Informatique industrielle

Prérequis

- **Les différentes bases de numérotation**
  (*binaire, octal, décimal, hexadécimal*)

- **Conversions et opérations sur les nombres binaires**

- **Notions d'électronique numérique**
  (*fonctions logiques combinatoires et séquentielles*)

Plan

Lien avec les TPs

0 Objectifs et plan du cours, lien avec les TPs

I Catégories de systèmes programmables, architecture des microcontrôleurs
- A Présentation de l'informatique industrielle et des systèmes micro-programmés
- B Architecture neuromorphique : comment le cerveau inspire l'informatique industrielle
- C Architecture des microcontrôleurs

II Codage, instructions assembleur, modes d'adressage des instructions
- A Codage binaire et hexadécimal, complément à deux
- B Premier programme assembleur et algorigramme correspondant (TP1)
- C Modes d'adressage des instructions

III Etude du fonctionnement du microcontrôleur
- A Structure des ports entrées sorties
- B Phase de démarrage du microcontrôleur
- C Exécution d'une instruction
- D Les registres
- E Organisation de la mémoire

IV Les interruptions, exemple avec bouton poussoir
- A Interruptions: définition et gestion (TP2)
- B Interruption par appui sur bouton poussoir

Objectifs du cours

L'objectif de ce cours: vous transmettre une culture des systèmes micro-programmés; vous rendre capable de programmer un microcontrôleur en langage assembleur pour une application visée.

Mots-clés, notions à retenir

- *Notions d'architecture*
- *Éléments constitutifs d'un microcontrôleur*
- *Fonctionnement*

Savoir-faire et réflexes à acquérir

- *Manier les différents types d'instruction assembleur*
- *Mettre à profit les interruptions*
- *Faire le lien entre un algorigramme et un programme*

Evaluation

TP: 50% *Exam: 50%* *(poly autorisé)*

Plan

Lien avec les TPs

0 Objectifs et plan du cours, lien avec les TPs

I Catégories de systèmes programmables, architecture des microcontrôleurs
- A Présentation de l'informatique industrielle et des systèmes micro-programmés
- B Architecture neuromorphique : comment le cerveau inspire l'informatique industrielle
- C Architecture des microcontrôleurs

II Codage, instructions assembleur, modes d'adressage des instructions
- A Codage binaire et hexadécimal, complément à deux
- B Premier programme assembleur et algorigramme correspondant (TP1)
- C Modes d'adressage des instructions

III Etude du fonctionnement du microcontrôleur
- A Structure des ports entrées sorties
- B Phase de démarrage du microcontrôleur
- C Exécution d'une instruction
- D Les registres
- E Organisation de la mémoire

IV Les interruptions, exemple avec bouton poussoir
- A Interruptions: définition et gestion (TP2)
- B Interruption par appui sur bouton poussoir

L'informatique industrielle

« L'informatique industrielle est une branche de l'informatique appliquée qui couvre l'ensemble des techniques de conception et de programmation, de systèmes informatisés à vocation industrielle, qui ne sont pas des ordinateurs. »
(Source : Wikipédia)

Source : Ascom S.A.

L'informatique industrielle

Domaines d'applications :

Alarme, automobile, aviation, instrumentation, médicale, téléphonie mobile, terminaux de paiement pour carte bancaire ...

Image fournie par Microchip

L'informatique industrielle

Applications :

- Automates, robotique,
- Mesures de grandeurs physiques,
- Systèmes temps-réel,
- Systèmes embarqués.

Source : Ascom S.A.

Les différents systèmes programmables

- **Les circuits spécialisés ou ASIC** *(Application Specific Integrated Circuit)* **:**

Les circuits ASIC sont des circuits spécialisés dès leur conception pour une application donnée.

Exemples : DSP (*Digital Signal Processing*), co-processeur arithmétique, processeur 3-D, contrôleur de bus, ...

Source : Texas Instruments

Source : NVidia

Avantages :

- Très rapide
- Consommation moindre
- Optimisé pour une application

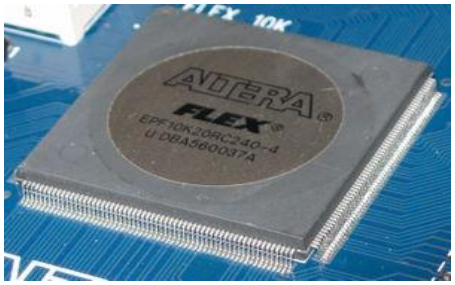
Inconvénients :

- Faible modularité
- Possibilité d'évolution limitée
- Coût

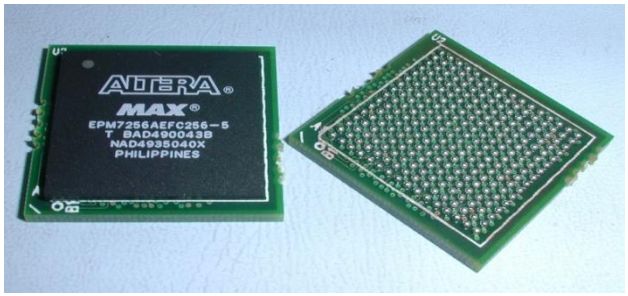
Les différents systèmes programmables

- **Les systèmes en logique programmée et/ou en logique programmable sont connus sous la désignation de PLD** (*programmable logic device, circuit logique programmable*)

- **FPGA** (*field-programmable gate array, réseau de portes programmables in-situ*),
- **PAL** (*programmable array logic, réseau logique programmable*),
- ...



