

Les Microcontrôleurs

PIC 16Cxx

Les microcontrôleurs PIC 16Cxx

Sommaire



*Mise en situation: - Evolution technologique
- Analyse fonctionnelle*



Microcontrôleur ou Microprocesseur?



Architecture interne: Von Neumann ou Harvard?



Les registres internes



La base de temps



Le jeu d'instructions et les modes d'adressage



Quitter

Mise en situation

Evolution technologique

L 'évolution des produits domestiques (ou industriels) rend compte d 'un phénomène directement lié à l '*évolution des technologies*:

Mise en situation

Evolution technologique

- Progrès de la *miniaturisation*.
Les téléphones portables en sont un exemple très actuel.

Mais cette miniaturisation ne peut se faire que avec une *évolution de la technologie* utilisée.

- Progrès de l'*intégration*.
Le nombre de structures intégrées à un seul composant est de plus en plus important.
Le nombre de circuits utilisés est ainsi réduit.



Miniaturisation



Intégration

Mise en situation

Evolution technologique

On comprendra aisément qu'un **système microprogrammé** tel que le téléphone portable ne peut être géré par un système minimum à microprocesseur 6809: ***trop encombrant!***

La solution est alors de remplacer le système minimum par ***un seul circuit:***
Le ***microcontrôleur.***

Système minimum à 6809:

- Microprocesseur
- RAM
- EPROM
- PIA
- Décodeur

5 circuits différents!!!

Solution:

- Microcontrôleur

1 seul circuit!!!

Mise en situation

Analyse fonctionnelle

On retrouve ainsi les microcontrôleurs **PIC** dans de nombreuses applications *industrielles* ou *domestiques*.

Prenons l'exemple d'une *télécommande infra-rouge*:

Saisie touche
FP1

Reconnaissance touche
Génération commande
FP2

Emission IR de
la commande
FP3

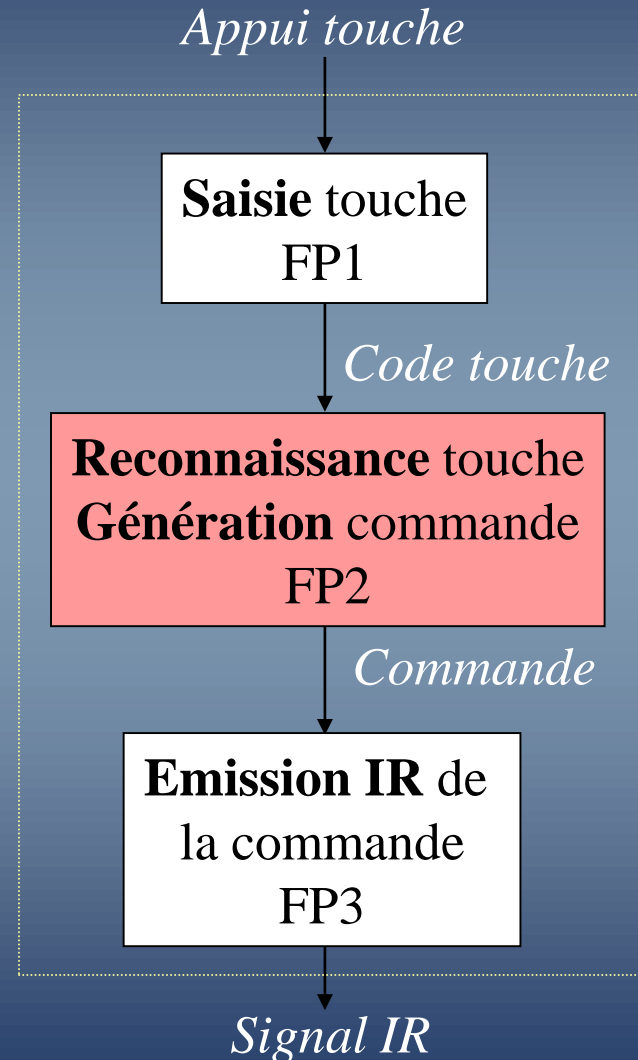
Mise en situation

Analyse fonctionnelle

La *fonction FP1* a pour rôle de prendre en compte l'appui sur une touche et de transmettre le code correspondant à la fonction FP2.

La *fonction FP2* a pour rôle d'identifier la touche à l'aide du « code touche » et de générer le signal commande associé.

La *fonction FP3* se charge de convertir et émettre le signal de commande sous forme de signal infra-rouge.

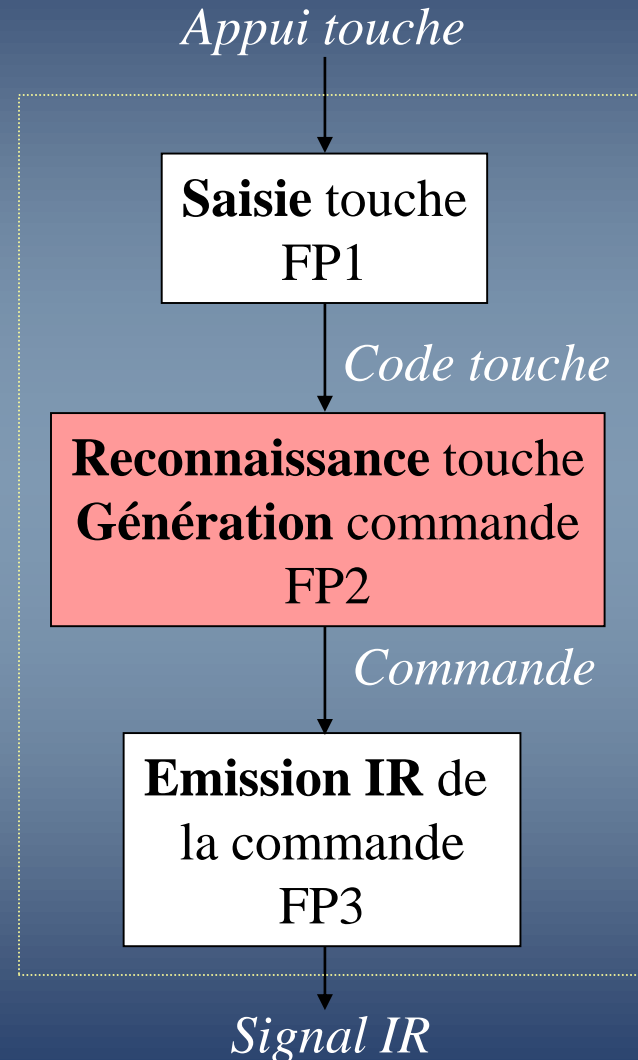


Mise en situation

Analyse fonctionnelle

La **fonction FP2**
« Reconnaissance touche et
génération commande » est
réalisée par une structure
microprogrammée.

C 'est ici un **microcontrôleur PIC** qui se charge, par l 'exécution de son programme, de faire l 'acquisition du signal « code touche », et de générer de signal de commande correspondant.



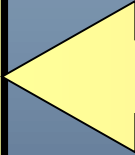
Microcontrôleur ou Microprocesseur?

Suivant le type d 'application envisagé, il est possible de faire appel à différents types de *structures microprogrammées*. Les plus répandues sont les suivantes:

- Le *microprocesseur*.
- Le *microcontrôleur*.



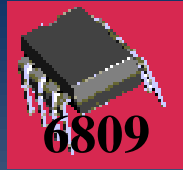
Ex: PC, système minimum à **6809**...



Ex: **PIC**, 68HC11...

Microcontrôleur ou Microprocesseur?

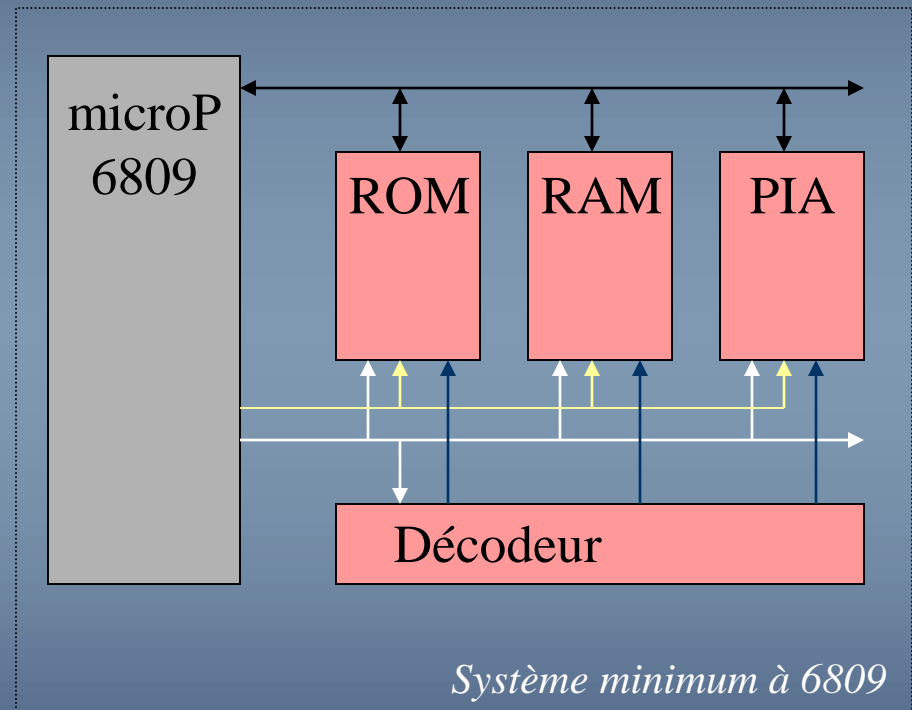
Microprocesseur



Considérons dans un premier temps le système minimum à **microprocesseur 6809**.

Seul, le 6809 ne peut fonctionner. Il requiert différentes ressources qui sont:

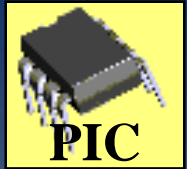
- Une **mémoire programme** (ROM, PROM, EPROM...).
- Une **mémoire données** (RAM).
- Une **interface d'entrées / sorties** (PIA).
- Un **décodeur d'adresse**.
- Différents **bus** d'interconnection.



Système minimum à 6809

Microcontrôleur ou Microprocesseur?

Microcontrôleur

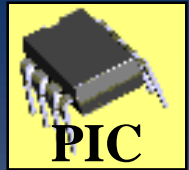


Un système à microprocesseur nécessite une grande *place matérielle* (nombreux circuits) ainsi qu'une bonne qualité de *connectique*.

Les microcontrôleurs permettent quant à eux de s'affranchir de ces contraintes puisque'ils *intègrent* en un seul circuit au moins toutes les ressources propres à un système minimum.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



Ainsi, les microcontrôleurs **PIC 16Cxx** disposent des principales ressources internes suivantes:

- ***Mémoire de programme.***

Les PIC 16Cxx se déclinent selon 3 versions de mémoire de programme:

- ROM** (ou OTPROM),
programmable une seule fois.

Capacité: 512 à 2Kmots.

- UVPR**OM, effaçable par rayonnement UV.

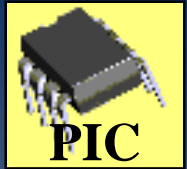
Capacité: 512 à 4Kmots.

- EEPROM**, effaçable électriquement.

