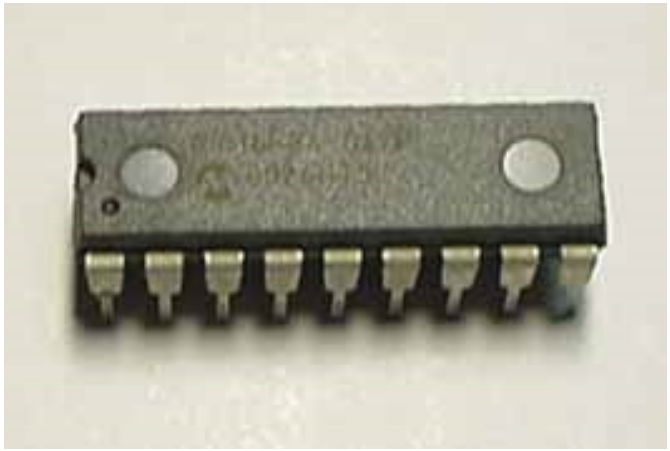
Architecture et fonctionnement du microcontrôleur PIC 16F84

Objectifs

Le but de ce chapitre est de :

- Présentation générale du micro-contrôleur PIC 16F84.
- Architecture interne du micro-contrôleur PIC 16F84.
- Principe de fonctionnement.

Présentation du PIC



Présentation du PIC

- Un PIC est un microcontrôleur de MICROCHIP *
- PIC: Peripheral Interface Controller
- Caractérisé par:
  - Séparation des mémoires de programmes et de données.
  - Communication avec l'extérieur seulement avec des ports.
  - Utilisation d'un jeu d'instructions réduit: RISC (**Reduced instruction set computer**)

* www.microchip.com

Identification des PICs

- Un PIC est identifié par un numéro de la forme suivante: xx(L) XXyy-zz (Exp : 16F84A)
  - xx: famille du composant.
  - L : Tolérance plus importante de la plage de tension .
  - XX: Type de mémoire de programme:
    - C: EPROM ou EEPROM.
    - CR: PROM.
    - F: Flush.
  - yy: Identification.
  - zz: Vitesse maximale du quartz.

Les familles des PICs

- La famille PIC 16F dispose de 3 sous-familles :

1. Base-Line : Les instructions sont codées sur **12 bits**.
2. Mid-Range, qui utilise des mots de **14 bits**.
3. High-End, qui utilise des mots de **16 bits.**

On s' intéresse dans ce cours à la famille **Mid-Range**.

Exemple

- Dans ce module, on va étudier le microcontrôleur PIC 16F 84 -10.

  - 16: Mid-Range (14 bits).
  - F: mémoire Flush.
  - 84: Type.
  - 10: Quartz à 10 MHZ au maximum.

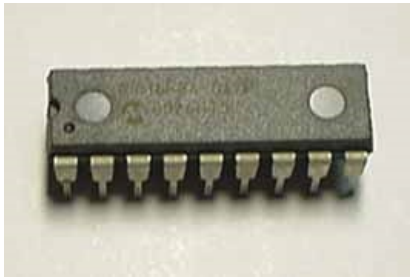
.

Pourquoi un PIC

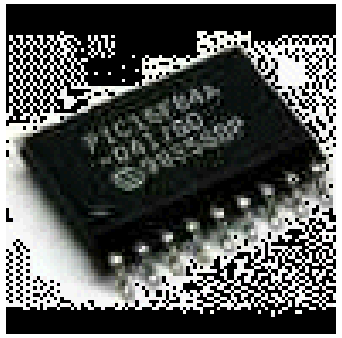
1. Les performances sont identiques voir supérieurs à ses concurrents.
2. Les prix sont les plus bas du marché.
3. Très utilisé donc très disponible.
4. Les outils de développement sont gratuits et téléchargeables sur le WEB.
5. Le jeu d'instruction réduit est souple, puissant et facile à maîtriser.
6. Les versions avec mémoire flash présentent une souplesse d'utilisation et des avantages pratiques indéniables .

Brochage

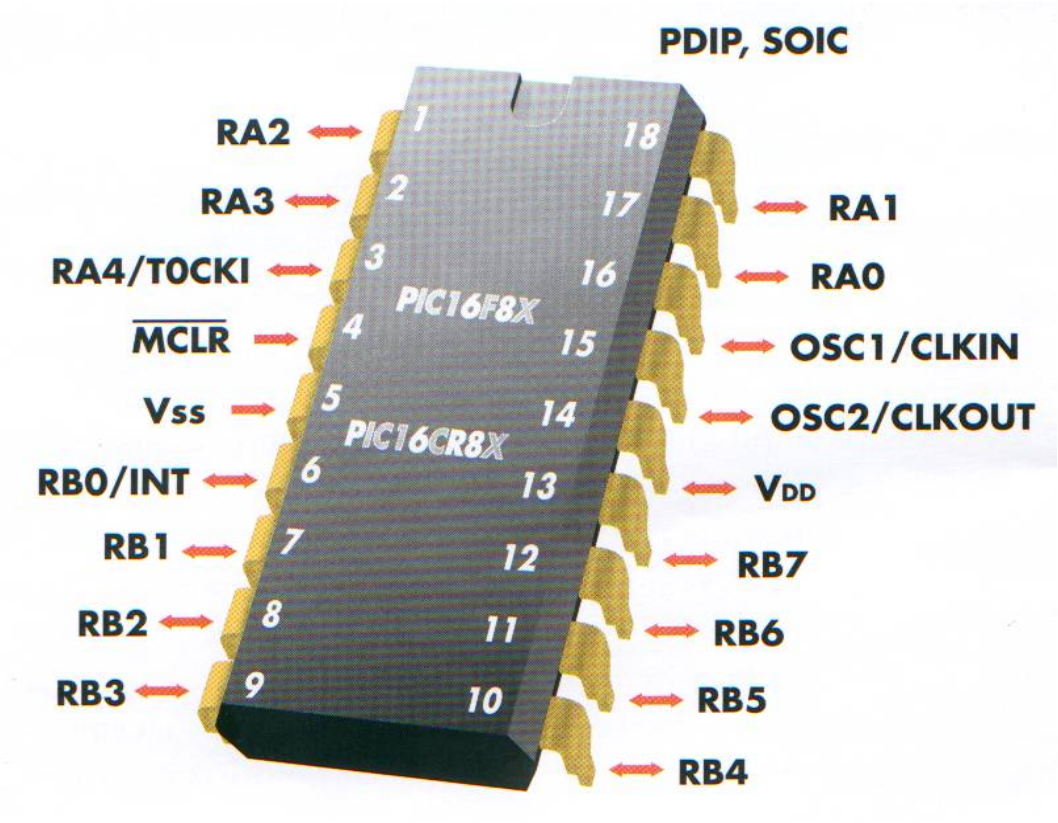
- Le PIC 16F84 est un circuit intégré de 18 broches.
- Le PIC 16F84 peut être présenté sous forme d'un:
  - Boitier DIP (Plastic Dual In-line Package) à 18 broches



  - Boitier SOIC (Small Outline Integrated Circuit)