Source : Altera



Source : Altera

« Un circuit logique programmable, ou réseau logique programmable, est un circuit intégré logique qui peut être reprogrammé après sa fabrication. Il est composé de nombreuses cellules logiques élémentaires pouvant être librement assemblées.» (Wikipédia)

Avantages :

- Forte modularité
- Rapidité

Inconvénients :

- Mise en oeuvre plus complexe
- Coûts de développement élevés

Les différents systèmes programmables

- **Les systèmes micro-programmés :**

Les micro-contrôleurs sont typiquement des systèmes micro-programmés.

Microcontrôleur Microchip PIC16F690
en boîtier DIL20

PIC : Programmable Interrupt Controller

Un **microcontrôleur** est un :

« Circuit intégré comprenant essentiellement un microprocesseur, ses mémoires, et des éléments personnalisés selon l'application. » (Arrêté français du 14 septembre 1990 relatif à la terminologie des composants électroniques.)

Un microcontrôleur contient un microprocesseur.

Avantages :

- Mise en oeuvre simple
- Coûts de développement réduits

Inconvénients :

- Plus lent
- Utilisation sous optimale

Nouveau paradigme: L'architecture neuromorphique

Principes de base des systèmes d'information:
Ils expriment l'information sous la forme d'états binaires (0 et 1)

Microprocesseurs classiques ⟶ transistors reposant sur la charge
Charges passantes (1) ou
Bloquées, absence de courant (0)

Autres possibilités, ⟶ lumière
Photons passants (1) ou
Bloqués, absence de lumière (0)

⟶ son
Phonons passants (1) ou
Bloqués, absence de son (0)

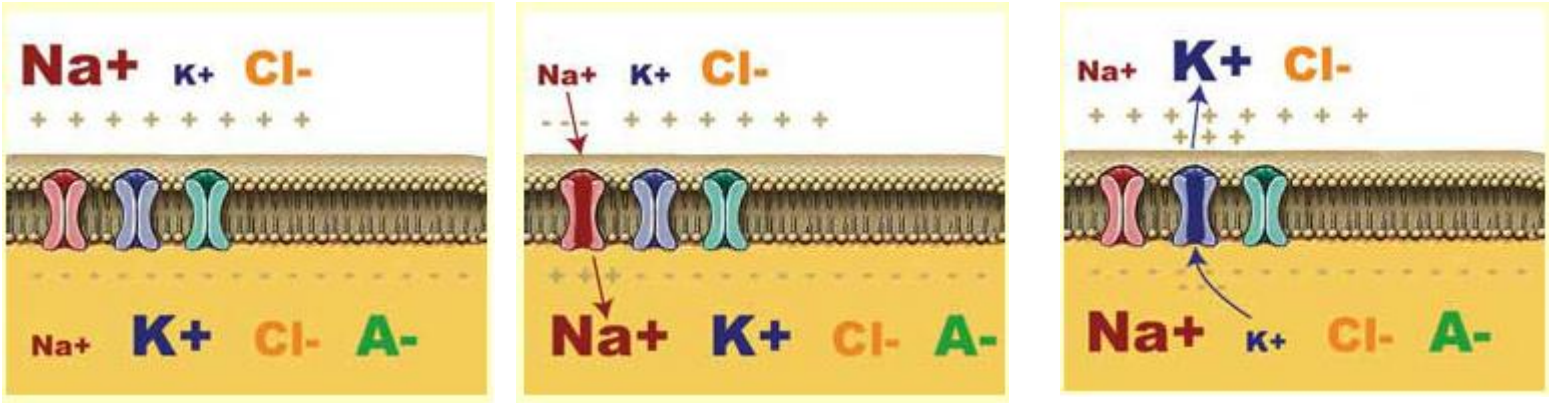
⟶ spin (spintronique)
Polarisation parallèle (1) ou
Polarisation antiparallèle (0)

Nouveau paradigme: L'architecture neuromorphique

Quelques chiffres:

1 mm3 de cortex: 90 000 neurones, 700 000 synapses, 4 kms d'axones;
Consommation 30 W (cerveau du chat 2W)

