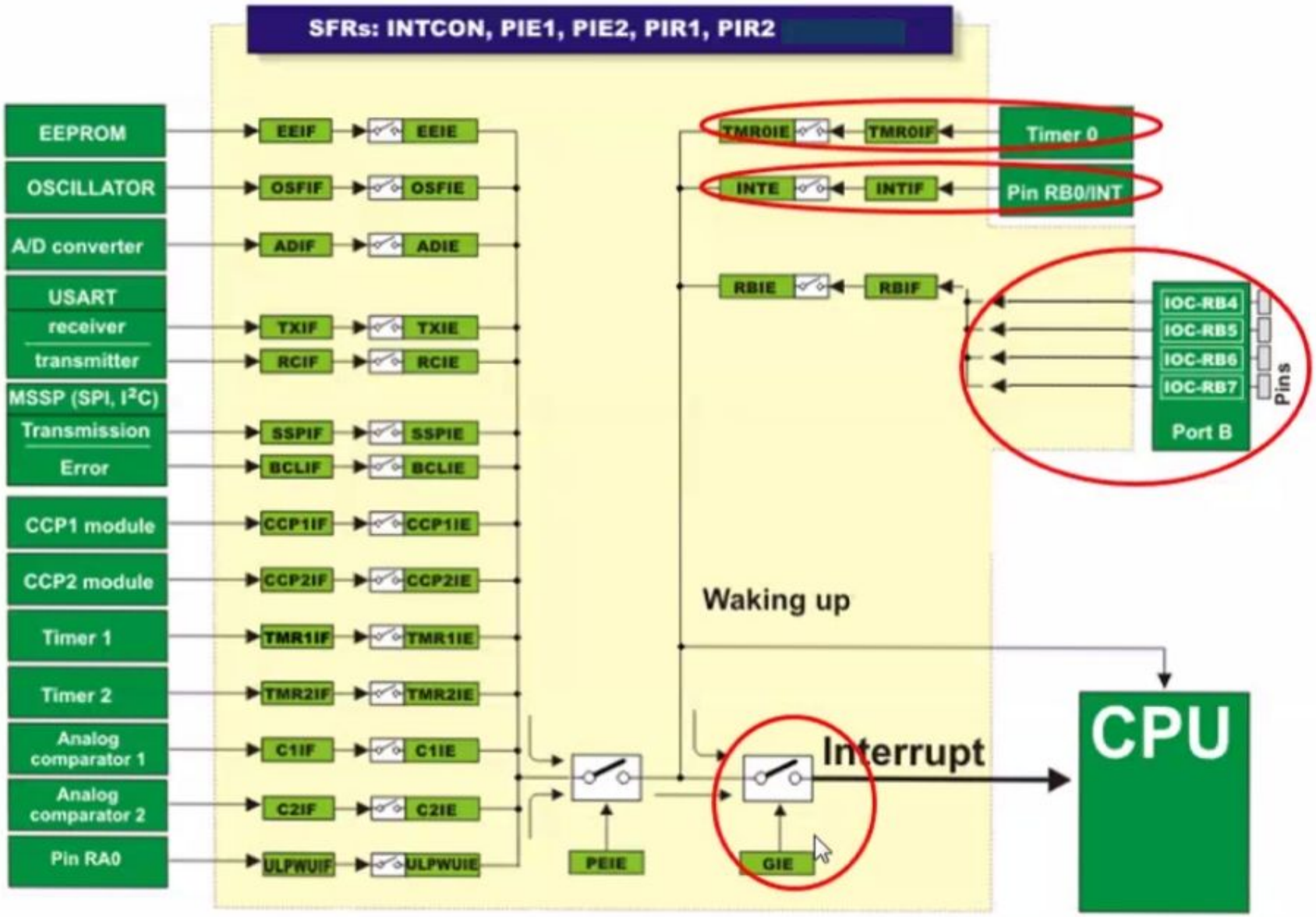
Les Interruptions



Les Interruptions



Les Interruptions



Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

GIE - Global Interrupt Enable bit - Active les interruptions

1 - Activées
0 - Désactivées

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

PEIE - Peripheral Interrupt Enable bit Active les interruptions masquées

1 - Active les interruptions masquées
0 - Désactive les interruptions masquées

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

T0IE - TMR0 Overflow Interrupt Enable bit Active l'interruption sur l'overflow de TMR0.

1 - Enables the TMR0 interrupt.
0 - Disables the TMR0 interrupt.