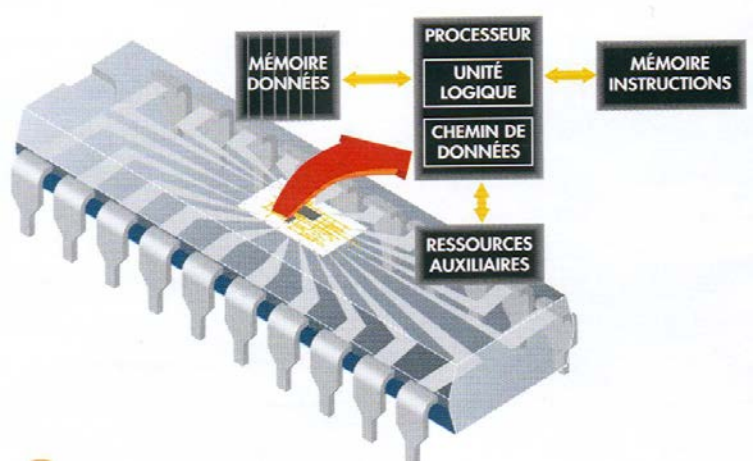


Brochage

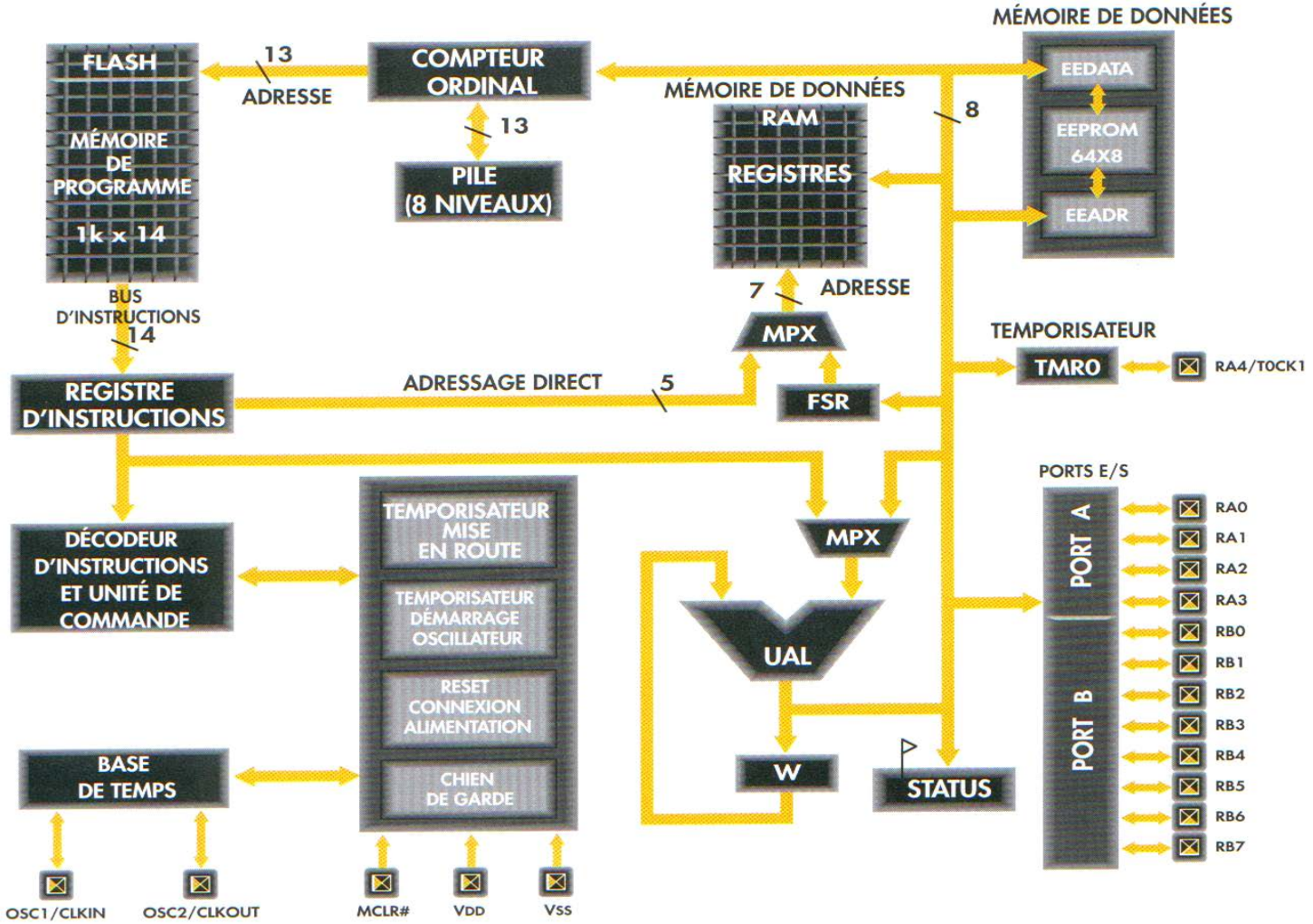

PDIP, SOIC
PIC16F8X
PIC16CR8X
RA2
RA3
RA4/T0CKI
MCLR
Vss
RB0/INT
RB1
RB2
RB3
RA1
RA0
OSC1/CLKIN
OSC2/CLKOUT
VDD
RB7
RB6
RB5
RB4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

Architecture Interne

- La structure générale du PIC 16F84 comporte 4 blocs :
  - Mémoire de programme,
  - Mémoire de données,
  - Processeur,
  - Ressources auxiliaires (périphériques).

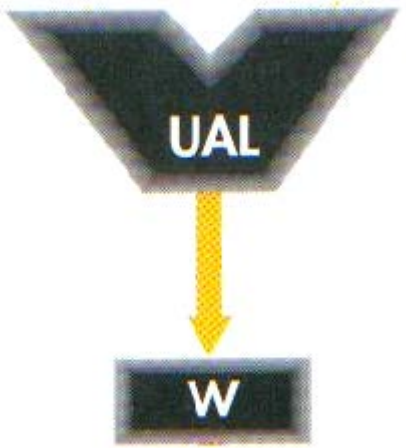


Architecture interne



Processeur

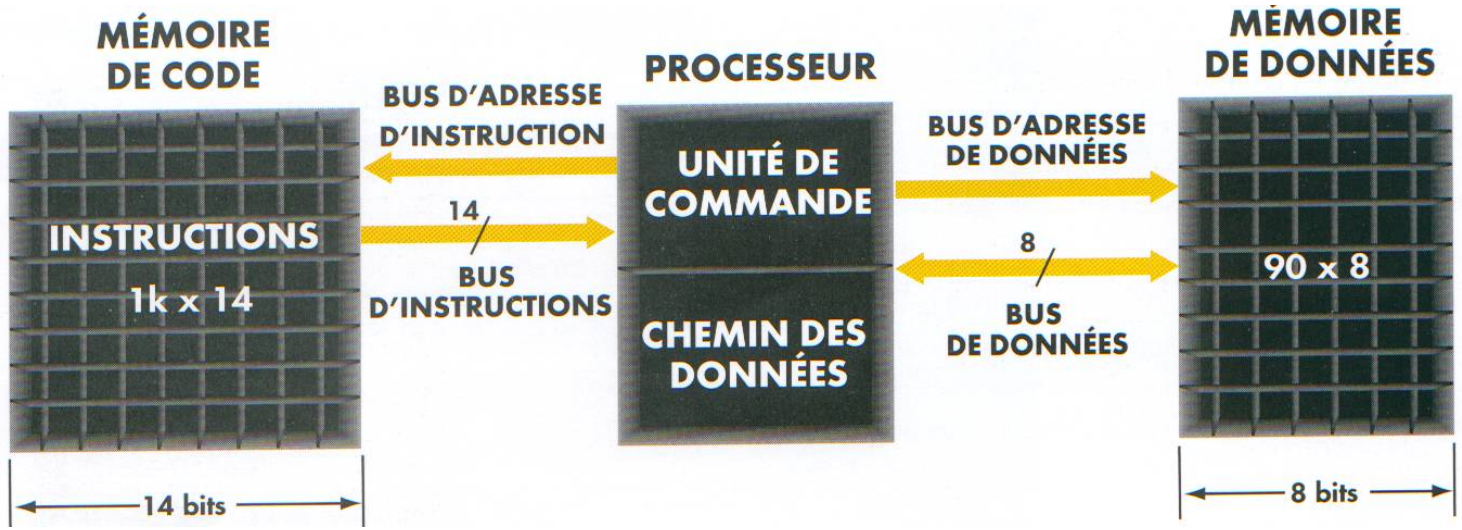
- Le processeur est composé en deux parties:
  - Unité arithmétique et logique(UAL): Chargé d' exécuter les opérations arithmétiques (addition, soustraction, division,…) et logiques (ET, Non, AND,….).
  - Registre de Travail noté W.



Ressources auxiliaires

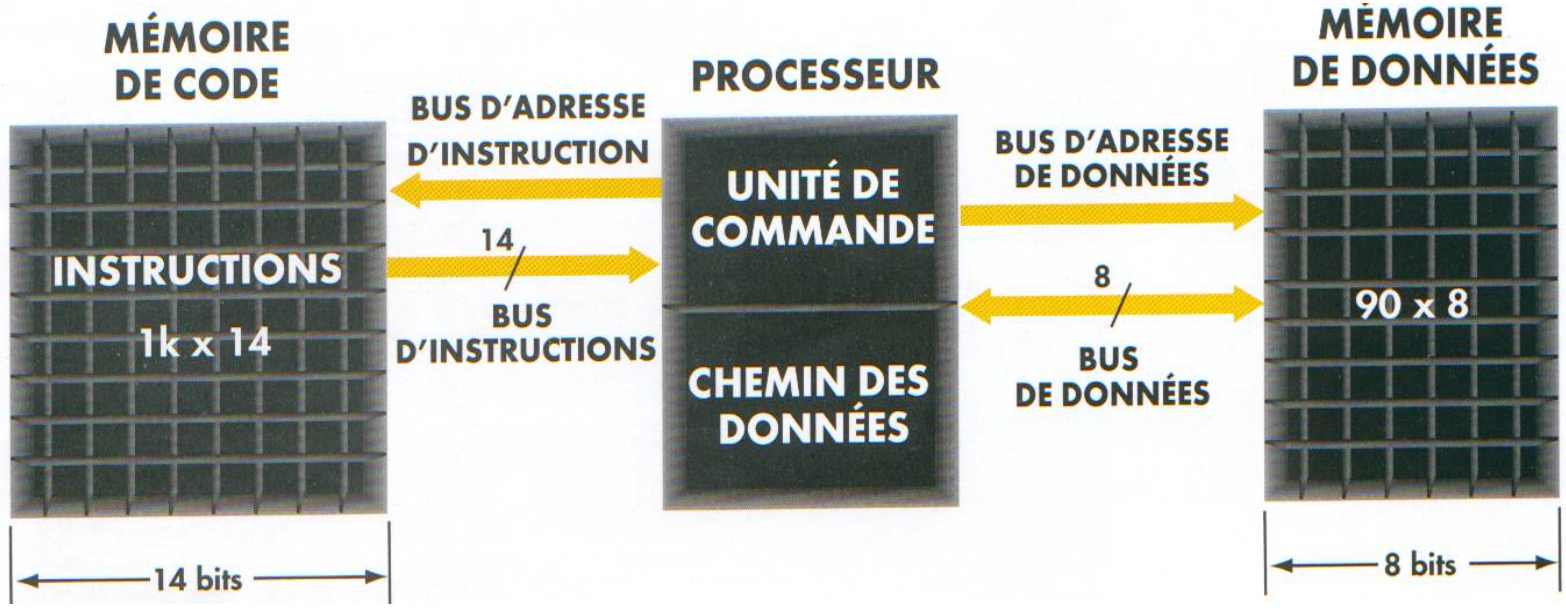
- Les ressources auxiliaires dans le PIC 16F84 sont:
  - Ports d'entrées et de sorties.
  - Temporisateur.
  - Interruptions.
  - Chien de garde.
  - Mode sommeil.