Un neurone présente une membrane, K+ à l'intérieur, Na+ à l'extérieur



- la différence de potentiel change
- ‘spike’ de 50 mV.
- Propagation de spikes le long d'un axone: (1)
- Absence de propagation de spikes: (0)

Nouveau paradigme: L'architecture neuromorphique

	Cerveau	Microprocesseur
Type de 'système'	Système parallélisé	Système à une unité de calcul, l'unité arithmétique et logique
Propagation de l'information	Echange d'ions chargés +	Courant électrique
Tensions	Potentiel d'action 50 mV (faible)	Tension d'alimentation 1V (élevée)
Puissance consommée	30 W	100 W / cm2 (=centrale nucléaire ou plaque chauffante)
Accès à la mémoire	Par contenu	Par adresse

Nouveau paradigme: L'architecture neuromorphique

	Cerveau	Exemple du plus gros ordinateur Blue Gene
Vitesse (Tera FLOPS) FLOPS: Floating Point Operations per Second	100 000 TFLOPS	478 TFLOPS
Puissance consommée	30 W	2 MW
Mémoire vive		16 To
Capacité de stockage		400 To
Nombre d' « unités centrales »	100 milliards de neurones	147 000 processeurs

Nouveau paradigme: L'architecture neuromorphique

Essentiel: ordre de grandeur du potentiel d'action faible (50 mV << 1 V)

Faible consommation d'énergie, hyperinterconnecté, mais très lent.

Dernières nouvelles, 1er novembre 2013: les neurones peuvent chez une même personne présenter des ADN différents (le cerveau a une capacité évolutive propre)
Perspectives: s'inspirer du cerveau pour concevoir un microprocesseur à très faible consommation.

Deux types de processeurs

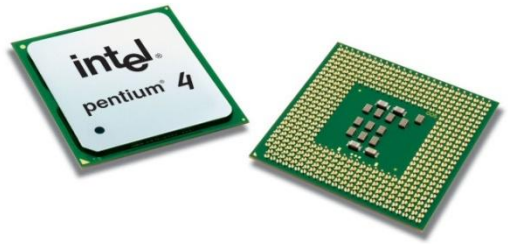
- **CISC** : *Complex Instruction Set Computer*

  Grand nombre d'instructions,
  Type de processeur le plus répandu

- **RISC** : *Reduced Instruction Set Computer*

  Nombre d'instructions réduit
  (sélection des instructions pour une exécution plus rapide)
  Décodage des instructions plus rapide

Évolution des processeurs



Source : Intel

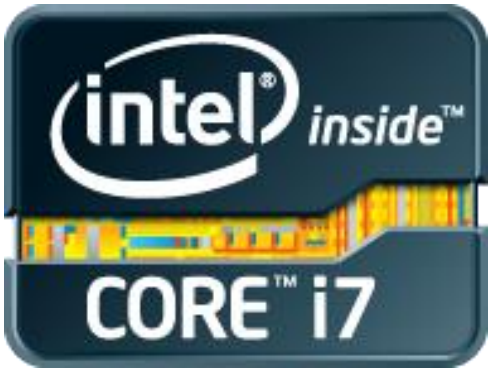
Intel Pentium 4 Northwood C (2002)
42 millions de transistors, gravés en 0,13 µm
architecture interne 32 bits
fréquence d'horloge 2,4/3,4 Ghz
(bus processeur : 200Mhz)
450 MIPS

Source : Intel

Intel 8086 (1978)
39 000 transistors, gravés en 3µm
architecture interne 16 bits
bus 16 bits
fréquence d'horloge 4,77/10 Mhz
0,33/0,75 MIPS

Source : Wikipédia

Évolution des processeurs



Intel Core i7 Ivy bridge (sept. 2013)
1,4 Milliards de transistors, gravés en 22nm
architecture interne 64 bits
4/12 coeurs
fréquence d'horloge 4,0 Ghz
Fréquence de bus: 0,2 GHz
6000 MIPS
→ Jeux 3D