Capacité: 1Kmots.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



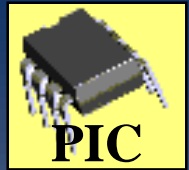
Ainsi, les microcontrôleurs **PIC 16Cxx** disposent des principales ressources internes suivantes:

- *Mémoire de programme.*
- *Mémoire de données.*

Les PIC 16Cxx disposent d'une *mémoire de données* (RAM) de capacité 25 à 192 octets.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



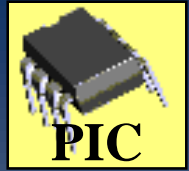
Ainsi, les microcontrôleurs **PIC 16Cxx** disposent des principales ressources internes suivantes:

- *Mémoire de programme.*
- *Mémoire de données.*
- *Entrées/sorties.*

Les PIC 16Cxx proposent un certain nombre de **broches d'entrées/sorties** (12 à 33) permettant l'acquisition ou la transmission de signaux numériques.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



Ainsi, les microcontrôleurs **PIC 16Cxx** disposent des principales ressources internes suivantes:

- *Mémoire de programme.*
- *Mémoire de données.*
- *Entrées/sorties.*

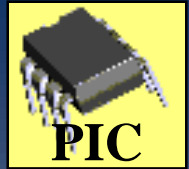
Et éventuellement:

- *Port série.*

Certains PIC 16Cxx possèdent 1 ou 2 *ports série* permettant la transmission série d'informations numériques.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



Ainsi, les microcontrôleurs **PIC 16Cxx** disposent des principales ressources internes suivantes:

- *Mémoire de programme.*
- *Mémoire de données.*
- *Entrées/sorties.*

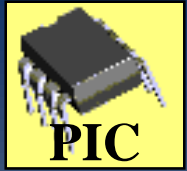
Et éventuellement:

- *Port série.*
- *Convertisseur CAN.*

Certains PIC 16Cxx possèdent en ressource interne un **Convertisseur Analogique Numérique 8bits** permettant l'acquisition de 4 à 8 signaux analogiques différents.

Microcontrôleur ou Microprocesseur?

Microcontrôleur



Conclusion:

Les microcontrôleurs PIC 16Cxx sont des circuits complets et performants.

Ils s'appliquent complètement dans la mise en œuvre de systèmes microprogrammés simples.

Architecture interne

La majorité des structures microprogrammées utilisent une architecture classique appelée:
Architecture Von Neumann.



Architecture Von Neumann:
PC, 6809, 68HC11...

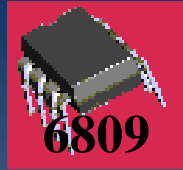
Les microcontrôleurs PIC ainsi que bien d'autres structures sont construites autour d'un autre type d'architecture:
Architecture Harvard.



Architecture Harvard:
PIC, DSP...

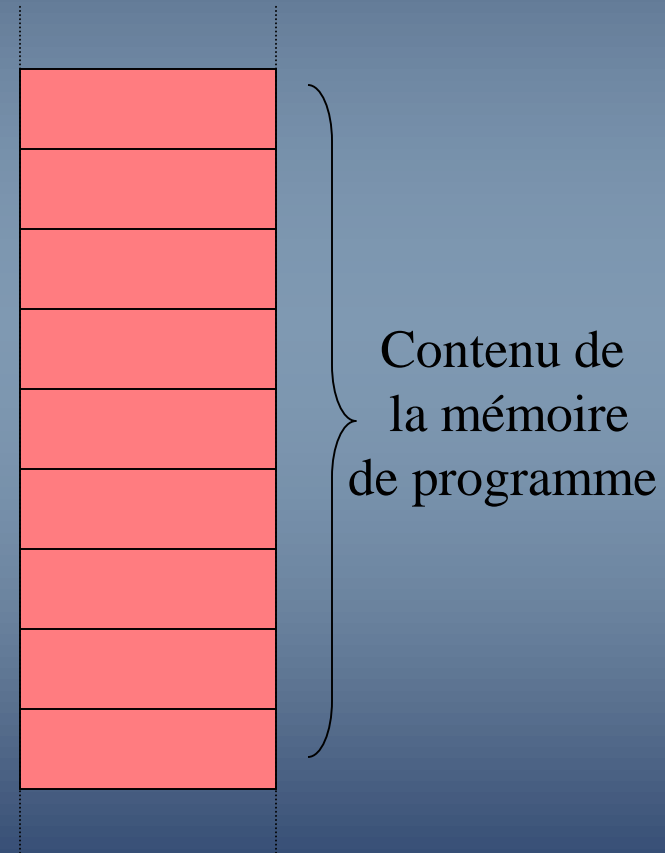
Architecture interne

Architecture Von Neumann



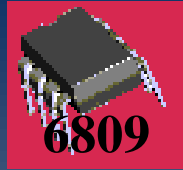
Prenons le cas du système minimum à **6809**. Son architecture est de type **Von Neumann**.

Sa *mémoire de programme* (EPROM) contient comme son nom l'indique le programme à exécuter.



Architecture interne

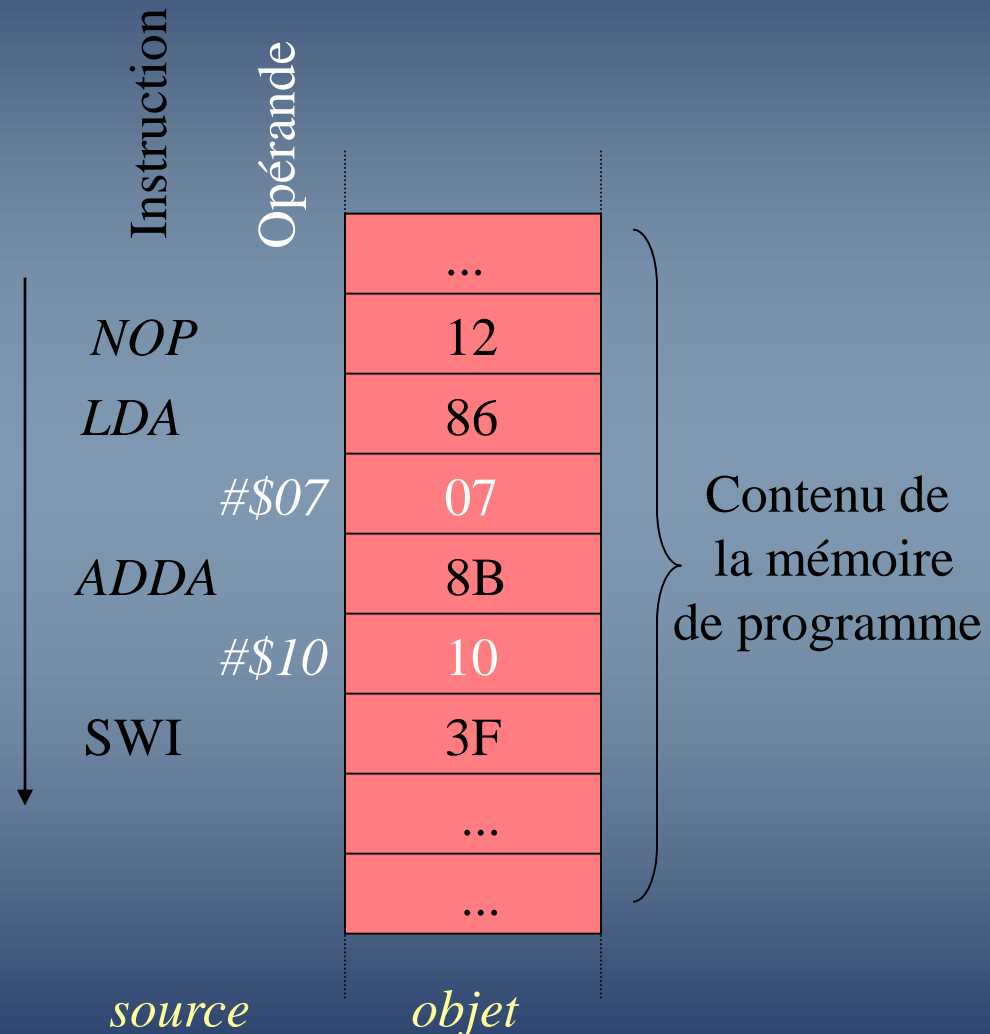
Architecture Von Neumann



Considérons l'exemple du programme source suivant.

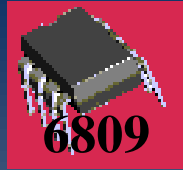
Après assemblage, chaque **instruction** et chaque **opérande** codée sur **un octet** (8 bits) est rangée dans une case de la mémoire.

La mémoire contient donc **successivement** les instructions et les opérandes du programme.



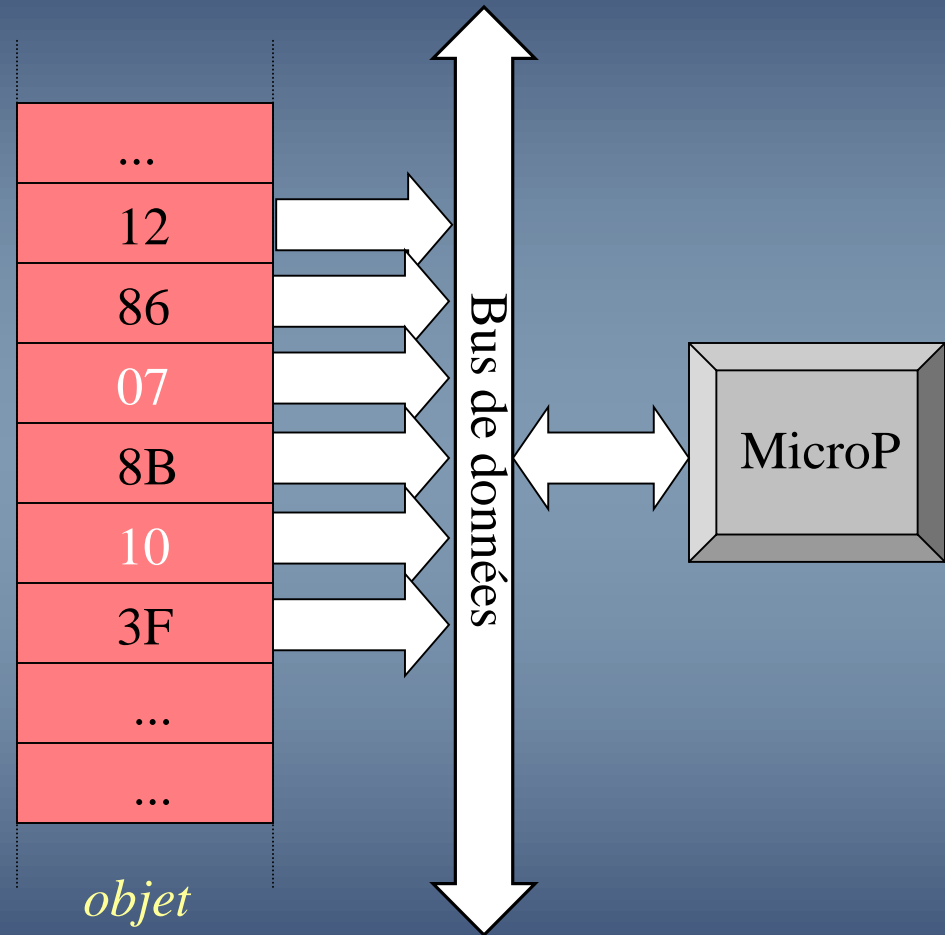
Architecture interne

Architecture Von Neumann



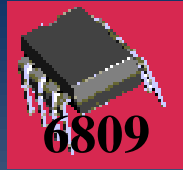
Afin d'exécuter le programme, le microprocesseur doit *lire dans l'ordre* le contenu de chacune des cases mémoires.