Organisation de la mémoire



Deux mémoires

- Le PIC 16F84 est conçu selon l'architecture de Harvard donc il possède 2 mémoires:
  - Mémoire de programme.
  - Mémoire des données.



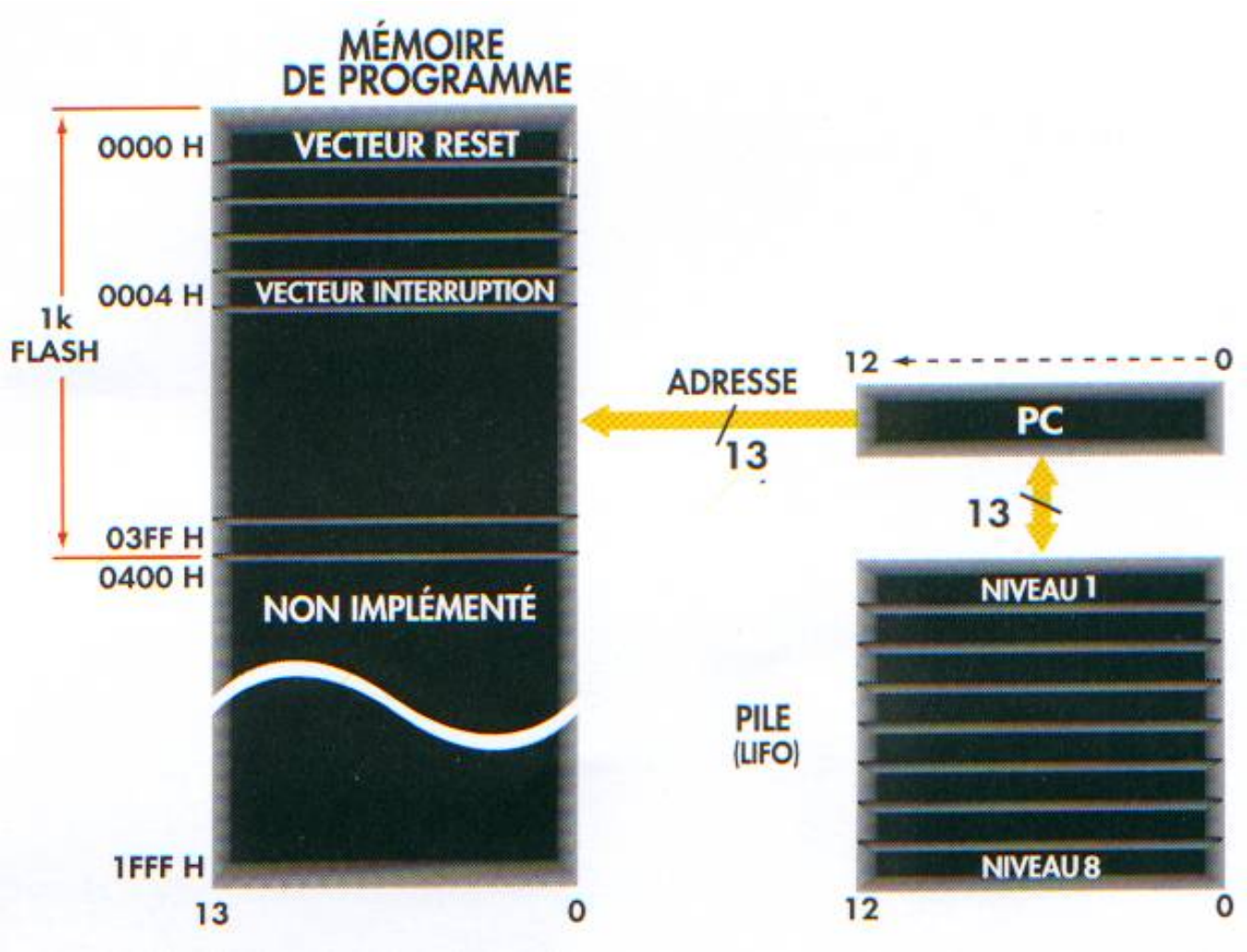
Mémoire de programme

- La mémoire programme est constituée de 1K mots de 14 bits(1024 emplacements).
- Elle contient le programme à exécuter .
- C'est une mémoire non volatile de type FLASH.
- La technologie utilisée permet plus de 1000 cycles d'effacement et de programmation

Mémoire de programme

- Le PIC 16F84 possède un compteur ordinal qui permet d'adresser 8K * 14 bits.
- L'adresse 0000h contient le vecteur du reset.
- l'adresse 0004h l'unique vecteur d'interruption du PIC,
- La pile contient 8 valeurs: Ce sont des zones réservées par le système(pas d'adresse).

Mémoire de programme



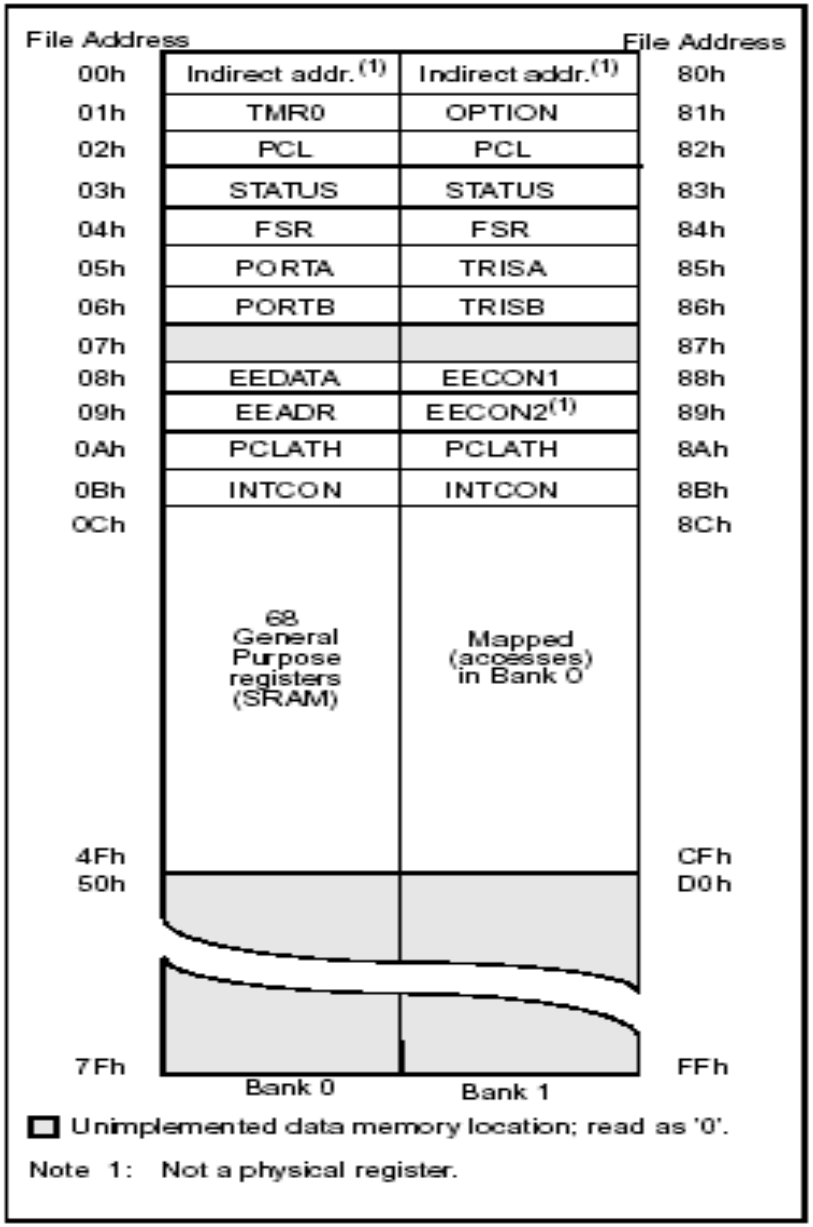
Mémoire de données

- Séparée en deux espaces:
  1. Mémoire RAM de 68 octets.
     - Mémoire volatile.
     - Mot mémoire: 8 bits.RAM
     - Elle contient deux partie:
       - Les SFR (Special Function Registers) qui permet de contrôler les opérations sur le circuit.
       - La seconde partie contient des registres généraux (GPR: General Purpose Registers) , libres pour l'utilisateur.
  2. Mémoire EEPROM de 64 octets.
     - Accessible pour lecture et écriture.
     - Non volatile.
     - Plus lente que la RAM.
     - On y accède à l'aide des registres EEADR et EEDATA sous le contrôle de deux registres de contrôle EECON1 et EECON2.