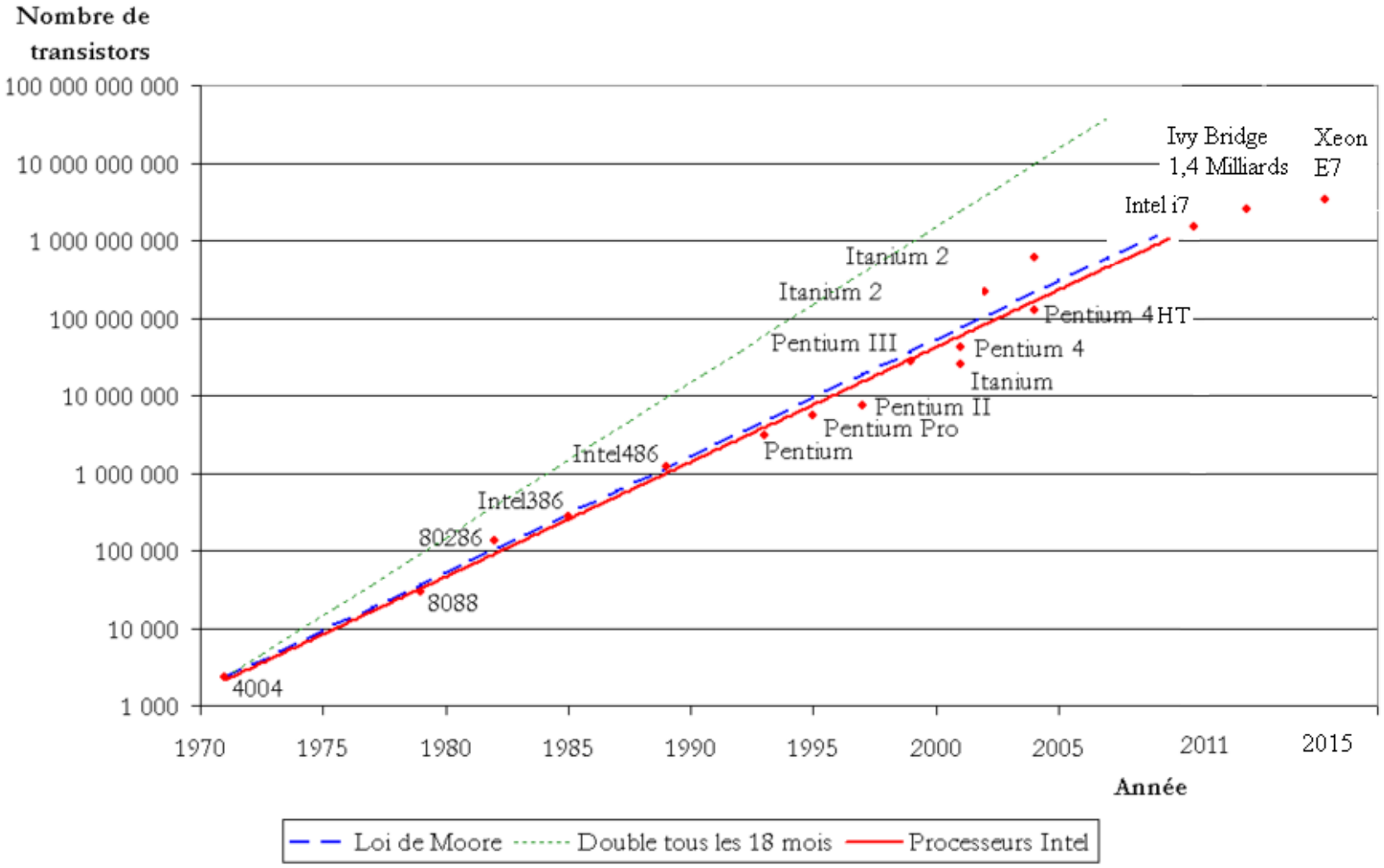
Intel Pentium 4 Northwood C (2002)
42 millions de transistors, gravés en 0,13 µm
architecture interne 32 bits
fréquence d'horloge 2,4/3,4 Ghz
Fréquence de bus: 0,2 GHz
450 MIPS

« The wall » :
limite industrielle et physique,
→ 20 nm (surmontée par la technologie 3D ?)