Pour cela, chacun des octets de la mémoire est acheminé vers le microprocesseur, via le *bus de données*.



Architecture interne

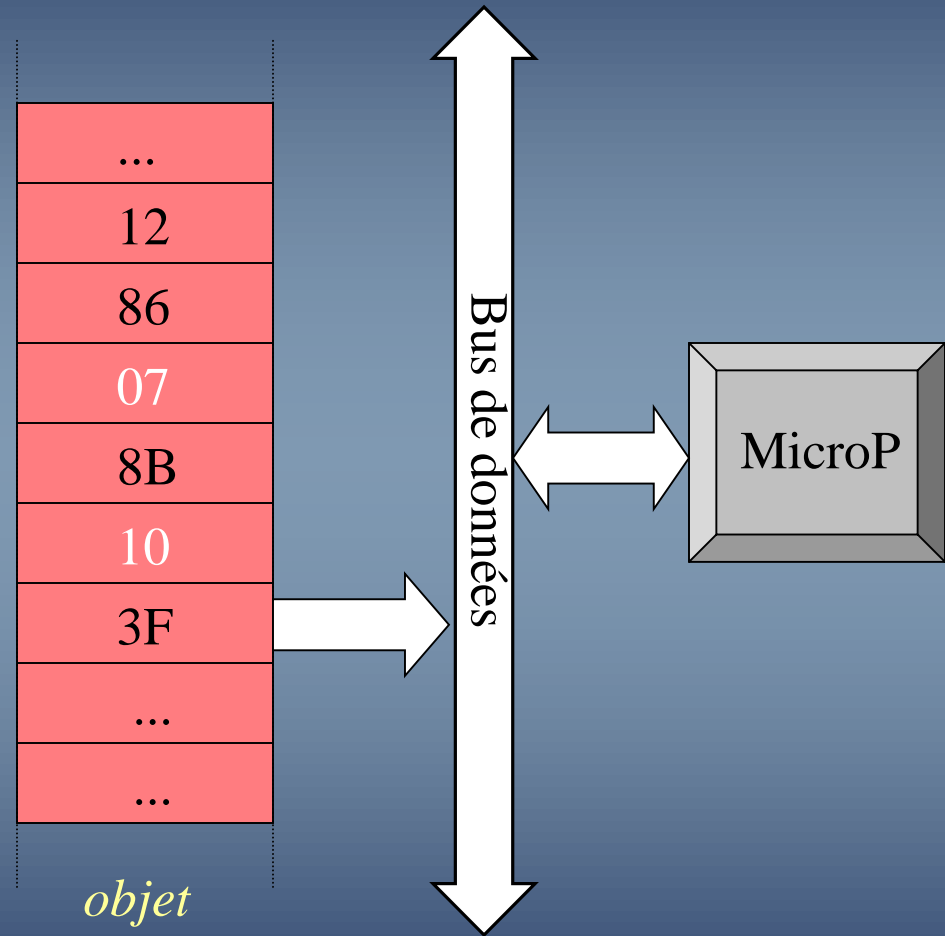
Architecture Von Neumann



Conclusion:

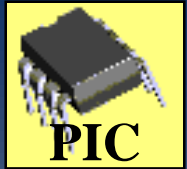
Dans le cas d'une architecture *Von Neumann*, le traitement d'une instruction et son opérande nécessite donc la lecture d'au moins **deux cases mémoires** (3 si l'opérande est codée sur deux octets).

Cela correspond à une durée de **2 ou 3 cycles machine**.



Architecture interne

Architecture Harvard



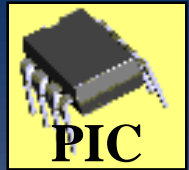
Les microcontrôleurs ***PIC*** ont eux une architecture appelée ***Harvard*** qui présente de nombreux avantages.

Les différences avec les architectures Von Neumann résident essentiellement dans:

- la ***mémoire de programme***
- les ***bus***.

Architecture interne

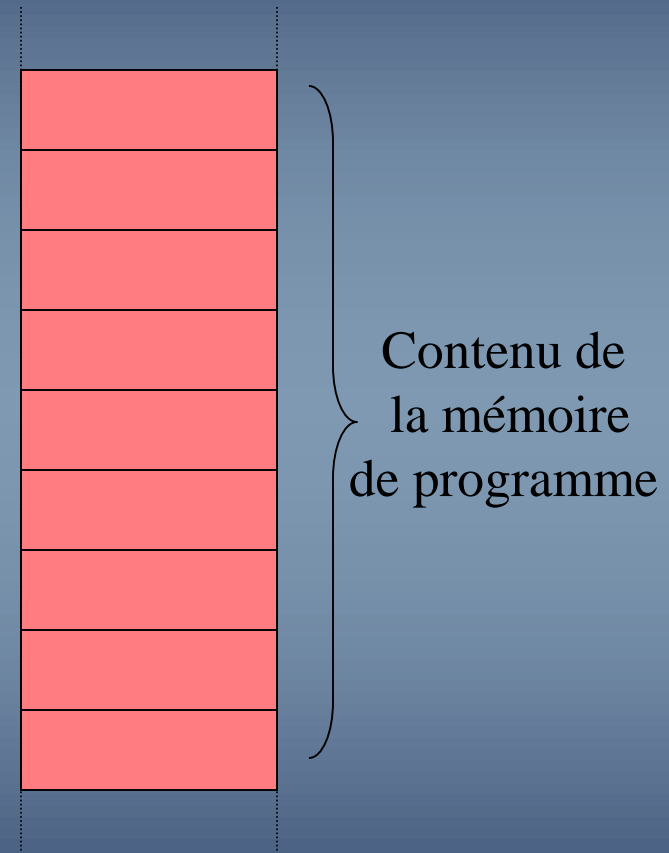
Architecture Harvard



La *mémoire de programme* des *PIC* contient bien entendu le programme à exécuter.

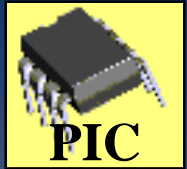
Comme précédemment, ce programme est composé d'*instructions* et d'*opérandes*.

Cependant, une case mémoire peut ici contenir à la fois une instruction *et* son opérande.



Architecture interne

Architecture Harvard

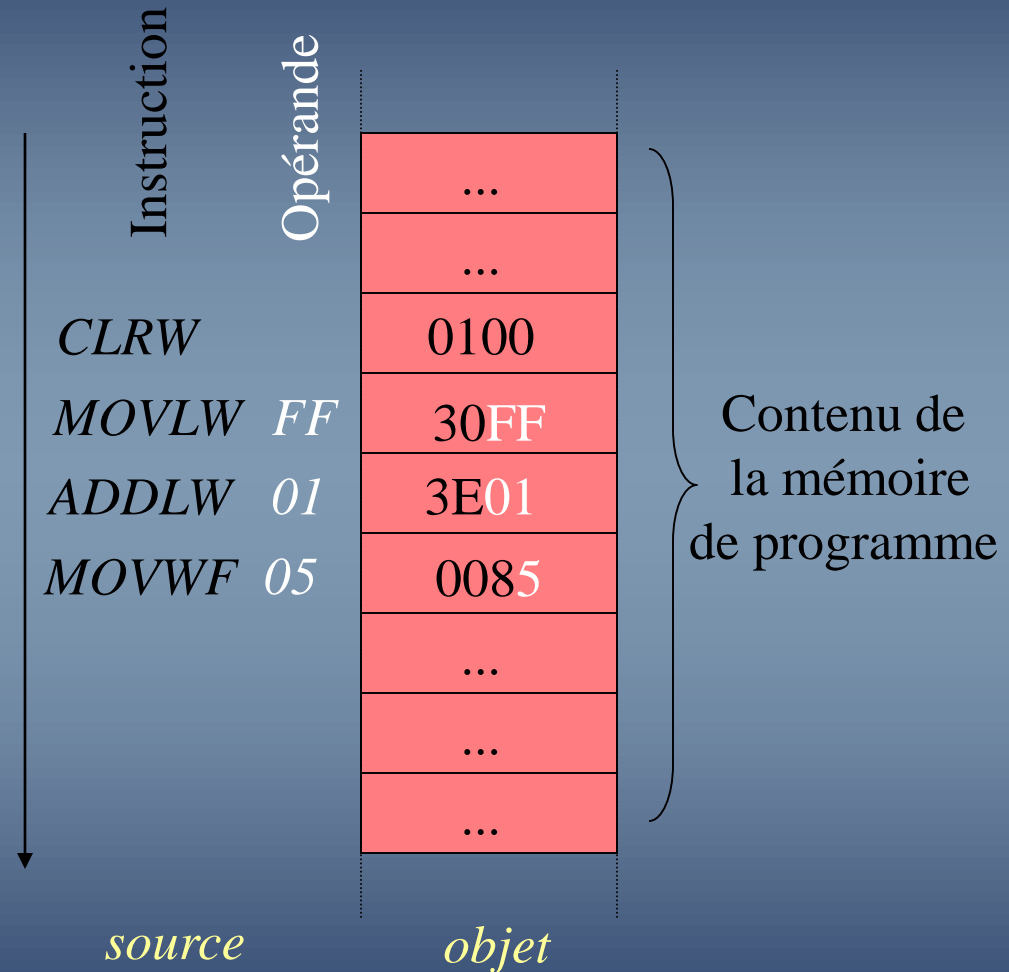


Considérons l'exemple du programme source suivant.

Après assemblage, chaque *instruction* et son *opérande* sont codées sur *un mot binaire* (12 ou 14 bits) puis rangées dans une case mémoire.

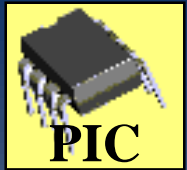
Chaque cas de la mémoire contient donc:

- l'*instruction* à exécuter.
- L'*opérande* associée (non obligatoire).



Architecture interne

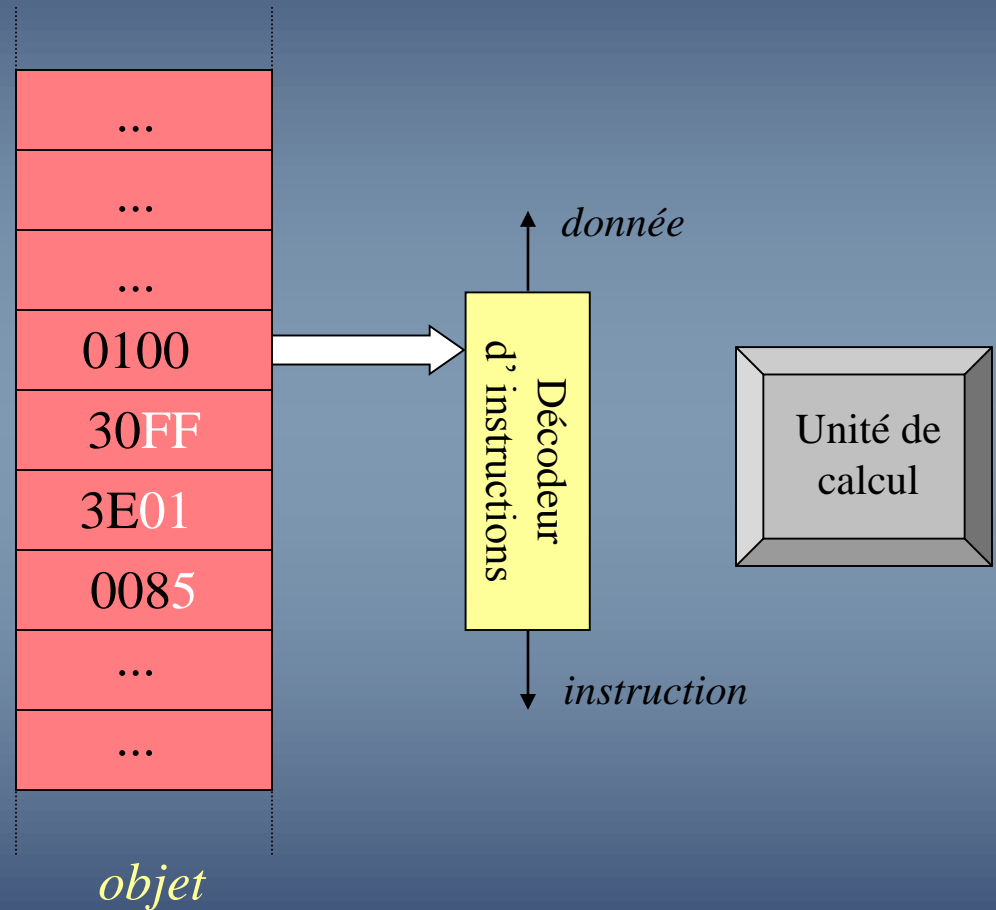
Architecture Harvard



Afin d'exécuter le programme, l'unité de calcul doit ensuite *lire* le contenu de chacune des cases de la mémoire.