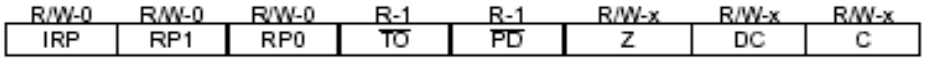
Mémoire de données

- La mémoire donnée est divisée en deux banques:
  - La banque 0 est sélectionnée en mettant le bit RP0 du registre STATUS a 0.
  - La banque 1 est sélectionnée en mettant le bit RP0 du registre STATUS a 1.
- Chaque banque est composée de 128 octets.
- Les 12 premières ligne de chaque banques sont réservées pour les SFR (Special Function Registers).

Mémoire de données



Registre STATUS



R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
0 = Bank 0, 1 (00h - FFh)
1 = Bank 2, 3 (100h - 1FFh)
The IRP bit is not used by the PIC16F8X. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16F8X. RP1 should be maintained clear.

bit 4: **$\overline{TO}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3: **$\overline{PD}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions) (For $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

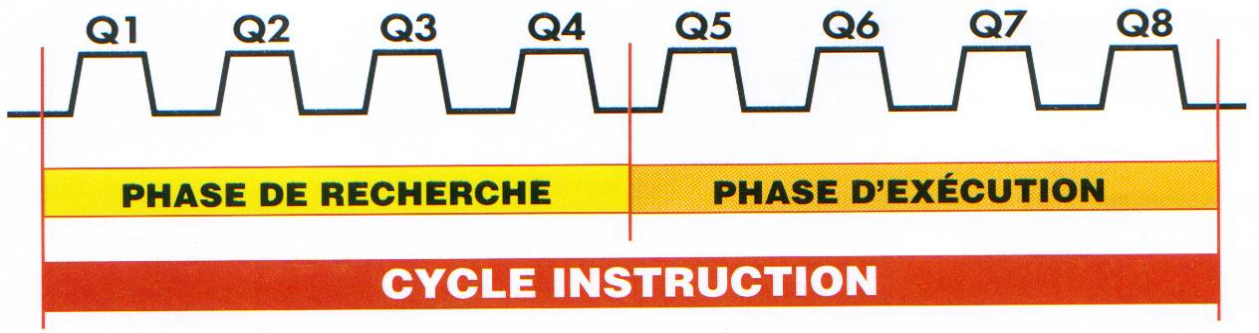
Mémoire EEPROM de données

- La mémoire EEPROM de données est constituée de 64 octets que l'on peut lire et écrire depuis un programme.
- Ces octets sont conservés après une coupure de courant et sont très utiles pour conserver des paramètres semi permanents.
- On y accède à l'aide des registres EEADR et EEDATA .
- Deux registres de contrôle (EECON1 et EECON2) sont associés à la mémoire EEPROM

Principe de fonctionnement

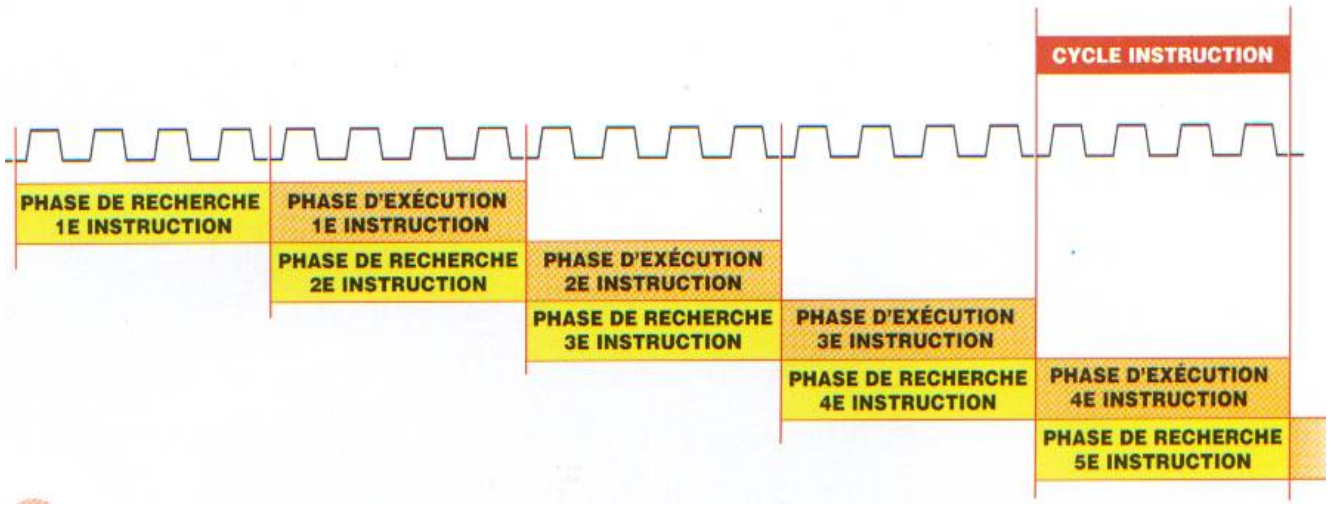
Principe de fonctionnement du PIC

- Un microcontrôleur exécute des instructions.
- "le cycle instruction " : le temps nécessaire à l'exécution d'une instruction.
- Une instruction est exécutée en deux phases :
  - Phase de recherche du code binaire de l'instruction stocké dans la mémoire de programme.
  - Phase d'exécution de l'instruction.

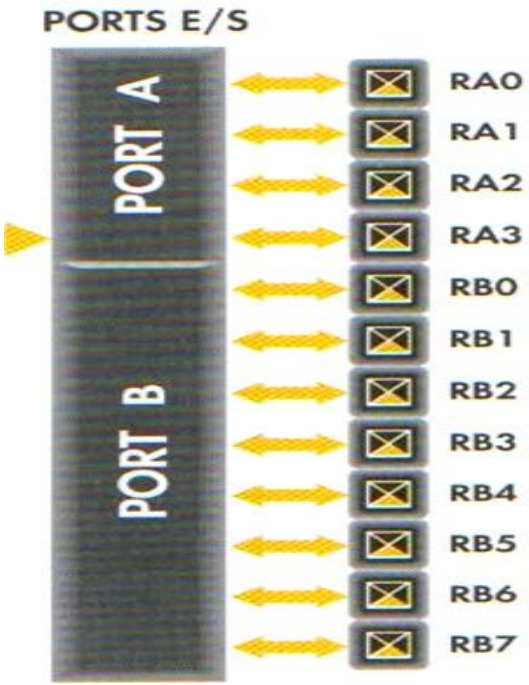


Principe de fonctionnement du PIC

- Normalement l'exécution d'un instruction dure 8 cycles d'horloges.
- L'architecture particulière des PICs(Bus différents pour les données et le programme) lui permet de réduire ce temps par deux.

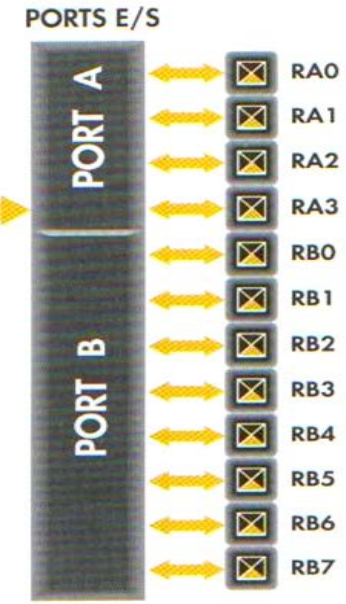


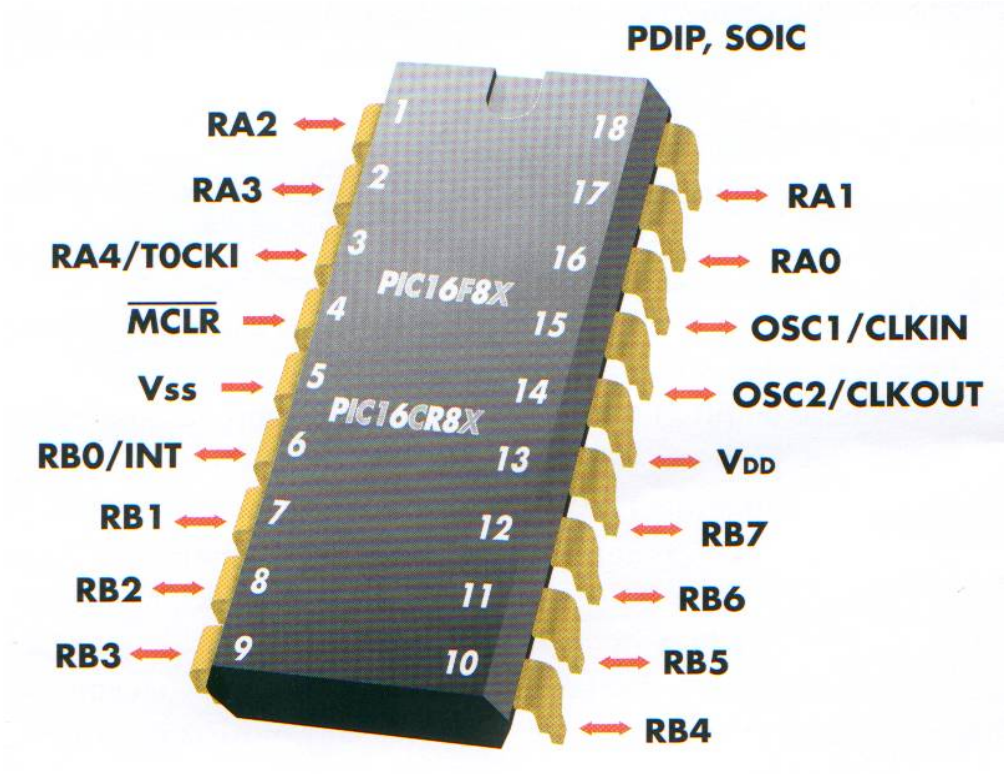
Les ports d'Entrée/Sortie



Ports d' Entrée / Sortie

- Le 16F84 possède 2 ports d'entrée sortie:
  - Port A (PORTA).
  - Port B (PORTB).