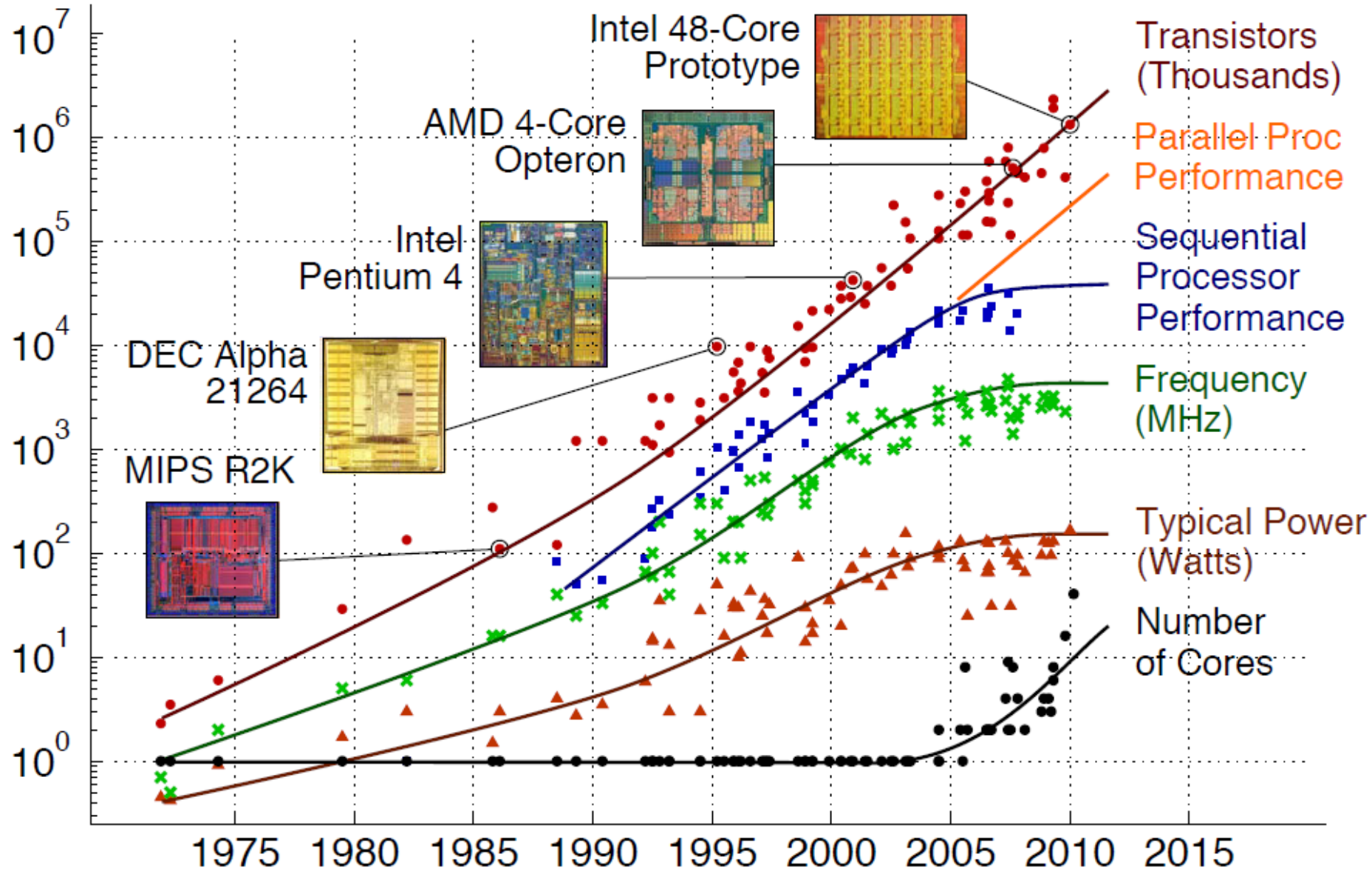
performance / Watt consommé

Source : Intel

Loi de Moore


Nombre de transistors
100 000 000 000
10 000 000 000
1 000 000 000
100 000 000
10 000 000
1 000 000
100 000
10 000
1 000
1970
1975
1980
1985
1990
1995
2000
2005
2011
2015
Année
Ivy Bridge
1,4 Milliards
Xeon
E7
Intel i7
Itanium 2
Itanium 2
Pentium 4 HT
Pentium III
Pentium 4
Itanium
Pentium II
Pentium Pro
Pentium
Intel486
Intel386
80286
8088
4004
Loi de Moore
Double tous les 18 mois
Processeurs Intel

Performances des processeurs sur 40 ans



Source: Batten, données collectées par Horowitz, Labonte, Shacham

- **Les bus d'un système micro-programmé**

« Un bus est un jeu de lignes partagées pour l'échange de mots numériques. » (Traité de l'électronique, Paul Horowitz & Winfield Hill)

Définition : Un bus permet de faire transiter (liaison série/parallèle) des informations codées en binaire entre deux points. Typiquement les informations sont regroupés en mots : octet (8 bits), word (16 bits) ou double word (32 bits).

Caractéristiques d'un bus:

- nombres de lignes,
- fréquence de transfert.

Il existe 3 types de bus :

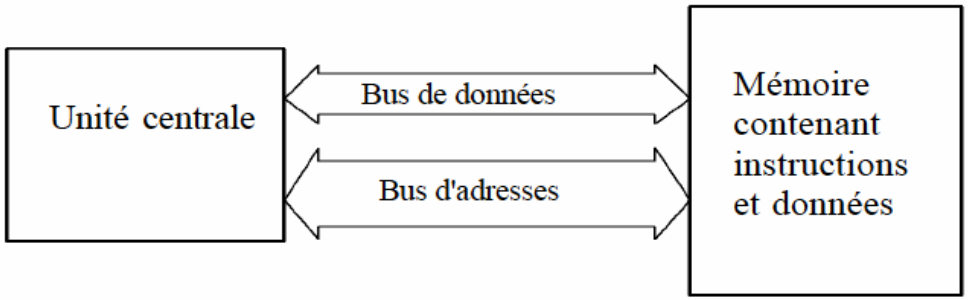
- **Bus de données** : permet de transférer entre composants des données,
  ex. : *résultat d'une opération, valeur d'une variable, etc.*