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

INTE - RB0/INT External Interrupt Enable bit Active l'interruption sur le changement d'état de portB pin 0

1 - Active
0 - Desactive

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

RBIE - RB Port Change Interrupt Enable bit. Active la détection d'un changement au niveaux du PORTB (4-7)quand il est configuré en entrée. génère une interruption

1 - Active la detection
0 - Désactive la détection

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

T0IF - TMR0 Overflow Interrupt Flag bit overflow du TMR0 commence de zero.

1 - TMR0 a fait un overflow
0 - pas d'overflow de TMR0

Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

INTF - RB0/INT External Interrupt Flag bit Flag de INTE

1 - l'interruption INTE a eu lieu
0 - l'interruption INTE n'a pas eu lieu

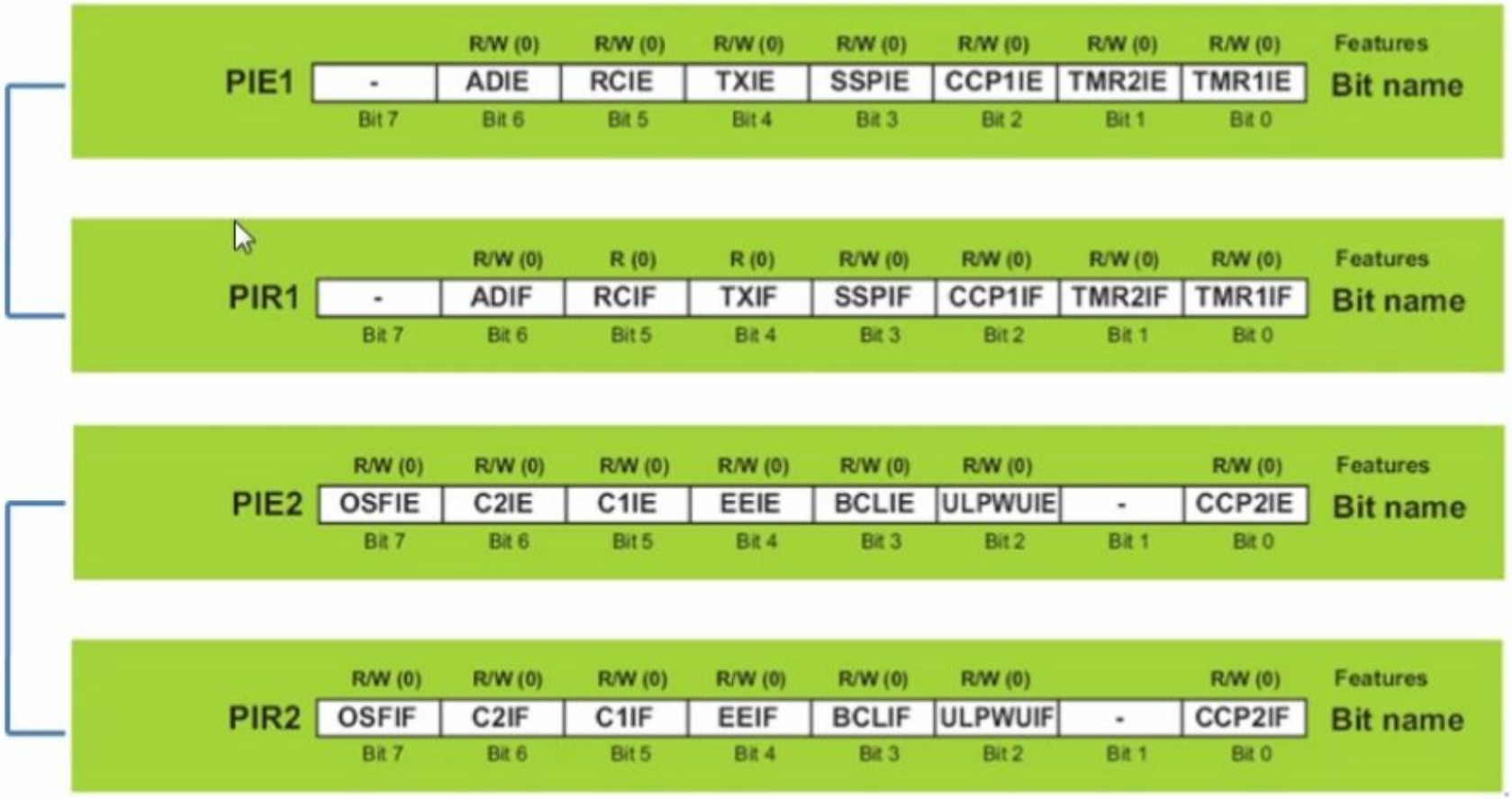
Les Interruption

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

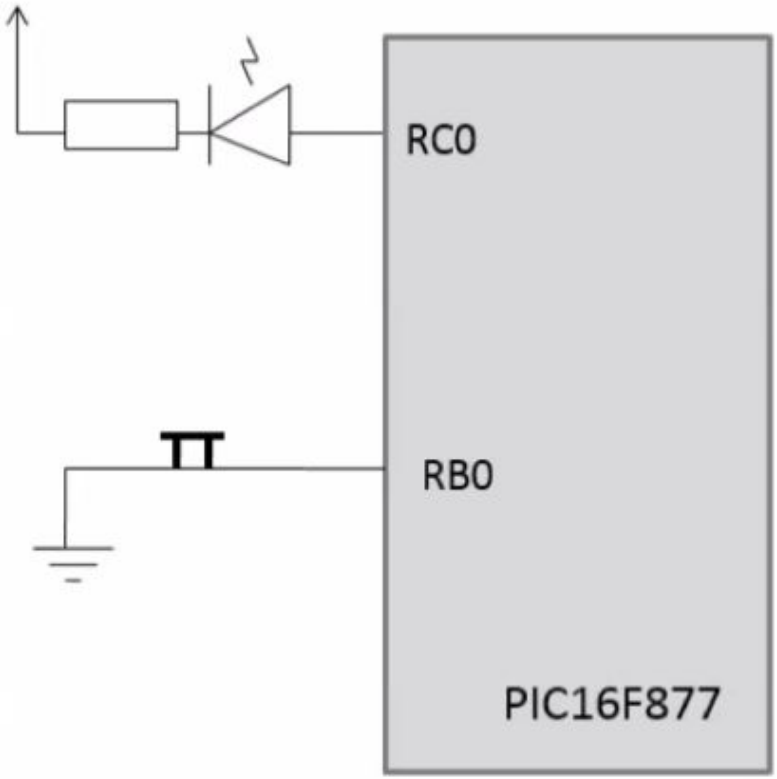
RBIF - RB Port Change Interrupt Flag bit Flag de l'interruption RBIE

1 - Au moins l'un des pins de PORTB a changé
0 - l'interruption n'a pas eu lieu.

Registres spéciaux



Exercice



On veut réaliser un circuit qui :

Quand je presse un bouton connecté à la pin Rb0 lance un timer . Quand le timer atteint son Max il inverse l'état d'une LED. Le circuit est illustré a gauche .

-Quel sont les interruptions qui doivent être autorisés?

-Donner la valeur du registre INTCON en mettant à 0 les bits qui ne sont pas utilisée?

Exercice

Solution

Les interruption qui doivent être autorisés sont:

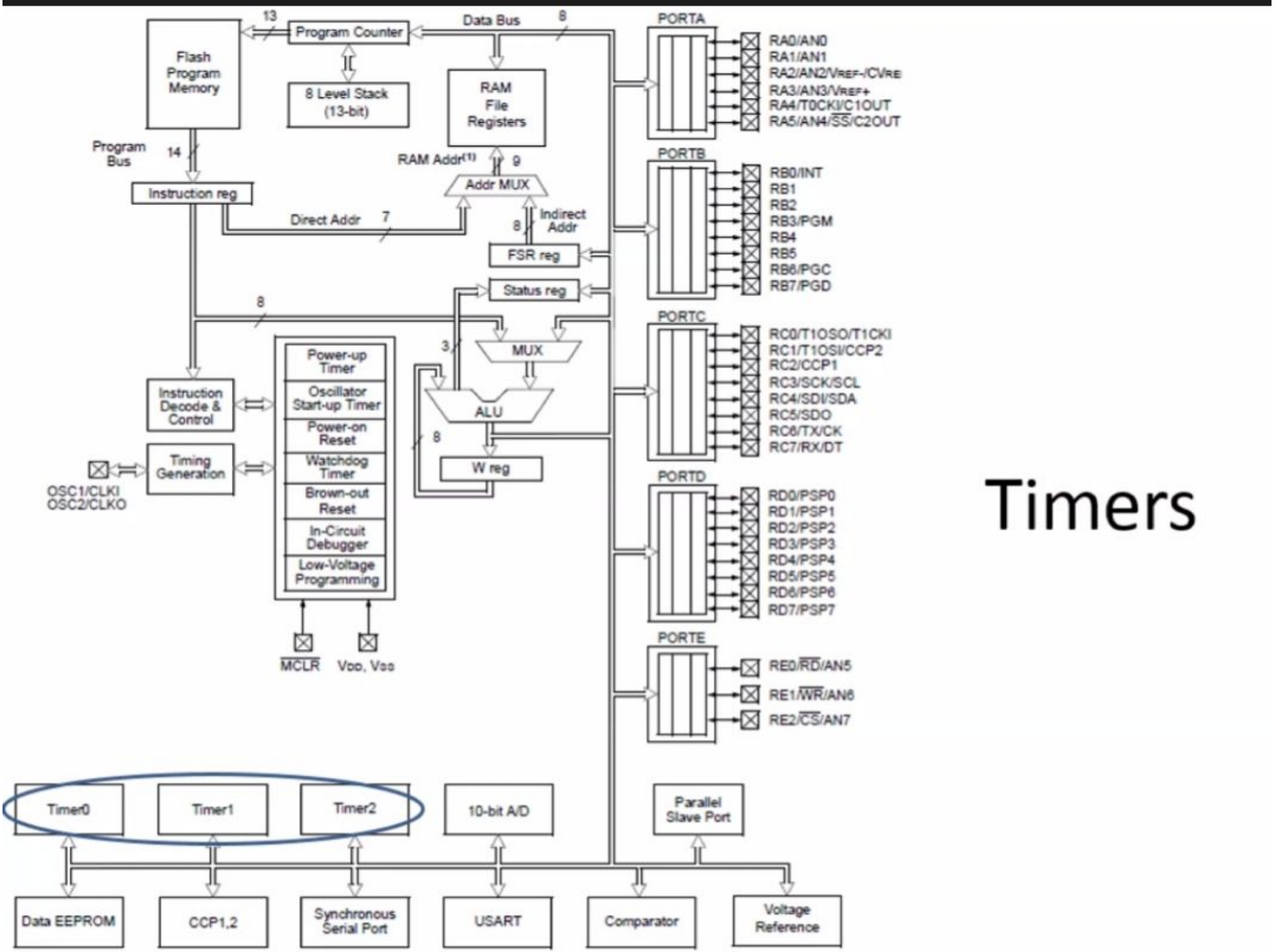
- L'interruption sur le portB pin 0 , pour générer une interruption sur le bouton.
- L'interruption sur le TIMER0 pour temporiser avant de changer l'etat de la LED.

La valeur de INTCON est :

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	1	0	1	1	0	0	0	0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