PORTA et registre TRISA

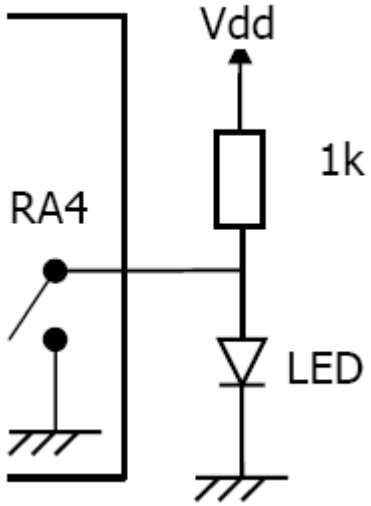
- Le port A désigné par PORTA est un port de 5 bits (RA0 à RA4).
- La configuration de direction pour chaque bit du port est déterminée avec le registre TRISA.
- Bit *i de TRISA = 0* → *bit i de PORTA configuré en* ***sortie***
- Bit *i de TRISA = 1* → *bit i de PORTA configuré en* ***entrée***

PORTA et registre TRISA

- La broche RA4 est multiplexée avec l'entrée horloge du Timer TMR0.
- Elle peut être utilisée soit:
  - comme E/S normale du port A,
  - comme entrée horloge pour le Timer TMR0
- le choix se fait à l'aide du bit T0CS du registre OPTION_REG.

PORTA et registre TRISA

- RA4 est une E/S à drain ouvert, si on veut l'utiliser comme sortie (pour allumer une LED par exemple), il ne faut pas oublier de mettre une résistance externe vers Vdd.
  - Si RA4 est positionnée à 0, l'interrupteur fermé, la sortie est reliée à la masse.
  - Si RA4 est placée à 1, l'interrupteur est la sortie est déconnectée d'où nécessite de la résistance externe.



LED rouges de 20mA (tension de seuil égale à 1.8V).

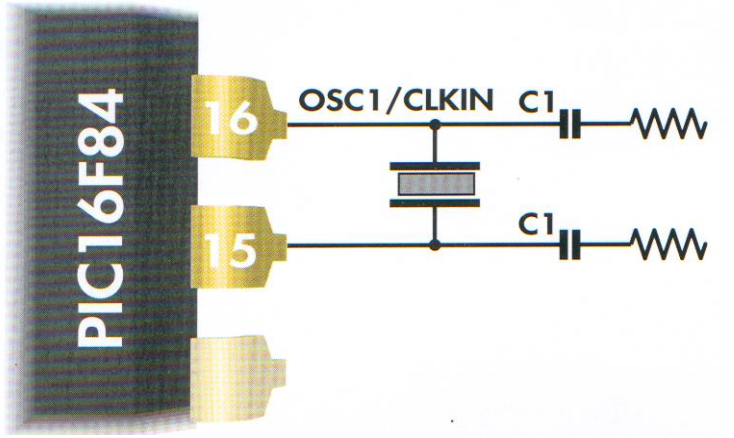
PORTB et registre TRISB

- Le port B désigné par PORTB est un port bidirectionnel de 8 bits (RB0 à RB7).
- La configuration de direction se fait à l'aide du registre TRISB.
- Même configuration que le PORTA.
- En entrée, la ligne RB0 appelée aussi INT peut déclencher l'interruption externe INT.
- En entrée, une quelconque des lignes RB4 à RB7 peut déclencher l'interruption RBI.

Mise en œuvre

Mise en œuvre

- L'utilisation et la mise en œuvre très simple des PICs les a rendus extrêmement populaire.
- Pour faire fonctionner le PIC, il suffit de:
  - Alimenter le circuit par ses deux broches VDD et VSS,
  - fixer sa vitesse de fonctionnement à l'aide d'un quartz,
  - d'élaborer un petit système pour permettre de réinitialiser le microcontrôleur sans avoir à couper l'alimentation.



Horloge

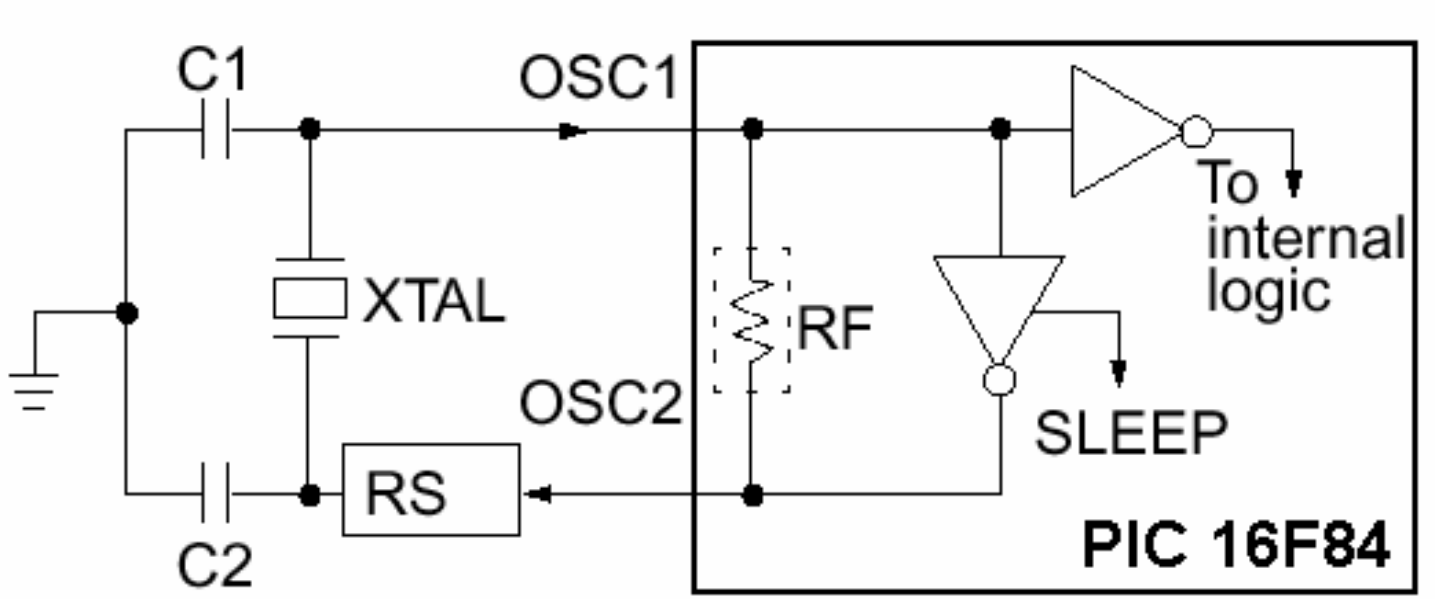
- L'horloge peut être soit interne soit externe ou interne.
- L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC.

- Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 4, 10 ou 20 MHz selon le type de microcontrôleur.

- Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser le PIC sur un processus particulier.