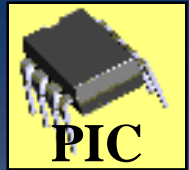
Chaque mot binaire contenu dans la mémoire de programme est alors acheminé vers un *décodeur d'instructions*.

Le rôle de ce décodeur est de *séparer* pour chacun des mots binaires, 1 *instruction* et la *donnée* (opérande).

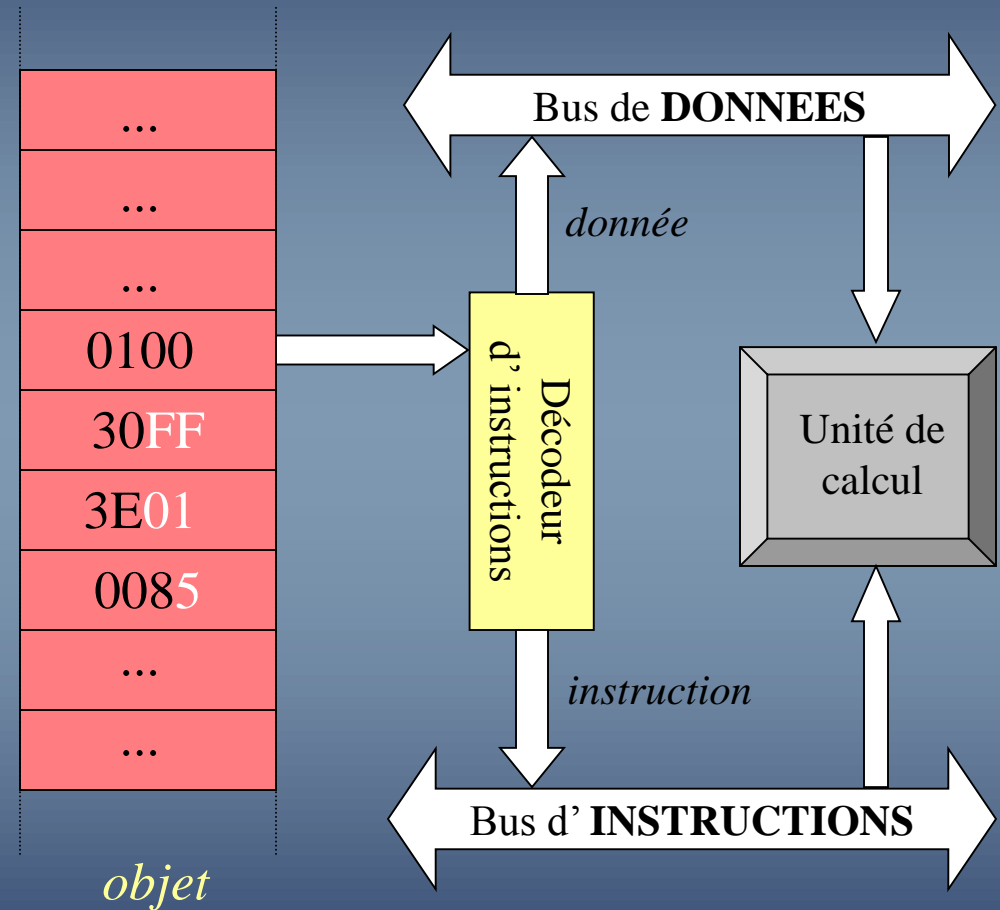


Architecture interne

Architecture Harvard

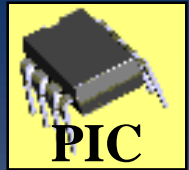


Les *instructions* et les *données* sont ensuite acheminées *simultanément* vers l'unité de calcul par l'intermédiaire de *deux bus différents*.



Architecture interne

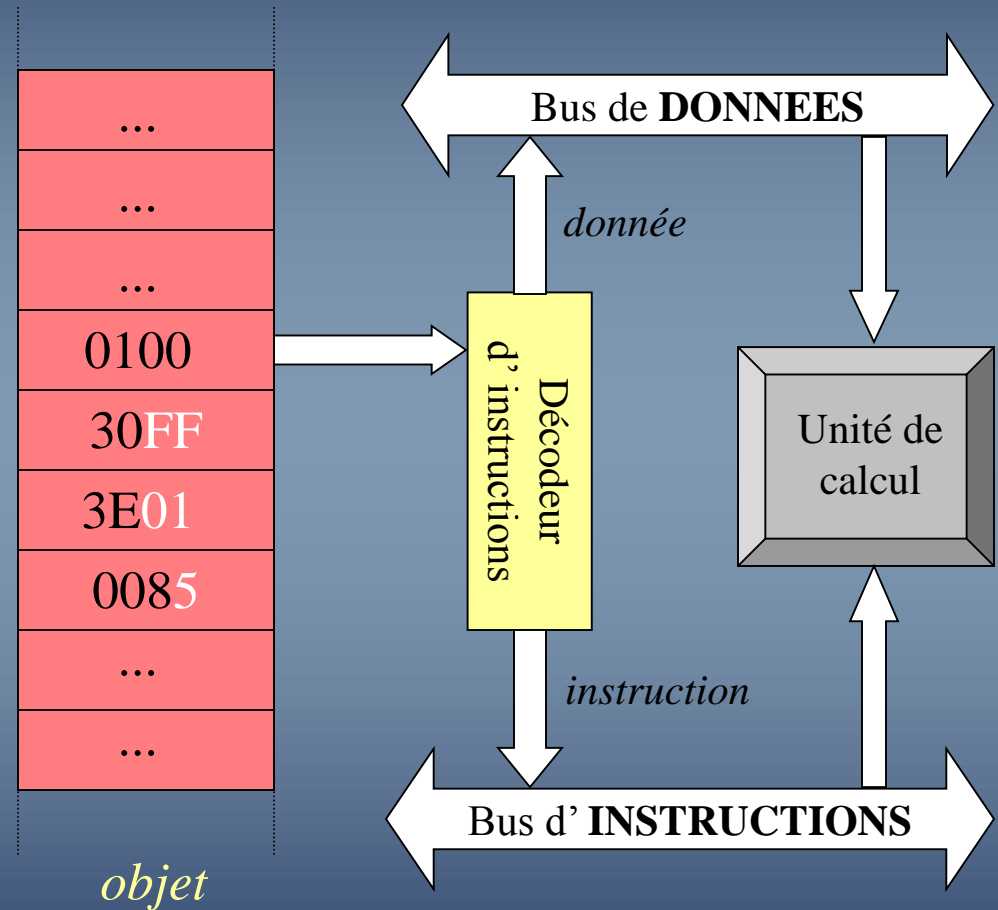
Architecture Harvard



Conclusion:

Dans le cas de l'architecture **Harvard** que possèdent les PIC, la lecture d '*une seule case mémoire* permet le traitement entier d 'une instruction et de son opérande.

Un seul cycle machine est donc nécessaire.



Les registres internes

Selon la version de *PIC 16Cxx* utilisée, le nombre de *registres internes* au circuit est différent.

Ainsi, les registres présentés ci-après sont les plus couramment utilisés:

- Registre de *travail*: **W**
- Registres d '*E/S*: **PORT**
- Registres de *direction*: **TRIS**
- Registre d '*état*: **STATUS**
- Registre *Compteur Programme*: **PC**

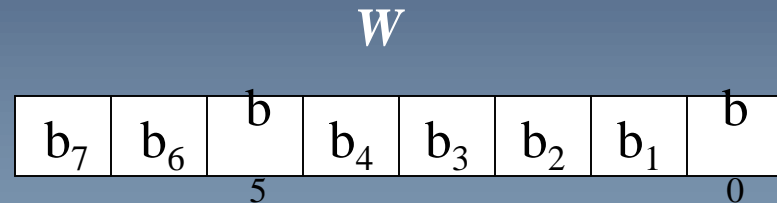
Les registres internes

Registre de travail W

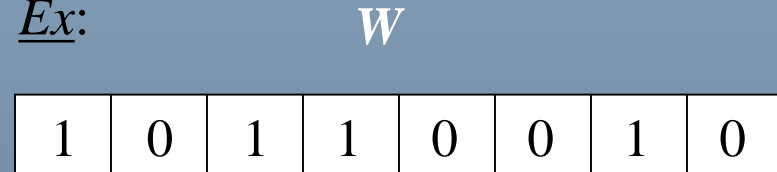
Le registre de travail *W* est un **registre 8 bits** destiné à la manipulation générale des données.

Il peut donc contenir une donnée de 8 bits que l'on appelle ici un ***littéral***.

Le registre *W* peut être comparé aux registres *A* ou *B* du 6809.



Ex:

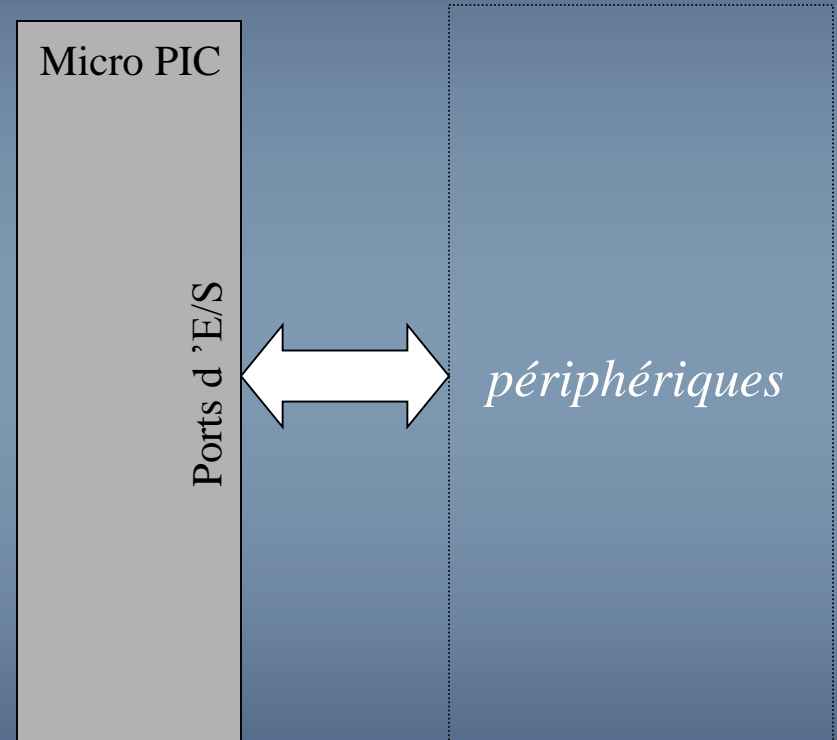


Le *littéral* chargé dans le registre de travail *W* a pour valeur hexadécimale B2.