- **Bus d'adresses** : permet de transférer entre composants des adresses,
  ex. : *adresse d'une case mémoire, etc.*

- **Bus de contrôle** : permet l'échange entre les composants d'informations de contrôle [bus rarement représenté sur les schémas].
  ex. : *périphérique prêt/occupé, erreur/exécution réussie, etc.*

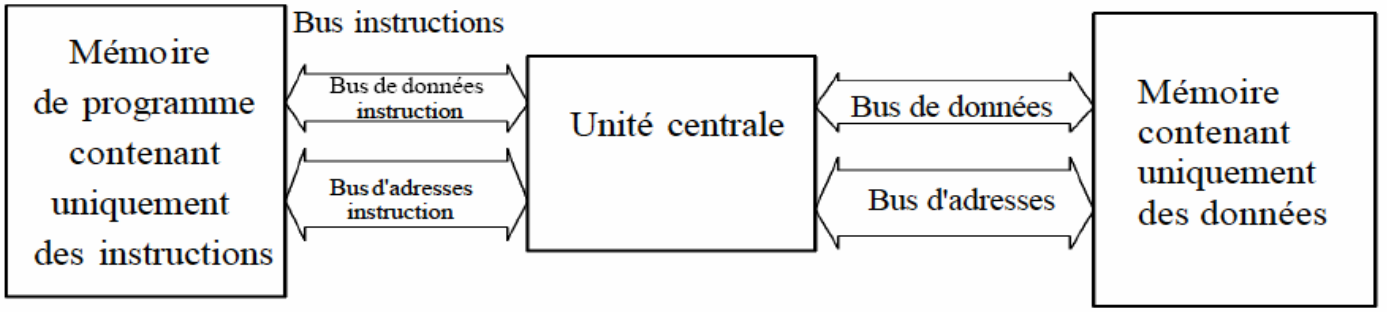
Définition : Une **adresse** est un nombre binaire qui indique un emplacement dans une zone mémoire

- **Structure de Von Neumann**



Extraits du cours intitulé « Les systèmes micro-programmés »