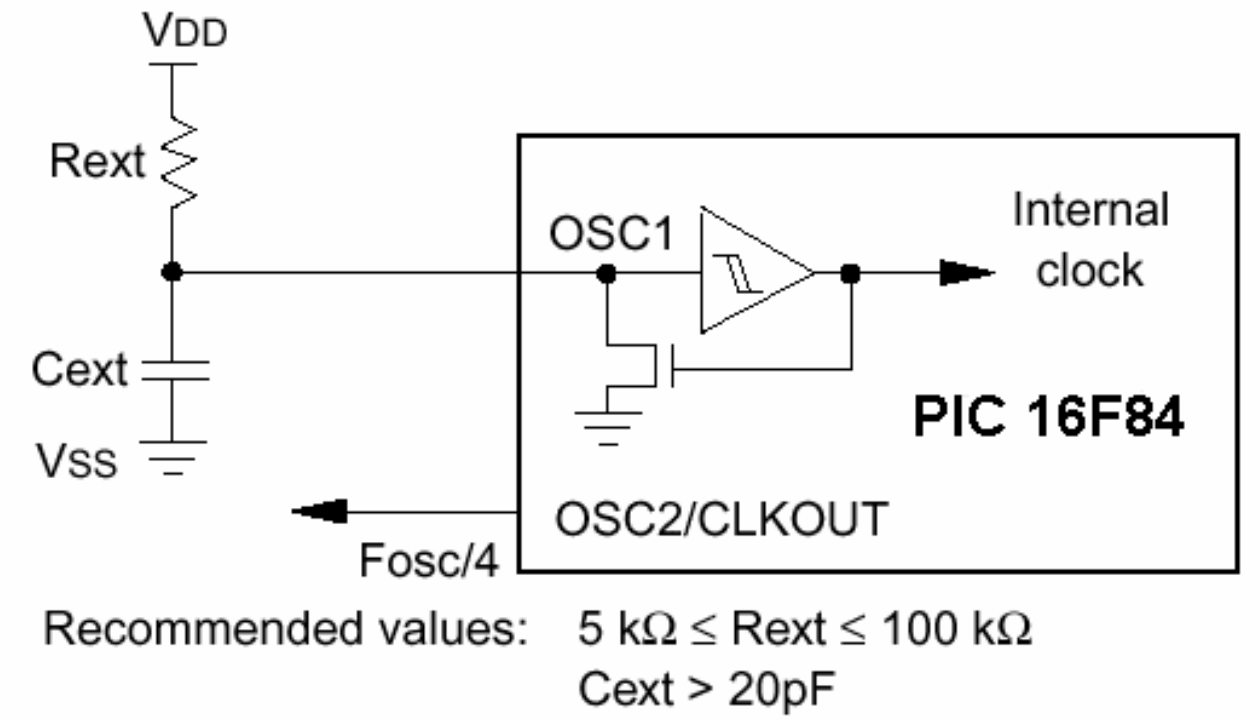
Horloge

- Oscillateur a quartz



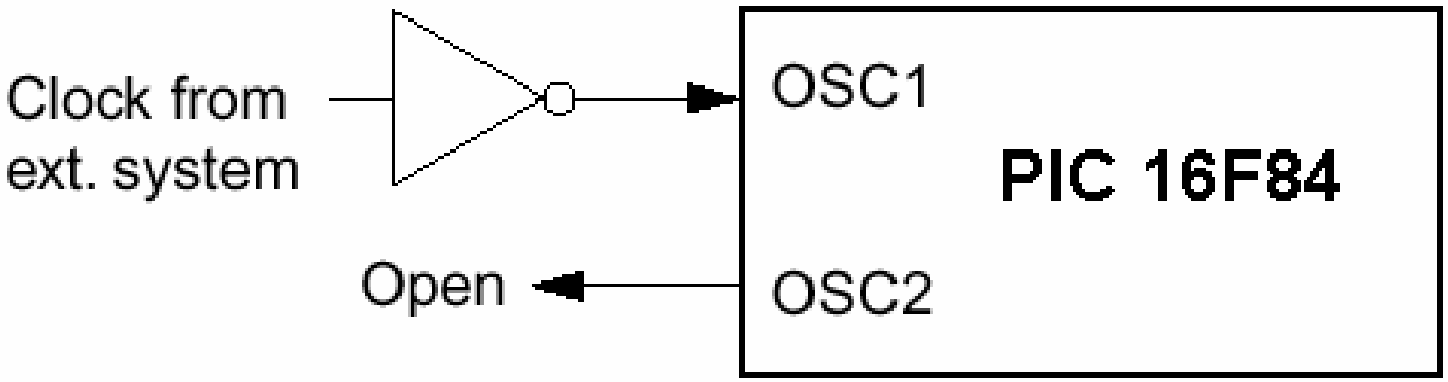
Horloge

- Oscillateur RC



Horloge

- Horloge externe



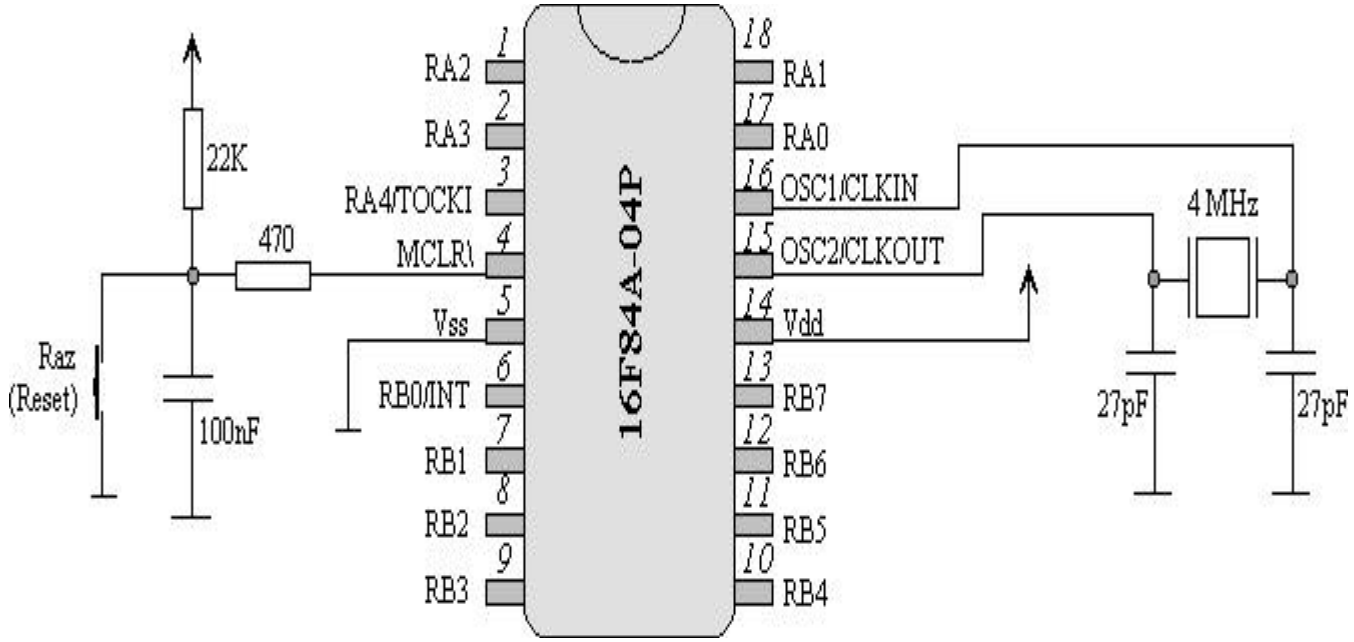
Horloge

- Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4.

- Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs

Reset

- Le reset se fait généralement via l'entrée MCLR (broche 4).



Codage binaire et hexadécimal, complément à deux

- On rappelle tout d'abord les différentes bases qui nous seront utiles :
- Le **binaire**(base2) est constitué de 2 chiffres: **0, 1**
- l'**octal** (base8), est constitué de 8 chiffres :

  0, 1, 2, 3, 4, 5, 6, 7
- Le **décimal** (base 10), est constitué de 10 chiffres :

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- l'**hexadécimal** (base16), est constitué de 16 chiffres :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- **Remarque** : pour connaître la base associée à un nombre, on le note entre parenthèse avec en indice une lettre b, o, d ou h selon qu'il s'agit d'un codage binaire, octal, décimal ou hexadécimal. Par exemple, $(1001)_b$, $(3F1)_h$ ou $(128)_d$.

Codes pondérés

Dans une base donnée, le nombre s'exprime comme une *somme pondérée*. Par exemple, le nombre 128 **décimal** (base10) est constitué de 3 chiffres :

- Le chiffre 8 est affecté du poids de1 (unités)
- Le chiffre 2 est affecté du poids de 10 (dizaines)
- Le chiffre1est affecté du poids de 100 (centaines)

Le nombre peut donc s'écrire

$$1 \times 100 + 2 \times 10 + 8 \times 1 = (\mathbf{128})_d$$

Chiffre

Poids

Codes pondérés

Le nombre 10 **binaire** (base2) est constitué de 2 chiffres :

- Le chiffre 0 est affecté du poids de 1 = 2^0
- Le chiffre 1 est affecté du poids de 2=2^1

- Le nombre peut donc s'écrire

$$1 \times 2 + 0 \times 1 = (\mathbf{10})_b$$

Chiffre

Poids

Codes pondérés

Le nombre 1F8 **hexadécimal** (base16) est constitué de 3 chiffres :

- Le chiffre 8 est affecté du poids de 1
- Le chiffre F est affecté du poids de 16
- Le chiffre 1 est affecté du poids de 16^2=256

- Le nombre peut donc s'écrire

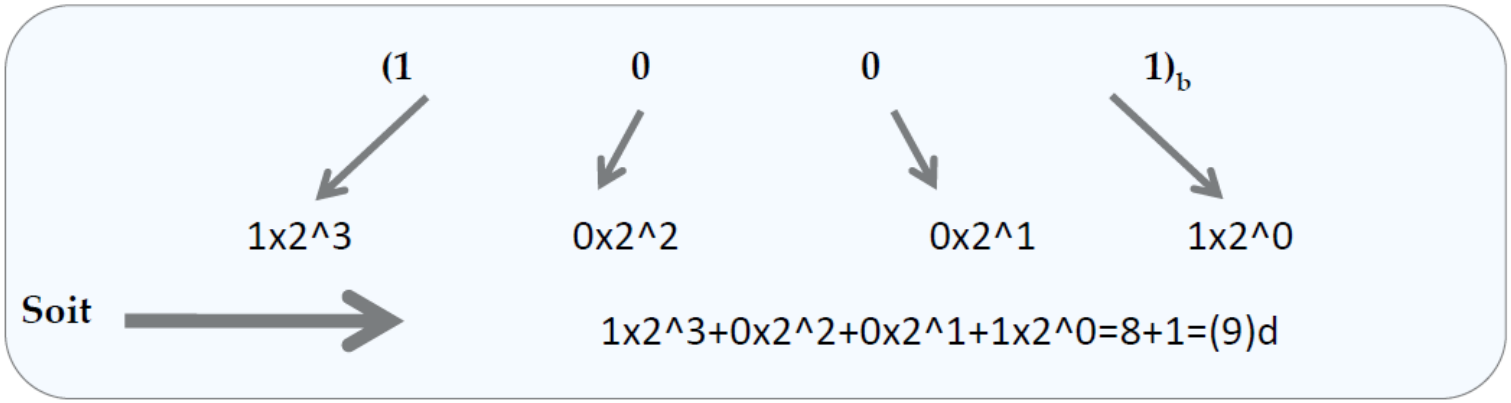
$$1 \times 256 + F \times 16 + 8 \times 1 = (\mathbf{1F8})h$$