INTCON=0b 10110000

PIC 16F877A
Timers

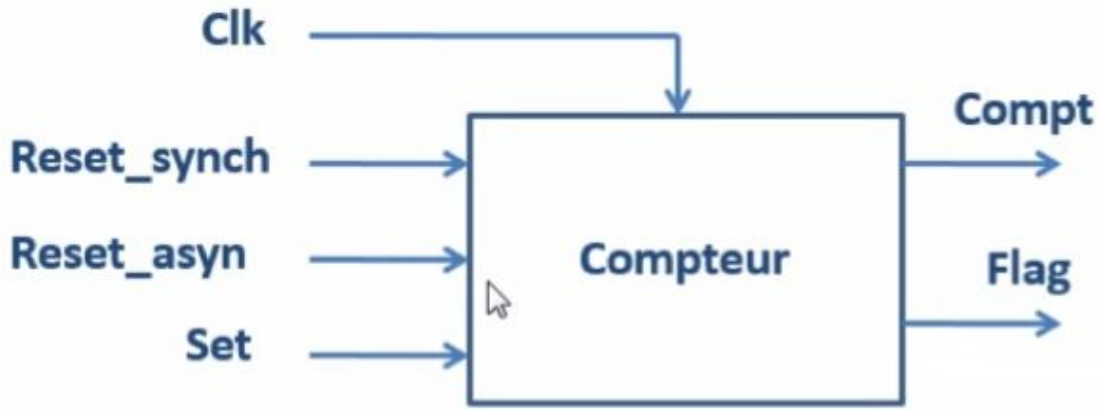


Timers

Timers Principe

Rôle : Réaliser une temporisation

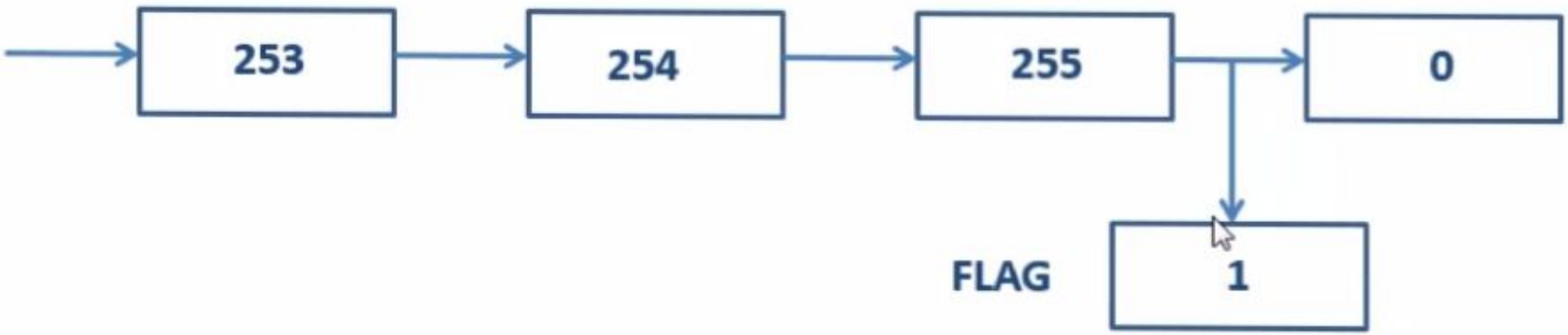
Élément essentiel : Compteur qui s'incrémente à chaque front montant du signal qui lui est appliqué :



Timers Principe

Rôle : Réaliser une temporisation

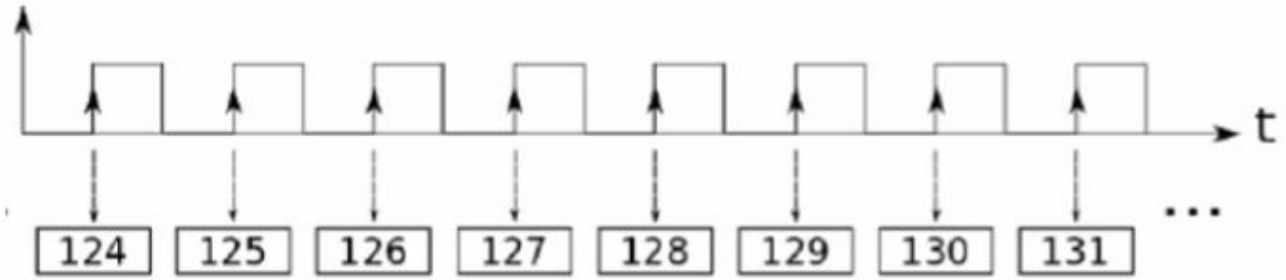
Élément essentiel : Compteur qui s'incrémente à chaque front montant du signal qui lui est appliqué :



Timers Principe

Rôle : Réaliser une temporisation

Élément essentiel : Compteur qui s'incrémente à chaque front montant du signal qui lui est appliqué :



Clk 1 Mhz => T= 1 us => temps calculé = différence Comptage * 1 us

Temps calculé = (131-124)* 1us = 7us

Timers Principe

Deux méthodes pour réaliser une temporisation (1ms ,50 ms ..)

Modifier La fréquence du signal applique au compteur

Le compteur s'incrementera ainsi plus ou moins vite.

Agir sur Le nombre d'impulsions à compter

Le drapeau se lève toujours lorsqu'il y a débordement, on peut donc initialiser le compteur avec une valeur non nulle pour réduire le temps de comptage.

Timers Principe

Première méthode

Modification de la fréquence du signal applique au compteur : le **pré-diviseur (prescaler en** anglais)

Exemple : pour compter 4 fois moins vite

Clk0 1 Mhz => Clk1 250Khz => T= 4 us
=> temps calculé = différence Comptage * 4 us
=> temps calculé = 7*4 us=28 us

Timers Principe

Deuxième méthode

Si le compteur démarre à 0, il mettra beaucoup de temps a atteindre sa valeur maximale.

Pour réduire le temps de comptage, on peut donc charger une valeur initiale non nulle dans le compteur.