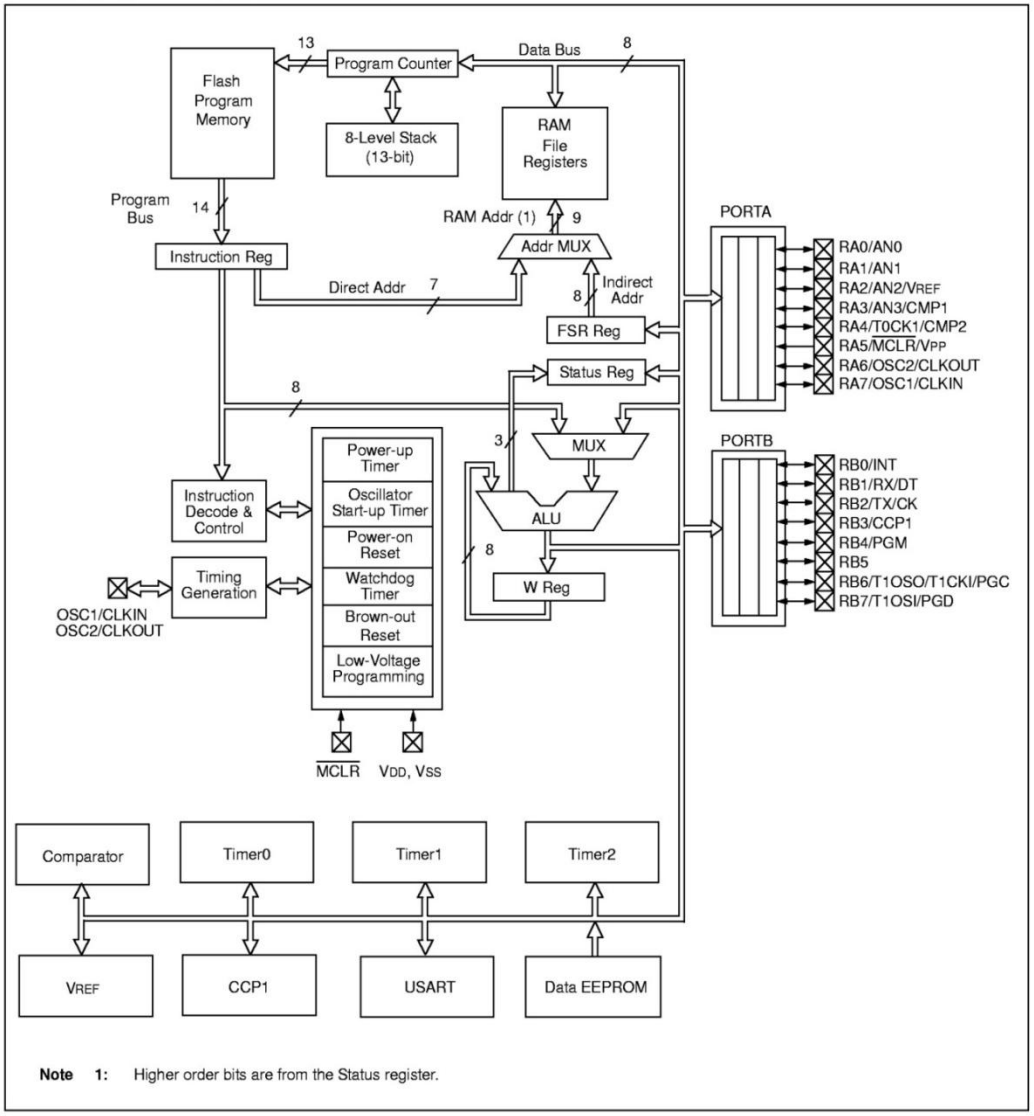
- **Structure de Harvard**



La différence se situe au niveau de la séparation ou non des mémoires programmes et données. La structure de Harvard permet de transférer données et instruction simultanément, ce qui permet un gain de performances.

C.3 Schéma bloc d'un microcontrôleur

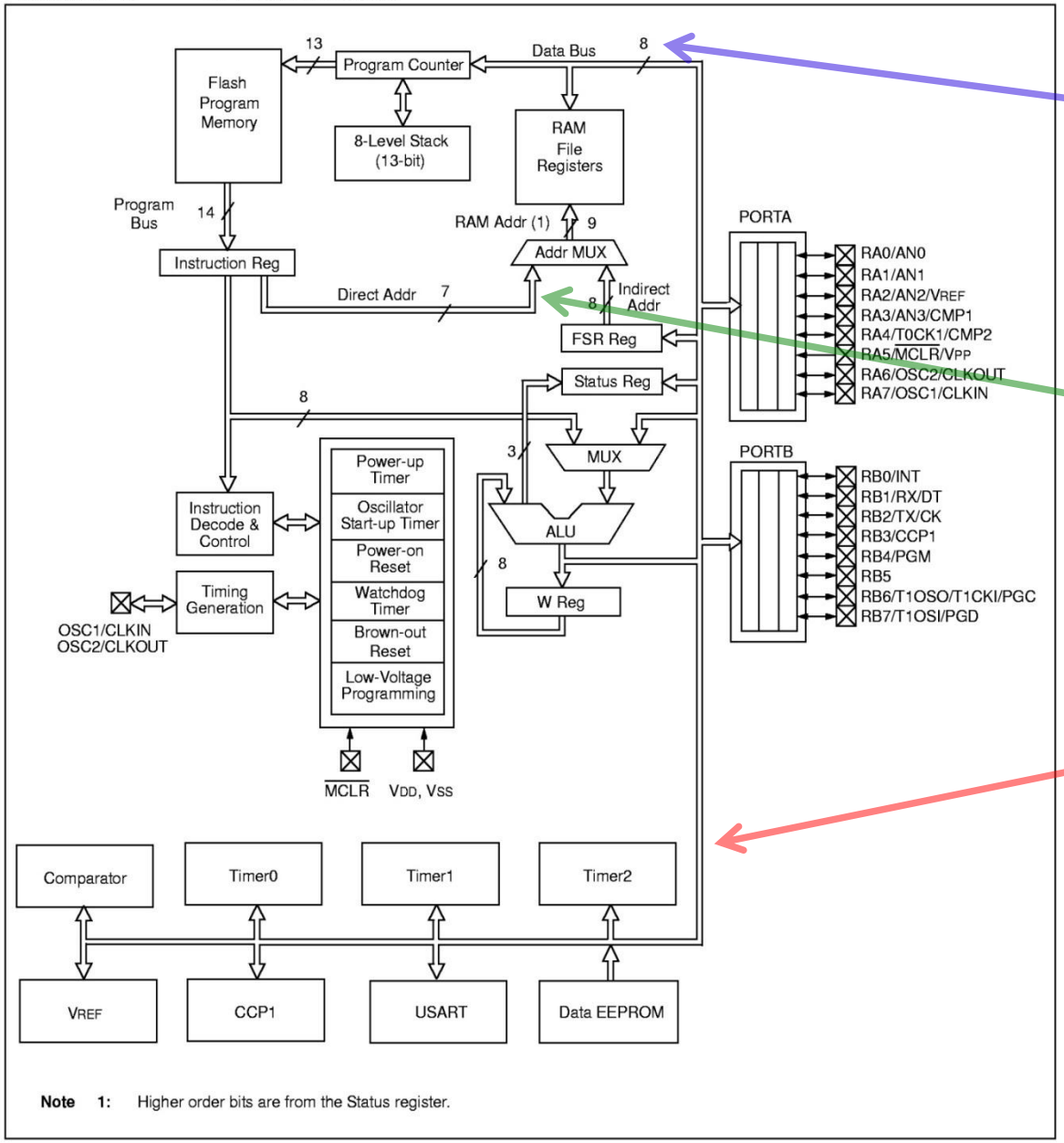
FIGURE 3-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the Status register.