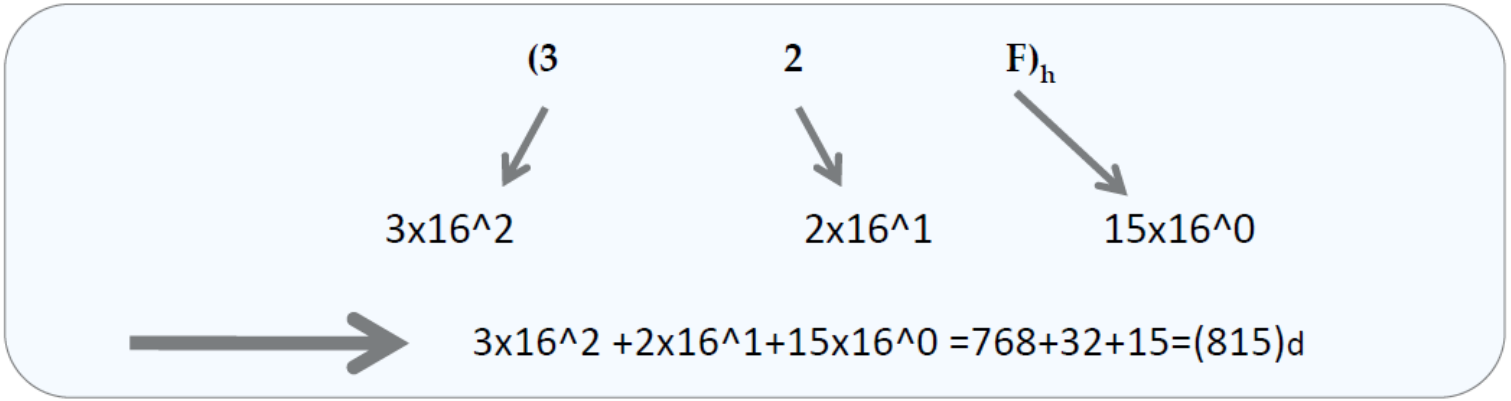
Conversion binaire-hexadécimal : le codage hexadecimal a été créé afin d'alléger l'exploitation des nombres binaires. Il permet en particulier une **conversion simple** par *regroupement des bits par 4 en partant de la droite,* chaque paquet étant alors simple à convertir :

0001	1111	1000	
1	**F**	**8**	= $(1F8)_h$

Conversion

Conversion en décimal : développement en somme de puissances de la base





Conversion décimal ⟹ binaire : division par 2 successives...

$(14)_d = (1\ 1\ 1\ 0)_b$

$(14)_d$	2
0	7 | 2
1	3 | 2
1	1

Sens de lecture...

Conversion décimal ⟹ hexadécimal : division par 16 successives...

$(282)_d = (11A)_h$

$(282)_d$	16
10=A	17 | 16
1	1

Sens de lecture...

Opérations arithmétiques binaires, complément à deux

Les techniques de calcul des opérations arithmétiques *peuvent être transposées* du décimal au binaire.

- **Addition:** V=A+B, ***Exemple*** : (0110)b + (0101)b = (1011)b

- **Multiplication:** V=AxB, ***Exemple*** **:** (0110)b . (0101)b = (011110)b

Une multiplication (division) par 2 correspond à un décalage à gauche (à droite).

- **Soustraction:** V=A–B,

Pour calculer V, on calcule la somme entre A et le *complément à deux* de B

Complément à deux : remplacer les 1 par des 0 (et vice-versa), puis ajouter 1.

Exemple : $(110)_b - (010)_b = (100)_b$

$(010)_b$ donne $(101)_b + (001)_b = (110)_b$

Les instructions du 16F84

- Tous les PICs Mid-Range ont un jeu de 35 instructions. Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (OC) ainsi que l'opérande. A part les instructions de saut, toutes les instructions sont exécutées en un cycle d'horloge. 4 types :

1. **Les instructions « orientées octet »**
2. **Les instructions « orientées bits »**
3. **Les instructions opérant sur une donnée**
4. **Les instructions de saut et appel de procédures**

- Sachant que l'horloge fournie au PIC est prédivisée par 4, si on utilise par exemple un quartz de 4MHz, on obtient donc 1000000 cycles/seconde, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d' Instructions Par Seconde). Avec une horloge de 20MHz, on multiplie par 5.

Les instructions « orientées octet »

Ce sont des instructions qui manipulent les données sous forme d'octets. Elles sont codées de la manière suivante :

- 6 bits pour l'instruction : logique, car comme il y a 35 instructions, il faut 6 bits pour pouvoir les coder toutes
- 1 bit (d) pour indiquer si le résultat obtenu doit être conservé dans le registre de travail (accumulateur) W de l'unité de calcul (W pour Work) ou sauvé dans un registre F (F pour File).
- Reste 7 bits pour encoder l'adresse de l'opérande (128 positions au total)

- Problème ! 7 bits ne donnent pas accès à la mémoire RAM totale, donc voici l'explication de la division de la RAM en deux banks. Pour remplacer le bit manquant, on utilise le bit RP0 du registre STATUS.
- Bien qu'on ne l'utilise pas sur le 16F84, le bit RP1 est aussi réservé pour le changement de bank, le 16F876 par exemple possède 4 banks.

Les instructions « orientées bits »

Ce sont des instructions destinées à manipuler directement les bits d'un registre d'une case mémoire. Elles sont codées de la manière suivante : -

- 4 bits pour l'instruction
- 3 bits pour indiquer le numéro du bit à manipuler (de 0 à 7)
- 7 bits pour indiquer l'opérande.

Les instructions opérant sur une donnée

Ce sont les instructions qui manipulent des données qui sont codées dans l'instruction directement. Elles sont codées de la manière suivante :

- L'instruction est codée sur 6 bits
- Elle est suivie d'une valeur IMMEDIATE codée sur 8 bits (donc de 0 à 255).

Les instructions de saut et appel de procédures

Ce sont les instructions qui provoquent une rupture dans la séquence de déroulement du programme. Elles sont codées de la manière suivante :

- Les instructions sont codées sur 3 bits
- La destination est codée sur 11 bits

INSTRUCTIONS OPERANT SUR REGISTRE (direct)			indicateurs	Cycles
ADDWF	**F,d**	W+F → {W,F ? d}	C,DC,Z	1
ANDWF	**F,d**	W and F → {W,F ? d}	Z	1
CLRF	**F**	Clear F	Z	1
CLRW		Clear W	Z	1
CLRWDT		Clear Watchdoc timer	TO', PD'	1
COMF	**F,d**	Complémente F → {W,F ? d}	Z	1
DECF	**F,d**	décrémente F → {W,F ? d}	Z	1
DECFSZ	**F,d**	décrémente F → {W,F ? d} skip if 0		1(2)
INCF	**F,d**	incrémente F → {W,F ? d}	Z	1
INCFSZ	**F,d**	incrémente F → {W,F ? d} skip if 0		1(2)
IORWF	**F,d**	W or F → {W,F ? d}	Z	1
MOVF	**F,d**	F → {W,F ? d}	Z	1
MOVWF	**F**	W → F		1
RLF	**F,d**	rotation à gauche de F a travers C → {W,F ? d}	C	1
RRF	**F,d**	rotation à droite de F a travers C → {W,F ? d}		1
SUBWF	**F,d**	F – W → {W,F ? d}	C,DC,Z	1
SWAPF	**F,d**	permute les 2 quartets de F → {W,F ? d}		1
XORWF	**F,d**	W xor F → {W,F ? d}	Z	1

INSTRUCTIONS OPERANT SUR BIT				
BCF	**F,b**	RAZ du bit b du registre F		1
BSF	**F,b**	RAU du bit b du registre F		1
BTFSC	**F,b**	teste le bit b de F, si 0 saute une instruction		1(2)
BTFSS	**F,b**	teste le bit b de F, si 1 saute une instruction		1(2)

INSTRUCTIONS OPERANT SUR DONNEE (Immediat)				
ADDLW	**K**	W + K → W	C,DC,Z	1
ANDLW	**K**	W and K → W	Z	1
IORLW	**K**	W or K → W	Z	1
MOVLW	**K**	K → W		1
SUBLW	**K**	K – W → W	C,DC,Z	1
XORLW	**K**	W xor K → W	Z	1

INSTRUCTIONS GENERALES				
CALL	**L**	Branchement à un sous programme de label L		2
GOTO	**L**	branchement à la ligne de label L		2
NOP		No operation		1
RETURN		retourne d'un sous programme		2
RETFIE		Retour d'interruption		2
RETLW	**K**	retourne d'un sous programme avec K dans W		2
SLEEP		se met en mode standby	TO', PD'	1