Les registres internes

Registres d'E/S PORT

Les microcontrôleurs **PIC** peuvent *recevoir ou transmettre* des informations avec des *périphériques extérieurs* par l'intermédiaire de leurs *ports d'E/S*.



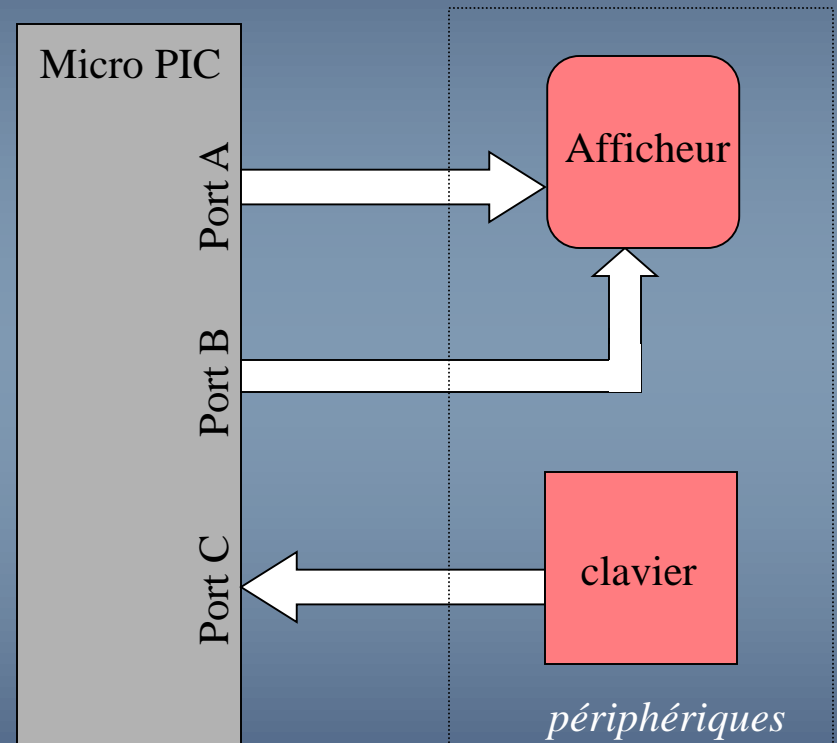
Les registres internes

Registres d 'E/S PORT

Suivant la version utilisée, les circuits proposent **2 ou 3 ports d 'E/S** différents.

Dans l 'exemple suivant, le portC est utilisé pour **recevoir** des informations provenant d 'un clavier.

Les ports A et B sont eux utilisés pour **transmettre** les données à afficher.



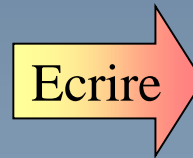
Les registres internes

Registres d 'E/S PORT

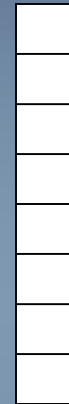
L 'utilisation des registres est ainsi la suivante:

- Pour *transmettre une donnée* sur un port, il faut **ECRIRE** la donnée dans le *registre PORT* correspondant.
- Pour *recevoir une donnée* sur un port, il faut **LIRE** la donnée dans le *registre PORT* correspondant.

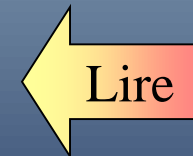
Donnée à transmettre



Port



Donnée reçue



Port



Les registres internes

Registres d 'E/S PORT

Remarque 1:

Les registres PORTA, PORTB et PORTC sont analogues aux registres ***ORA et ORB du 6821.***

Les registres internes

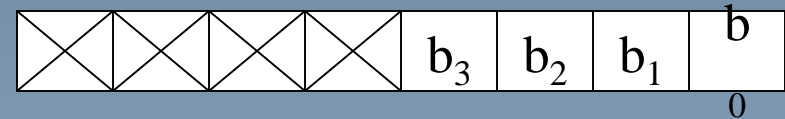
Registres d'E/S PORT

Remarque 2:

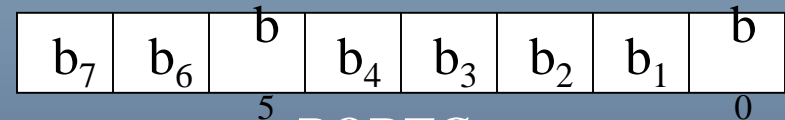
Sur les PIC 16Cxx, le **port A** du circuit ne présente en fait que **4 broches d'E/S**. Le registre correspondant (8 bits) n'a donc d'actifs que les bits **b_0 à b_3** .

Les ports **B et C** possèdent eux bien **8 lignes d'E/S**. Les registres correspondant ont donc les **8 bits actifs**.

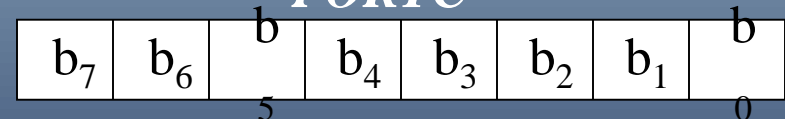
PORTA



PORTB



PORTC



Les registres internes

Registres de direction TRIS

Les registres de direction **TRIS** (**8 bits**) sont directement liés aux registres PORT.

Le rôle des registres **TRIS** est de programmer chacune des **lignes** des ports soit en **entrée**, soit en **sortie**.

Les différentes broches (lignes) d'un même port peuvent donc avoir un rôle différent:

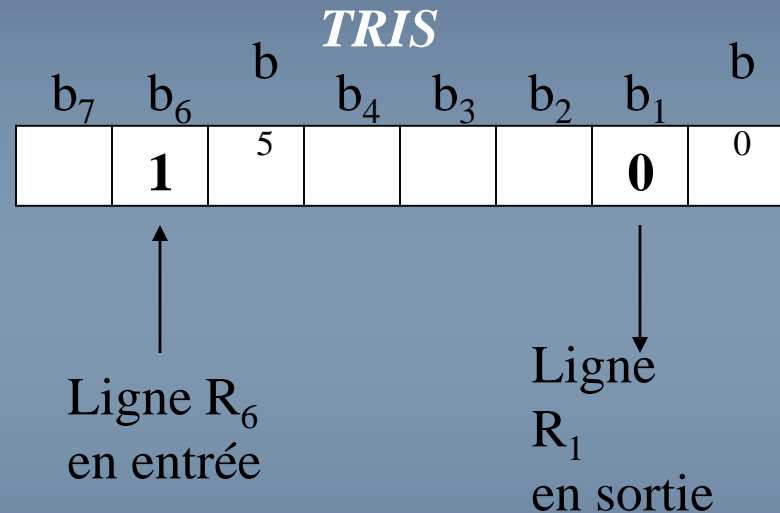
transmettre ou **recevoir** une valeur logique (« 0 » ou « 1 »).

Les registres internes

Registres de direction TRIS

La programmation des registres **TRIS** est la suivante:

- La mise à « **0** » du bit programme la ligne correspondant en *sortie*.
- La mise à « **1** » du bit programme la ligne correspondante en *entrée*.



Les registres internes

Registres de direction *TRIS*

Exemple:

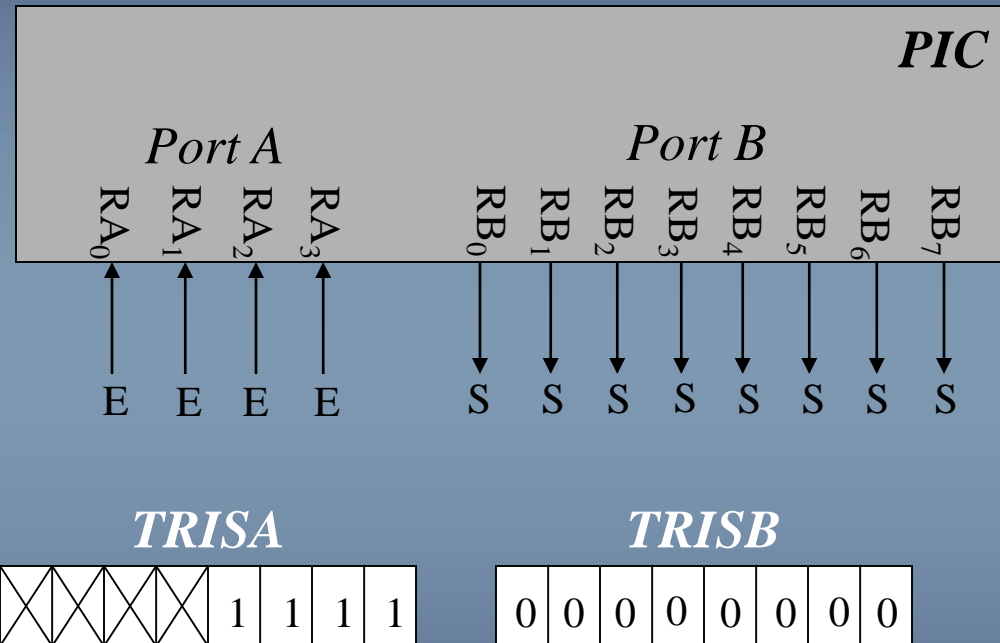
Programmons le *port A en entrée*
et le *port B en sortie*.

Remarque: le port A ne possède
que 4 lignes d'E/S.

En conséquence, les registres
TRIS se programment avec les
valeurs:

TRISA ← F

TRISB ← 00



« 0 » = sortie

« 1 » = entrée

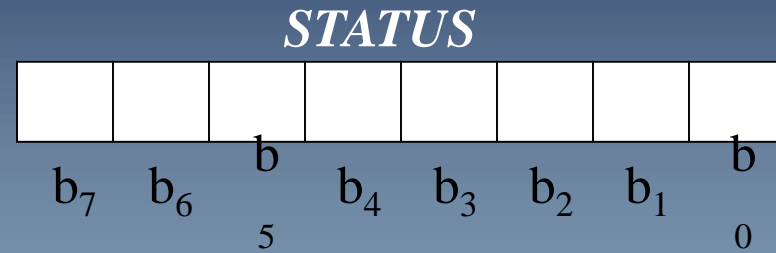
Les registres internes

Registre d'état STATUS

Le registre d'état **STATUS** est un registre **8 bits**.

Le rôle de ce registre est de donner diverses informations à l'utilisateur sur **l'état** de fonctionnement ou sur le résultat d'une opération.

On s'intéressera en fait à seulement 2 bits du registre d'état.



Les registres internes

Registre d'état STATUS

Le bit b_2 : **Z** (Zéro)

Lorsqu'une opération arithmétique ou logique est réalisée, le bit **Z** est mis à **1** si le résultat est nul et à **0** dans le cas contraire.