Exemple : Valeur initiale égale a 250

Temps calculé = (différence de calcule)* période

Timers du PIC 16F877A

Le 16F887A en compte 3 :

– Timer0 et Timer2 sur 8 bits

– Timer1 sur 16 bits

T= T quartz × Valeur du pré-compteur fixe × Valeur du pré-compteur réglable × Nombre d'impulsions à compter

$F_{quart_3} \rightarrow \frac{1}{4} \rightarrow \frac{1}{N} \rightarrow F_{Timer} = \frac{F_\varphi}{4 \times N}$

$\rightarrow T_{Tim} = \frac{4 \times N}{F_\varphi} = 4 \times N \times T_\varphi$

$\rightarrow T_{global} = T_{timer} \times Nbr_{periodes}$

$(2^{Tb} - Valeur\ init)$

Le Timer 0 (TMR0)

Le timer 0 est le premier timer parmi 3 dans le PIC 16F877A .

Caractéristique de TMR0 :

- compteur sur 8 bits
- compteur à valeur initiale réglable entre 0 et 255
- Le présalaire qui lui est associé peut prendre les paramètre de 1 jusqu'à 256

Le Timer 0 (TMR0)

Le maximum de temps que peut calculer le TMR0 est obtenu avec le prescalair à 256 , et la valeur d'initialisation de 0.

Avec une horloge de 4 Mhz on obtient

$$Tmax= 1/(4*10^{6})*4*256*(255-0+1)=65,536ms$$

Exercice : Temporisation de 10 ms

$T = 10.10^{-3} = (1/(4.10^{6})) \times 4 \times X \times Y$ avec

X= valeur du pré-compteur

y= valeur à compter=256- valeur initiale

Soit $X \times Y = 10000$

X	Y	Y arrondi	Valeur initiale (256-Y arrondi)
1	10 000	Impossible	
2	5 000	Impossible	
4	2 500	Impossible	
8	1 250	Impossible	
16	625	Impossible	
32	312,5	Impossible	
64	156,25	156	256-156=100
128	78,125	78	256-78=178
256	39,0625	39	256-39=217

Pré-diviseur =64
Valeur à compter=156
Soit valeur init =100

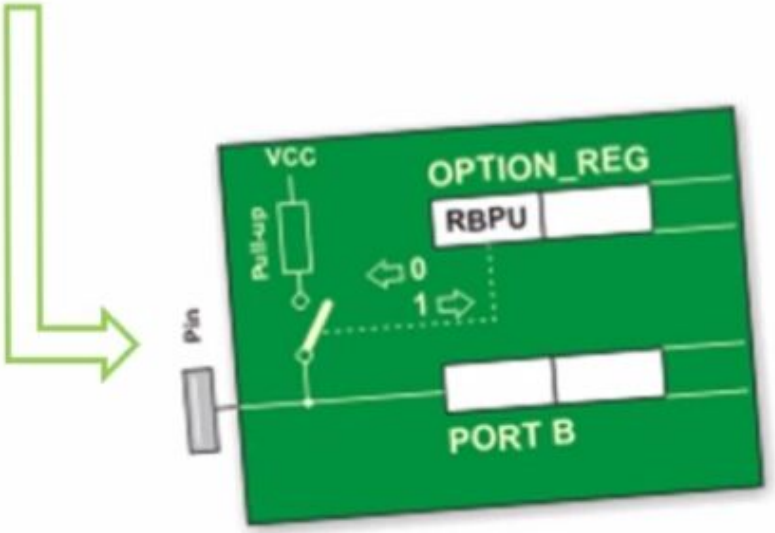
64×156=9984
128×78=9984
256×39=9984

PIC16F877 Le Timer 0

- REGISTRES necessaire :
  - OPTION REG
  - INTCON
  - TMRO
  - TRISA

Registre OPTION REG

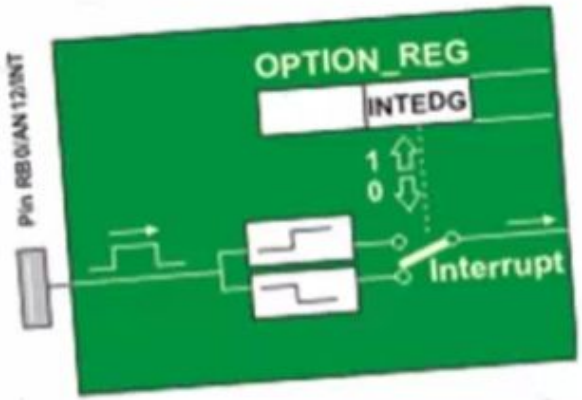
	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	



RBPU: Resistance de Pull Up
1: Désactivé
0: Activé

Registre OPTION REG

	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	



INTEDG: front de l'interruption sur RB0
1: Front montant
0: Front descendant