{W,F ? d} signifie que le résultat va soit dans W si d=0 ou w, soit dans F si d= 1 ou f

Exemples d'instruction

MOVWF F ; recopie W dans le registre d'adresse F

(W → F)

F (File) désigne l'adresse de n'importe quel registre SFR ou GPR. Pour les registres SFR, on peut utiliser leurs noms à condition d'inclure le fichier p16F84.inc dans le programme

MOVWF 0x2C ; recopie W dans la case mémoire d'adresse 2Ch
MOVWF EEDATA ; recopie W dans le registre EEDATA

MOVF 0x08 ; recopie W dans le registre EEDATA

Exemples d'instruction

ADDWF f,d

W+f → W (d=0) ou → f (d=1); [C,Z,DC]

BCF f,b

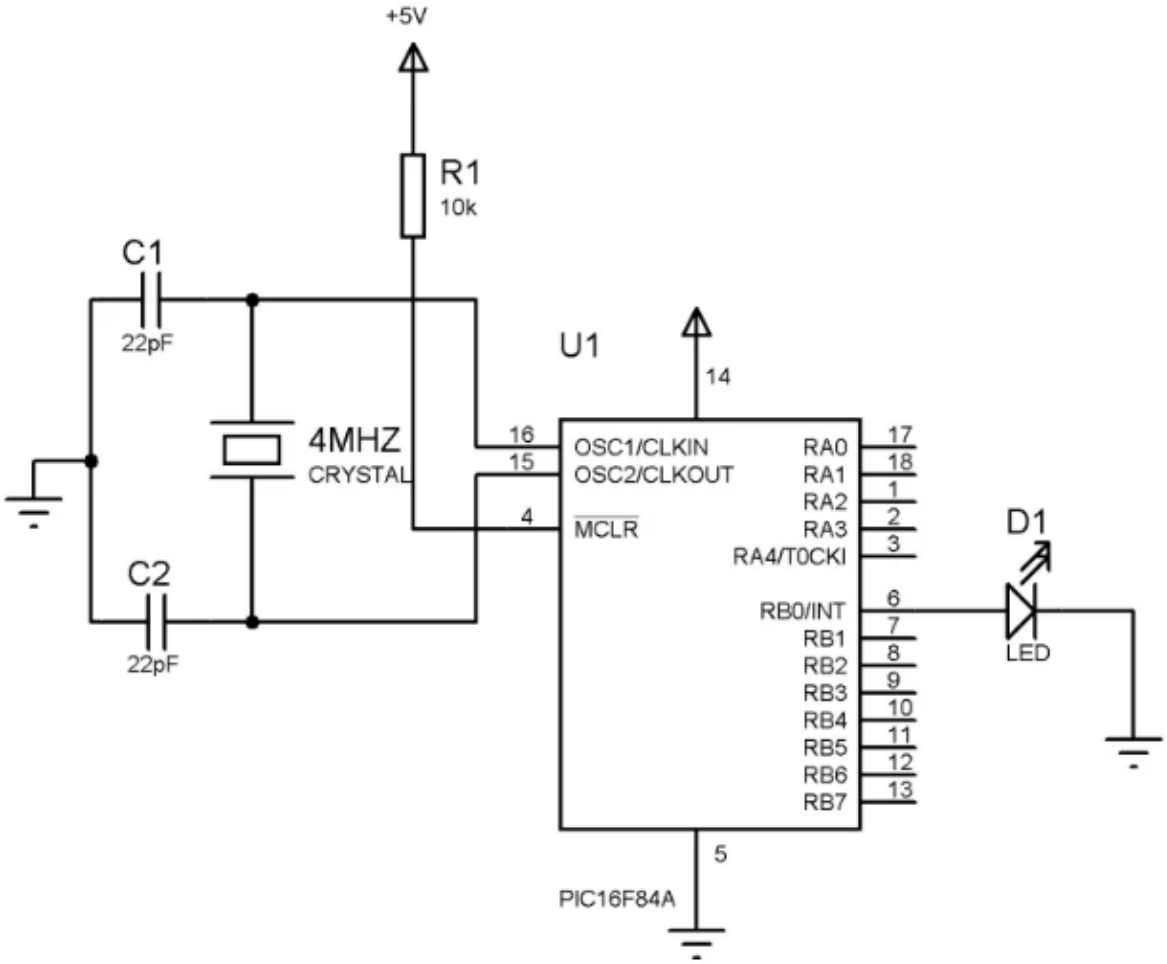
f[b]=0

GOTO étiquette

saut inconditionnel (pile non-utilisée)

Programme simple: LED blink pin #6 (RB0/INT)

Schéma electrique



Programme simple: LED blink pin #6 (RB0/INT)

Programme Assembleur

#INCLUDE <P16F84A.INC>

```
8
9  CBLOCK 0x0C
10     COUNT1
11     COUNT2
12 ENDC
13
14
15 MAIN_PROG CODE                      ; let linker place main program
16
17  START
18     BSF STATUS, RP0
19     MOVLW 0xFE
20     MOVWF TRISB
21     BCF STATUS, RP0
22
23  MAIN
24     BSF PORTB,0
25     CALL DELAY
26     BCF PORTB,0
27     CALL DELAY
28     GOTO MAIN
29
30  DELAY
31     LOOP1 DECFSZ COUNT1, 1
32     GOTO LOOP1
33     DECFSZ COUNT2,1
34     GOTO LOOP1
35     RETURN
36
37     END
```

Programme simple: LED blink pin #6 (RB0/INT)

Déclaration des variables : COUNT1 et COUNT2 à l'aide de CBLOCK.

```
1 CBLOCK 0x0C
2  COUNT1
3  COUNT2
4 ENDC
```

- Toute zone définie par l'utilisateur commence avec la directive CBLOCK, suivie par l'adresse du début de la zone 0x0C. Ensuite, nous pouvons utiliser 68 emplacements mémoire.

Programme simple: LED blink pin #6 (RB0/INT)

- Changement de « bank »

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

```
1 BSF STATUS, RP0
```

- Registre TRISB

```
1 MOVLW 0xFE
2 MOVWF TRISB
```

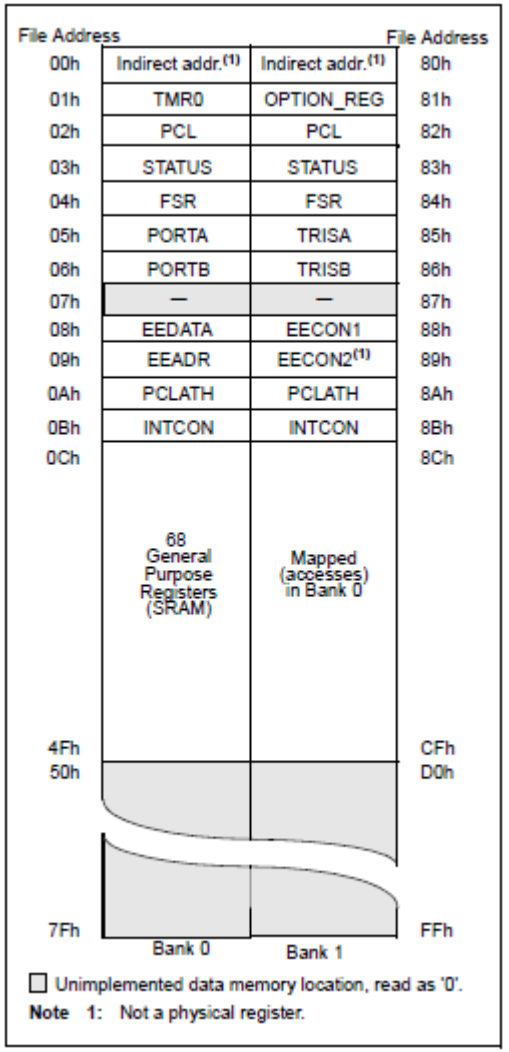
- La valeur de TRISB est 0xFE

TRISB

1	1	1	1	1	1	1	0
bit 7							bit 0

- Rechangement de « bank »

```
1 BCF STATUS, RP0
```



Programme simple: LED blink pin #6 (RB0/INT)

- Mettre la pin RB0 high (1) i.e. BSF
- Appeler la subroutine « DELAY »
- Mettre la pin RB0 low (0) i.e. BCF
- Appeler la subroutine « DELAY »

```
1 MAIN
2     BSF PORTB, 0
3     CALL delay
4     BCF PORTB, 0
5     CALL delay
6     GOTO main
```

Programme simple: LED blink pin #6 (RB0/INT)

- Chaque line est exécuté à une 1 microsecondes (crystal 4 MHz).
- Le temps entre RB0 high et low (1 et 0) est de 1 µs (blink non visible).
- Sous-routine DELAY :
- DECFZ : On diminue le contenu du registre f d'une unité, le résultat est placé dans f si d=1, dans W si d=0 (dans ce cas f reste inchangé). Si le résultat est nul, l'instruction suivante est ignorée.
- COUNT1 (comme COUNT2) a une valeur initiale de 0xFF ou 255
- Ainsi, la sous-routine delay va prendre 255*255 cycles d'instruction (65 ms)

```
1 DELAY
2   LOOP1 DECFSZ COUNT1, 1
3   GOTO LOOP1
4   DECFSZ COUNT2,1
5   GOTO LOOP1
6   RETURN
```