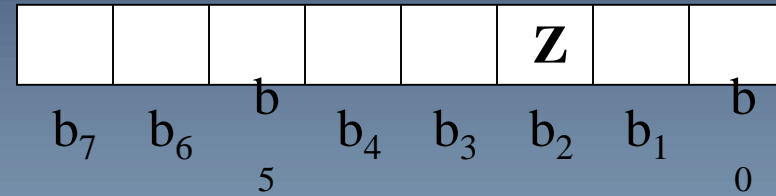
Ex1: L'unité centrale effectue l'opération $7-6=1$.

Z=0

Ex2: L'unité centrale effectue l'opération $7-7=0$.

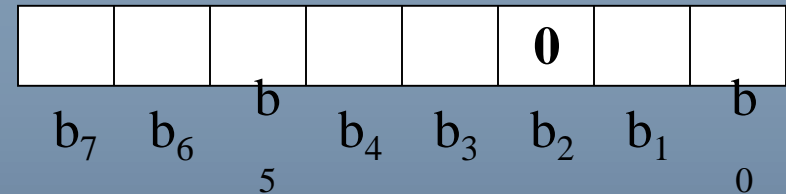
Z=1

STATUS



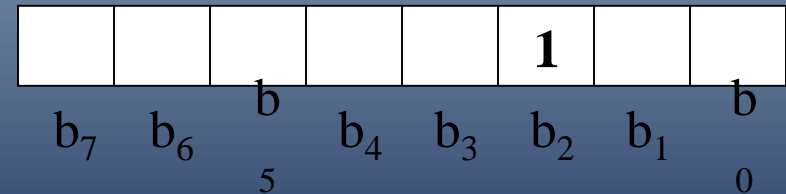
STATUS

Ex1:



STATUS

Ex2:



Les registres internes

Registre d'état STATUS

Le bit b_0 : **C** (Carry = retenue)

Ce bit est positionné à **1** si une addition ou une soustraction génère *une retenue* depuis le bit de poids fort.

Ex1: L'unité centrale effectue 1 opération sur 8 bits:

FE+01=FF

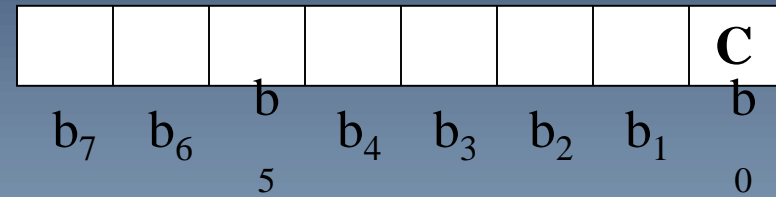
C=0

Ex2: L'unité centrale effectue 1 opération hexadécimale:

FF+1=00 (et une retenue)

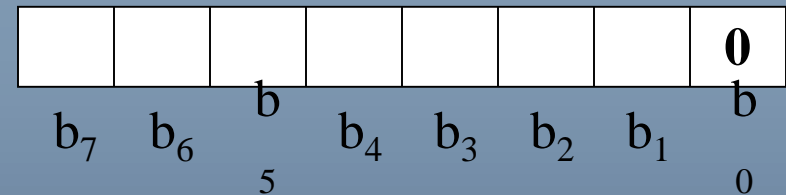
C=1

STATUS



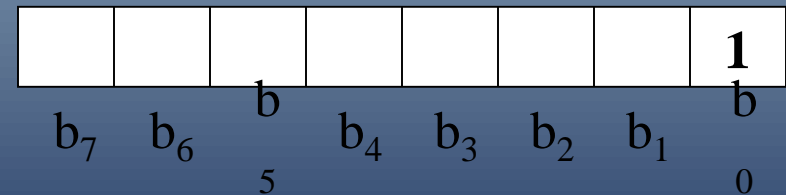
STATUS

Ex1:



STATUS

Ex2:

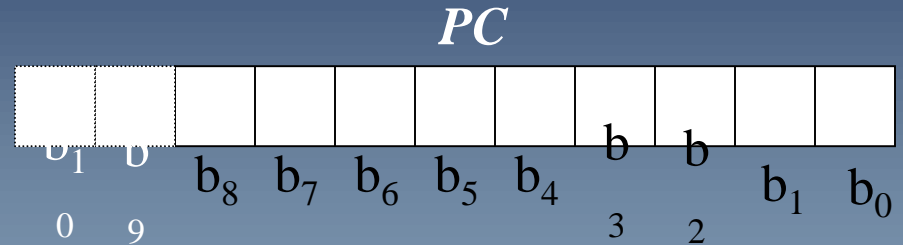


Les registres internes

Compteur programme PC

Le registre **PC** est un registre spécifique **9 ou 11 bits**, suivant le modèle de PIC.

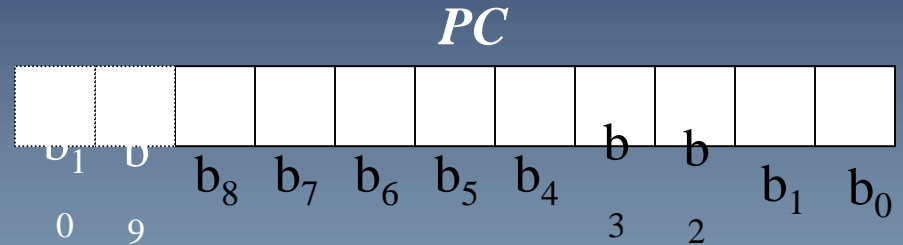
C 'est en fait un **compteur ordinal** qui contient l 'adresse en mémoire de la **prochaine instruction** à exécuter.



Les registres internes

Compteur programme PC

Prenons l'exemple d'un *programme* objet stocké en *mémoire de programme* à partir de l'adresse *000*:



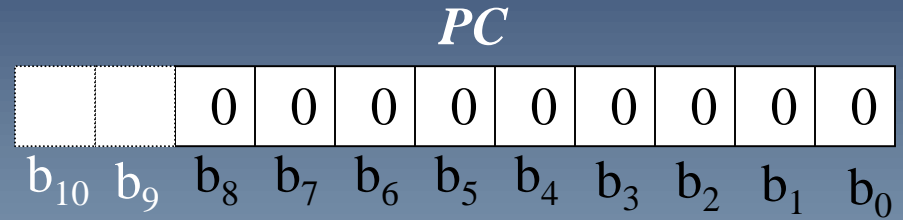
Adresses	Prog.
000	0100
001	30FF
002	3E01
003	0085
004	...
005	...

Les registres internes

Compteur programme PC

1^{er} cycle machine:

Le registre PC est chargé avec l'*'adresse* de la première instruction du programme.



Adresses	Prog.
000	0100
001	30FF
002	3E01
003	0085
004	...
005	...

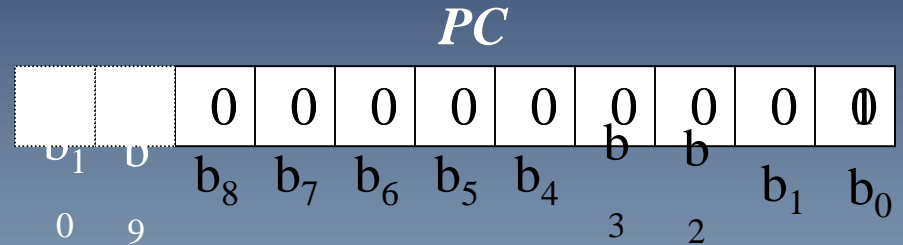
Les registres internes

Compteur programme PC

2^{ème} cycle machine :

De façon simultanée,

- Le registre PC s *'incrémente*.
- La donnée précédemment pointée par le registre PC est *exécutée*.



Adresses

Prog.

000

0100

001

30FF

002

3E01

003

0085

004

...

005

...

Exécution
de l'instruction

0100

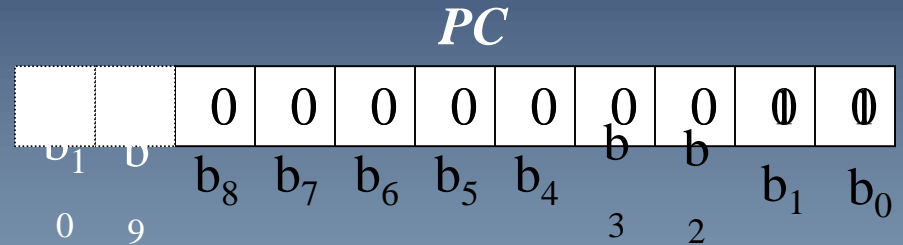
Les registres internes

Compteur programme PC

3^{ème} cycle machine :

De façon simultanée,

- Le registre PC s *'incrémente*.
- La donnée précédemment pointée par le registre PC est *exécutée*.



Adresses

Prog.

000

0100

001

30FF

002

3E01

003

0085

004

...

005

...

Exécution
de l'instruction

30FF

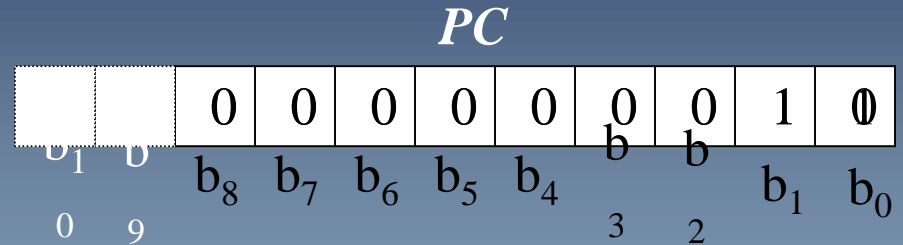
Les registres internes

Compteur programme PC

4^{ème} cycle machine :

De façon simultanée,

- Le registre PC s *'incrémente*.
- La donnée précédemment pointée par le registre PC est *exécutée*.



Adresses

Prog.

000

0100

001

30FF

002

3E01

003

0085

004

...

005

...

Exécution
de l'instruction

3E01

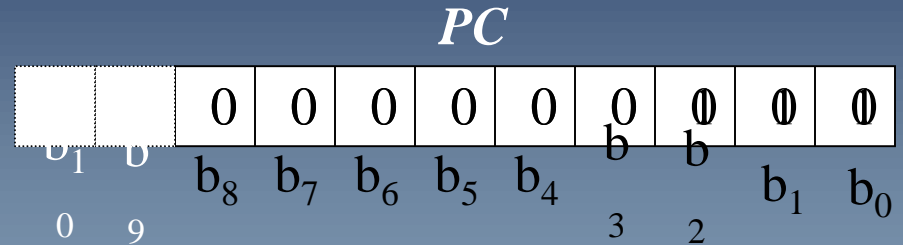
Les registres internes

Compteur programme PC

5^{ème} cycle machine :

De façon simultanée,

- Le registre PC s *'incrémente*.
- La donnée précédemment pointée par le registre PC est *exécutée*.



Adresses

Prog.

000

0100

001

30FF

002

3E01

003

0085

004

...

005

...

Exécution
de l'instruction

0085

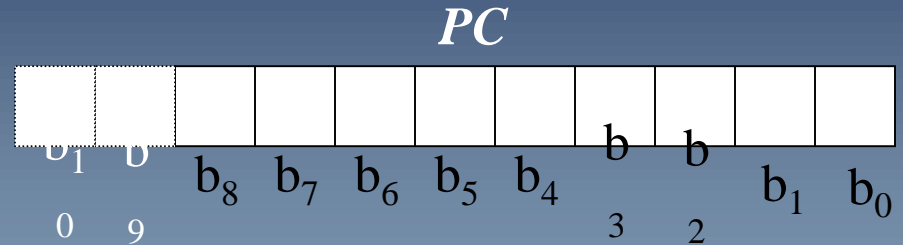
Les registres internes

Compteur programme PC

Conclusion

Le registre PC contient donc à l'instant t l'adresse de la **prochaine instruction** à exécuter.

Ainsi, lorsqu'une instruction est exécutée, la suivante est **déjà pointée** par le registre PC.



Adresses Prog.

000	0100
001	30FF
002	3E01
003	0085
004	...
005	...

Exécution
de l'instruction

La base de temps

Comme tous les circuits microprogrammés, les microcontrôleurs **PIC 16Cxx** fonctionnent à partir d'une base de temps (horloge) appliquée par des composants externes.

Ainsi, les PIC peuvent adopter 4 modèles d'horloge qui sont:

- Version **XT**
- Version **HS**
- Version **RC**
- Version **LP**

Version XT:

oscillateur à quartz jusqu'à 4 MHz.

Version HS (High Speed):

oscillateur à quartz jusqu'à 20 MHz.

Version RC (Résistance-Condensateur):

oscillateur RC jusqu'à 4 MHz.

Version LP (Low Power):

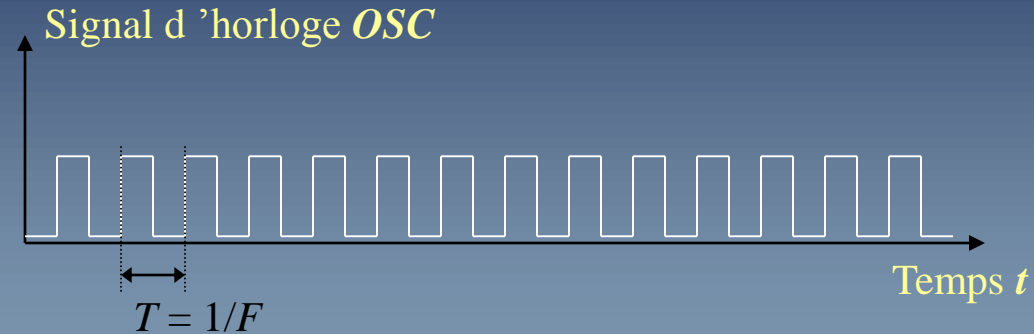
oscillateur à quartz jusqu'à 200 kHz.
Prévu pour des applications à faible consommation.

La base de temps

Le rôle de l'horloge est de ***cadencer*** les différentes opérations effectuées par le microcontrôleur et notamment l'***exécution des instructions*** du programme.

Ainsi, le signal d'horloge possède les caractéristiques suivantes:

- Signal ***carré***.
- De fréquence ***F*** et de période ***T***.

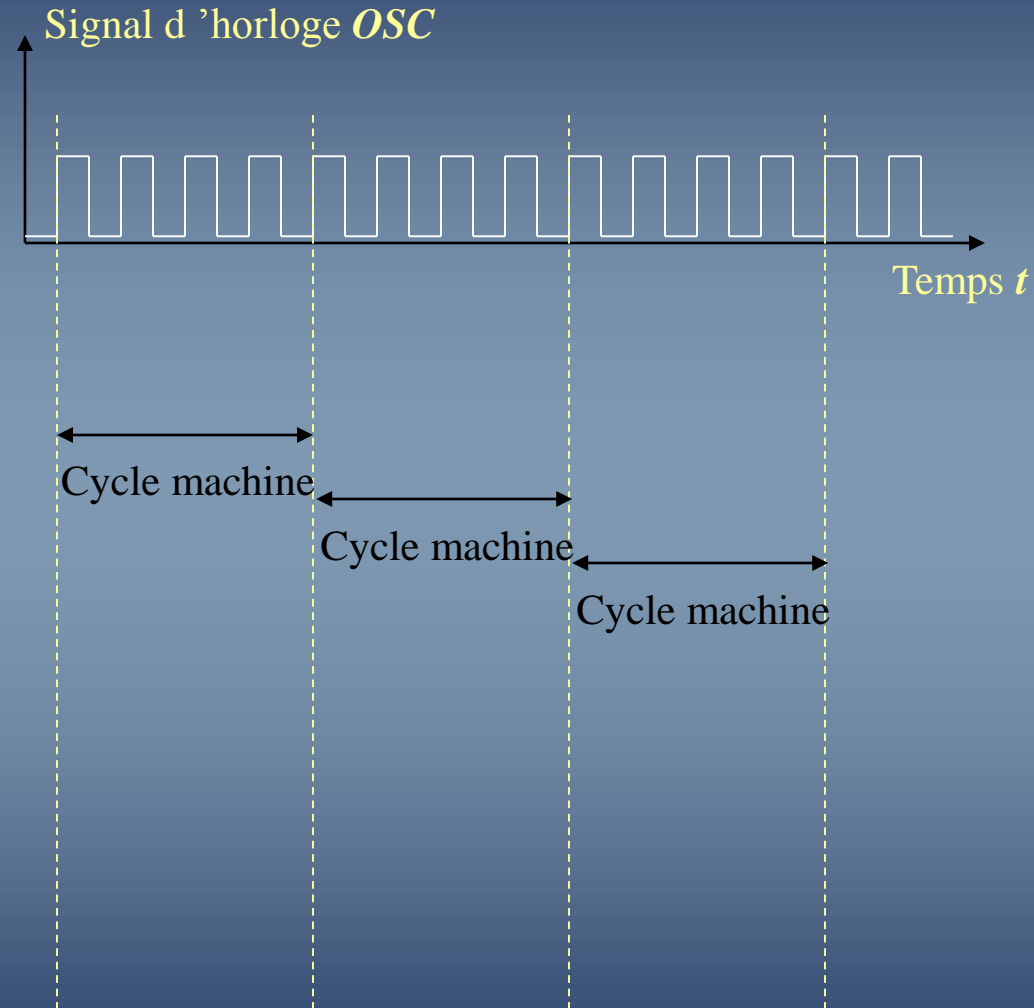


La base de temps

Le signal d'horloge ***OSC*** est en fait délivré par un oscillateur ***externe*** qui peut être un ***quartz*** ou une cellule ***RC***.

Ce signal appliqué au PIC est ensuite, de façon interne, ***divisé par 4***.

On appelle alors ***cycle machine*** la durée caractérisant 4 périodes d'horloge.

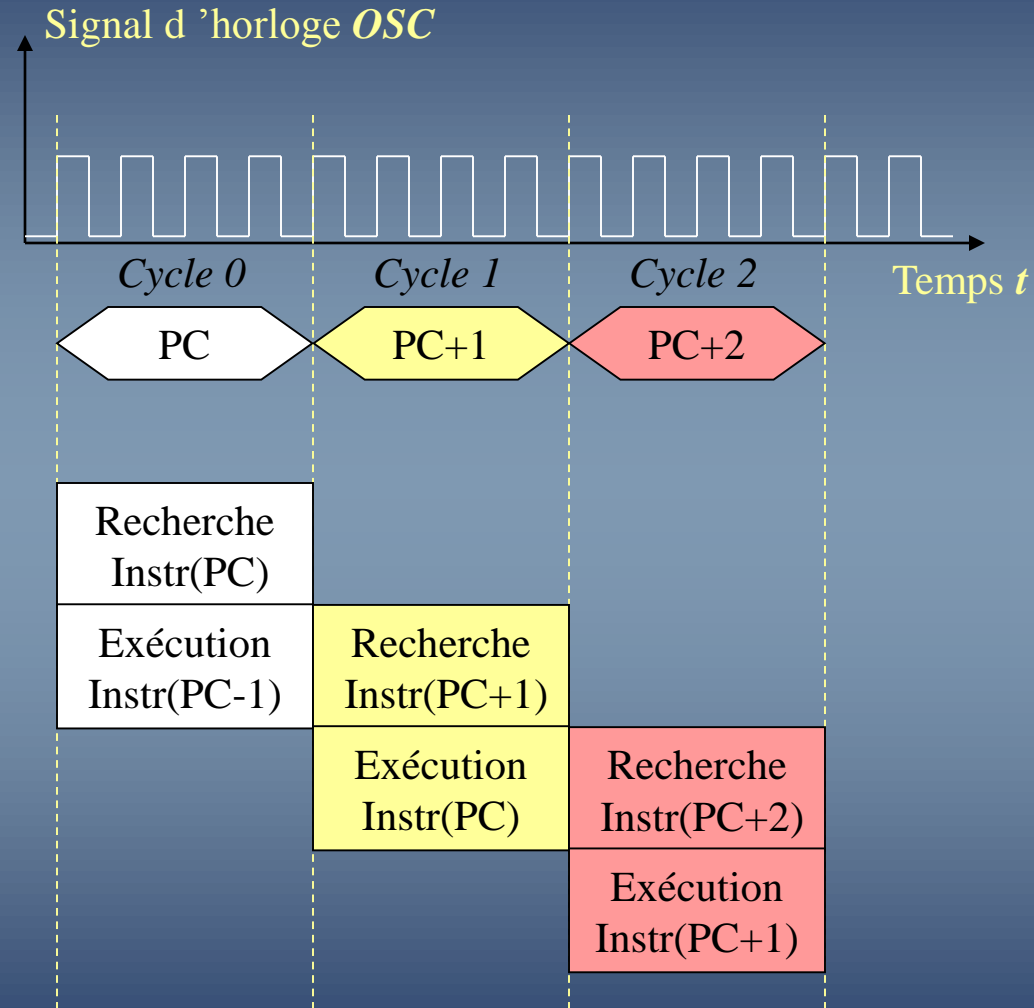


La base de temps

Cette base de temps permet alors de rythmer l'exécution des instructions du programme:

Ainsi, au cours de chaque cycle machine:

- **Incrémentation** du registre **PC**.
- **Recherche** de l'instruction dont l'adresse est contenue dans le registre PC.
- **Exécution** de l'instruction qui était pointée par le registre PC au cours du *cycle précédent*.



La base de temps

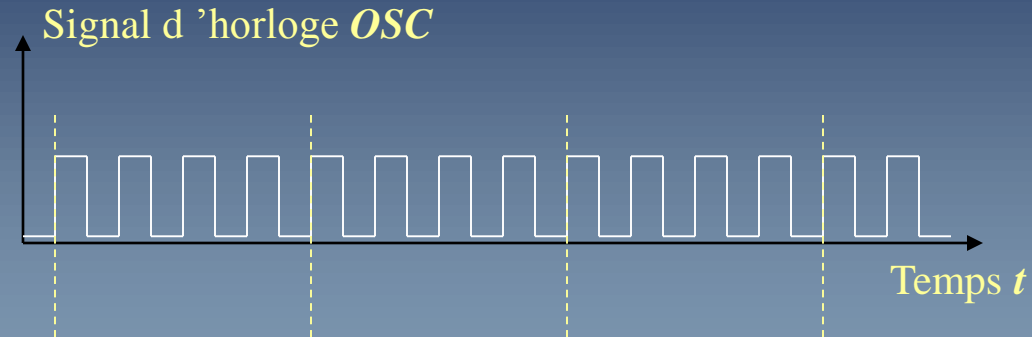
Conclusion:

Il faut donc *un cycle machine* pour exécuter une instruction, soit *4 périodes d'horloge*.

Ex:

Dans le cas d'un oscillateur à quartz à **20MHz**, le temps d'exécution d'une instruction est donc:

200 ns



$$1 \text{ instruction} = 1 \text{ cycle machine} = 4 \times T_{\text{osc}}$$

$$\text{Si } F_{\text{osc}} = 20 \text{ MHz} \quad T_{\text{osc}} = 1/F_{\text{osc}}$$

$$T_{\text{osc}} = 50 \text{ ns}$$

$$\text{D'où } 4 \times T_{\text{osc}} = 200 \text{ ns.}$$

Le jeu d'instructions et les modes d'adressage

Outre la facilité de mise en œuvre matérielle, l'intérêt des microcontrôleurs PIC réside dans le *jeu d'instruction* et les *modes d'adressage* considérablement réduits par rapport à d'autres structures programmables.

Le jeu d'instructions et les modes d'adressage

Jeu d'instructions

En plus de bénéficier d'une architecture dite Harvard, les microcontrôleurs **PIC** sont constitués autour d'une architecture appelée **RISC**.

Ainsi, contrairement à de nombreux circuits mettant en jeu une centaine d'instructions différentes, les **PIC** voient leur nombre d'instructions limités à **33 ou 35**.

Reduced Instruction Set Computer
= *Circuit à jeu d'instructions réduit*

33 ou 35 instructions seulement!

Le jeu d'instructions et les modes d'adressage

Jeu d'instructions

Les différents *mnémoniques* du jeu d'instructions adoptent certaines appellations dont il est nécessaire d'être informé:

- *k* est un littéral, c'est-à-dire une valeur codée sur un octet (8 bits).

Ex: MOVLW *k*

Le littéral *k* (1 octet) est placé dans le registre de travail W.

$k \rightarrow W$

Le jeu d'instructions et les modes d'adressage

Jeu d'instructions

Les différents *mnémoniques* du jeu d'instructions adoptent certaines appellations dont il est nécessaire d'être informé:

- *k* est un littéral, c'est-à-dire une valeur codée sur un octet (8 bits).
- *f* est le symbole correspondant à un registre.

Ex: MOVWF *f*

Le contenu du registre de travail W est transféré dans le registre représenté par *f*.

$W \rightarrow f$

Le jeu d'instructions et les modes d'adressage

Jeu d'instructions

Les différents *mnémoniques* du jeu d'instructions adoptent certaines appellations dont il est nécessaire d'être informé:

- *k* est un littéral, c'est-à-dire une valeur codée sur un octet (8 bits).
- *f* est le symbole correspondant à un registre.
- *b* est le numéro du bit concerné par l'instruction.

Ex: BCF *f, b*

Le bit *b* du registre *f* est mis à zéro (Bit Clear).

0 → bit *b* de *f*

Le jeu d'instructions et les modes d'adressage

Jeu d'instructions

Les différents mnémoniques du jeu d'instructions adoptent certaines appellations dont il est nécessaire d'être informé:

- ***k*** est un littéral, c'est-à-dire une valeur codée sur un octet (8 bits).
- ***f*** est le symbole correspondant à un registre.
- ***b*** est le numéro du bit concerné par l'instruction.
- ***d*** caractérise le registre où doit être placé le résultat de l'opération.

Ex: **ADDWF *f, d***

Le contenu du registre W est ajouté au contenu du registre *f*.

Si *d*=0 le résultat est placé dans W.

Si *d*=1 le résultat est placé dans *f*.

Si *d*=0 $W+f \rightarrow W$

Si *d*=1 $W+f \rightarrow f$

Le jeu d'instructions et les modes d'adressage

Modes d'adressage

La encore, les *modes d'adressage* sont réduits puisqu'on en compte que 4:

- Adressage immédiat.

Ex: `MOVLW k`

La donnée manipulée (*k*) est codée *immédiatement* avec l'instruction.

Le jeu d'instructions et les modes d'adressage

Modes d'adressage

La encore, les *modes d'adressage* sont réduits puisqu'on en compte que 4:

- Adressage immédiat.
- Adressage direct.

Ex: MOVWF *f*

Le registre concerné (*f*) est codé *directement* dans l'instruction.

Le jeu d'instructions et les modes d'adressage

Modes d'adressage

La encore, les *modes d'adressage* sont réduits puisqu'on en compte que 4:

- Adressage immédiat.
- Adressage direct.
- Adressage bit à bit.

Ex: BCF *f, b*

Il permet de manipuler n'importe quel *bit individuel* de n'importe quel registre.

Le jeu d'instructions et les modes d'adressage

Modes d'adressage

La encore, les *modes d'adressage* sont réduits puisqu'on en compte que 4:

- Adressage immédiat.
- Adressage direct.
- Adressage bit à bit.
- Adressage indirect.

Le registre concerné est atteint via un registre d'indirection. Ce mode est en fait très peu utilisé.

Instructions du PIC 16F8XX

35 instructions

- Instructions sur les registres: octet*
- Instructions sur les registres: bit*
- Instructions de contrôle et de saut*

Les instructions sont codées sur 14 bits

Instructions sur les registres: octet

Se sont les instructions qui manipulent les données, elles sont codées comme suit:

- Les 6 bits de poids fort contiennent le code de l'instruction (opcode)*
- Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)*
- Le bit 7 indique où sera placé le résultat de l'opération:*
 - Bit 7=0 -> le résultat sera dans le registre de travail (W)*
 - Bit 7=1 -> le résultat sera dans la même case mémoire RAM (ou le registre)*

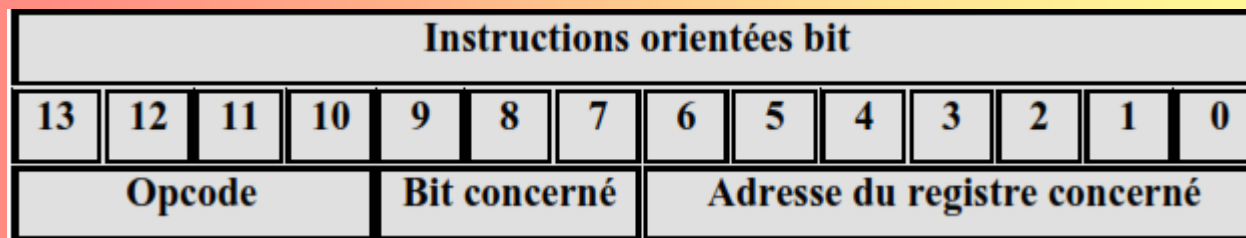
Instructions orientées octet													
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode						0/1	Adresse du registre concerné						

Exp : Movf Reg_temp, W

Instructions sur le bit d'un registre

Se sont les instructions qui manipulent directement les bits d'un registre, elles sont codées comme suit:

- Les 4bits de poids fort contiennent le code de l'instruction (opcode)*
- Les bits 7, 8 et 9 déterminent le bit concerné par l'opération (bit 0 à bit 7)*
- Les 7 bits de poids faible contiennent l'adresse de la case mémoire vive (ce peut être un registre) contenant l'octet à traiter (+RP0 du registre Status donc 8bits)*



Exp : BSF Port_b, 3

Instructions sur les registres (octets)

mnémonique	Description	cycle	bits
ADDWF f,d	d:=-W+f	1	C,DC,Z
ANDWF f,d	d:=-W AND f	1	Z
CLRF f	f:=0	1	Z
CLRW	W:=0	1	Z
COMF f,d	d:=-NOT(f)	1	Z
DECF f,d	d:=-f-1	1	Z
DECFSZ f,d	d:=-f-1 : Skip if Zero	1(2)	.
INCF f,d	d:=-f+1	1	Z
INCFSZ f,d	d:=-f+1 : Skip if Zero	1(2)	.
IORWF f,d	d:=-W OR f	1	Z
MOVWF f,d	d:=-f (permet de savoir si f=0 en faisant	1	Z
MOVWF f	W:=-f	1	.
NOP	n'effectue aucune opération	1	.
RLF f,d	d=f SHR 1 	1	C
RRF f,d	d=f SHL 1 	1	C
SUBWF f,d	d:=-f-W (en complément à 2)	1	C,DC,Z
SWAPF f,d	d:=-f[4..7] <-> f[0..3] (inverse les quartets)	1	.
XORWF f,d	d:=-W XOR f	1	Z

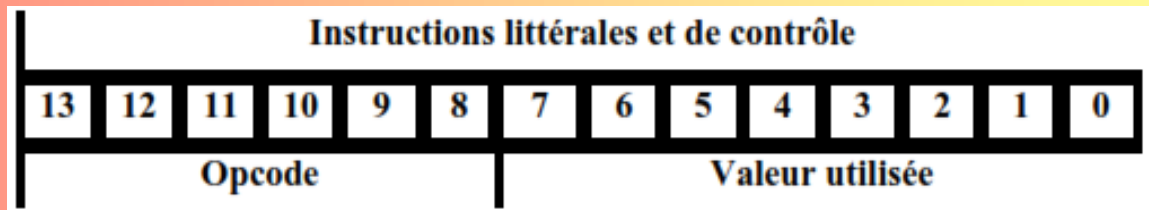
Instructions sur les registres (bit par bit)

mnémonique	Description	cycle	bits affect
BCF f,b	f[b]:=0 (mets à 0 le bit b de f)	1	.
BSF f,b	f[b]:=1 (mets à 1 le bit b de f)	1	.
BTFSC f,b	teste le bit b de f ; Skip if Clear (0)	1(2)	.
BTFSS f,b	teste le bit b de f ; Skip if Set (1)	1(2)	.

Instructions générales de contrôle

Se sont les instructions qui manipulent les données qui sont codées dans l'instruction directement:

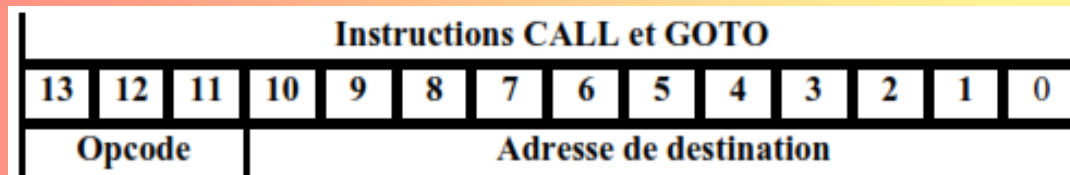
- Les 6 bits de poids fort contiennent le code de l'instruction (opcode)*
- Les 8 bits de poids faible contiennent la valeur numérique ou littérale qui sera utilisée par l'instruction.*



Exp : Movlw 01010101b

Instructions de saut et appel

Se sont les instructions qui provoquent une rupture dans la séquence de déroulement du pg qui est écrit dans la mem programme (1k) 10bits



Exp : Call sous_P

Instructions de contrôle et de saut

mnémonique	Description	cycle	bits affect
ADDLW k	$W := W + k$	1	C,DC,Z
ANDLW k	$W := W \text{ AND } k$	1	Z
CALL k	appel un sous programme	2	.
CLRWDT	remet à 0 le timer du chien de garde	1	TO,PD
GOTO k	se branche à l'adresse k	2	.
IORLW k	$W := W \text{ OR } k$	1	Z
MOVLW k	$W := k$	1	.
RETFIE	fin d'une interruption	2	.
RETLW k	$w := k$, puis effectue un retour de sous programme	2	.
RETURN	effectue un retour de sous programme	2	.
SLEEP	circuit en mode sommeil et stoppe l'oscillateur	1	TO,PD
SUBLW k	$W := k - W$	1	C,DC,Z
XORLW k	$W := W \text{ XOR } k$	1	Z

Les PIC n 'ont désormais plus de secrets pour vous...

FIN

Il est donc temps de les mettre en œuvre dans une application...