Issu de la documentation technique du PIC16F628

FIGURE 3-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the Status register.

- **« Largeur du bus »**

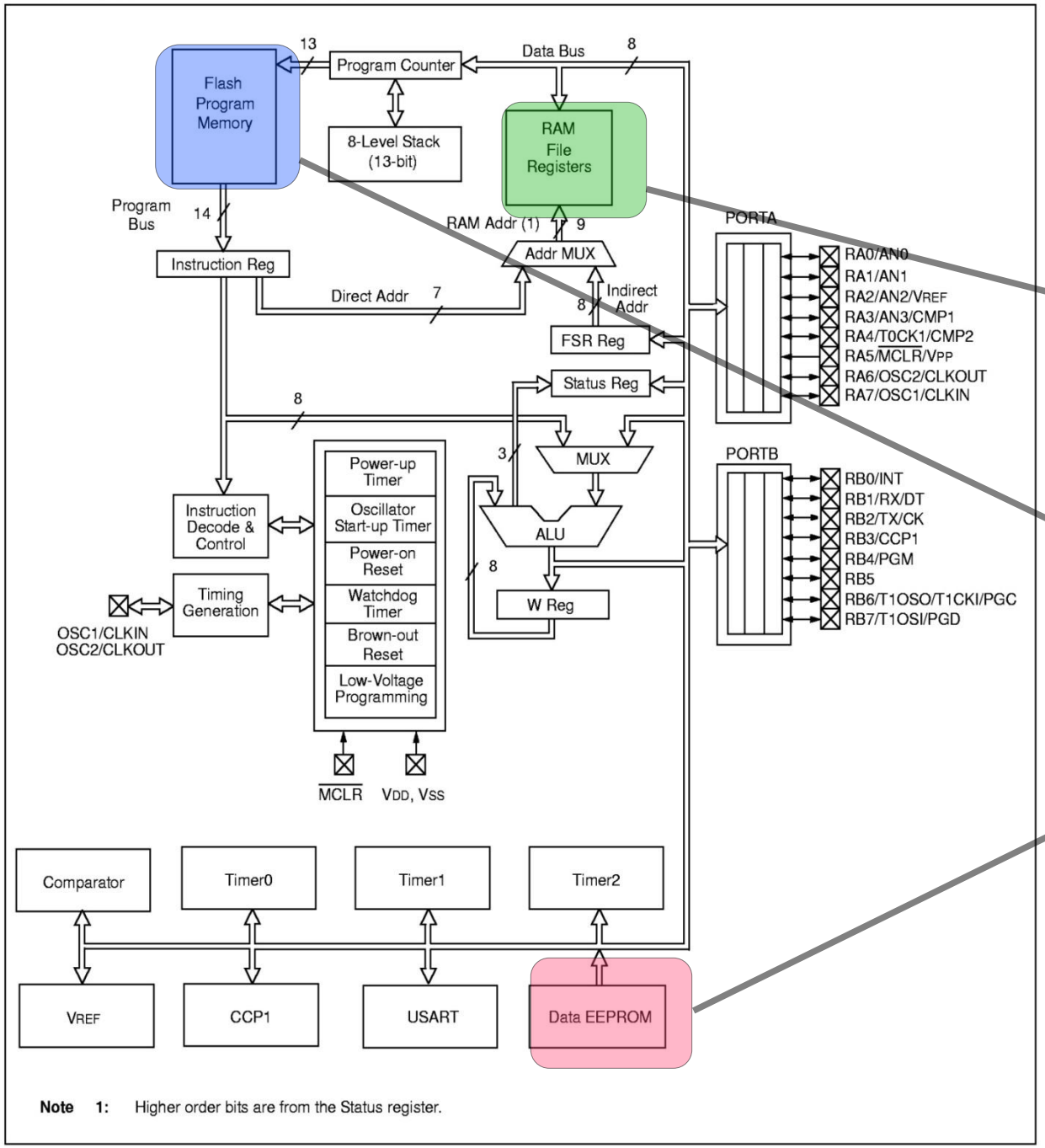
8

- **Unidirectionnel**

- **Bidirectionnel**

Issu de la documentation technique du PIC16F628

FIGURE 3-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the Status register.

Les mémoires :

RAM (Random Access Mem.)

mémoire rapide qui permet de stocker temporairement des données.

ROM (Read Only Memory)

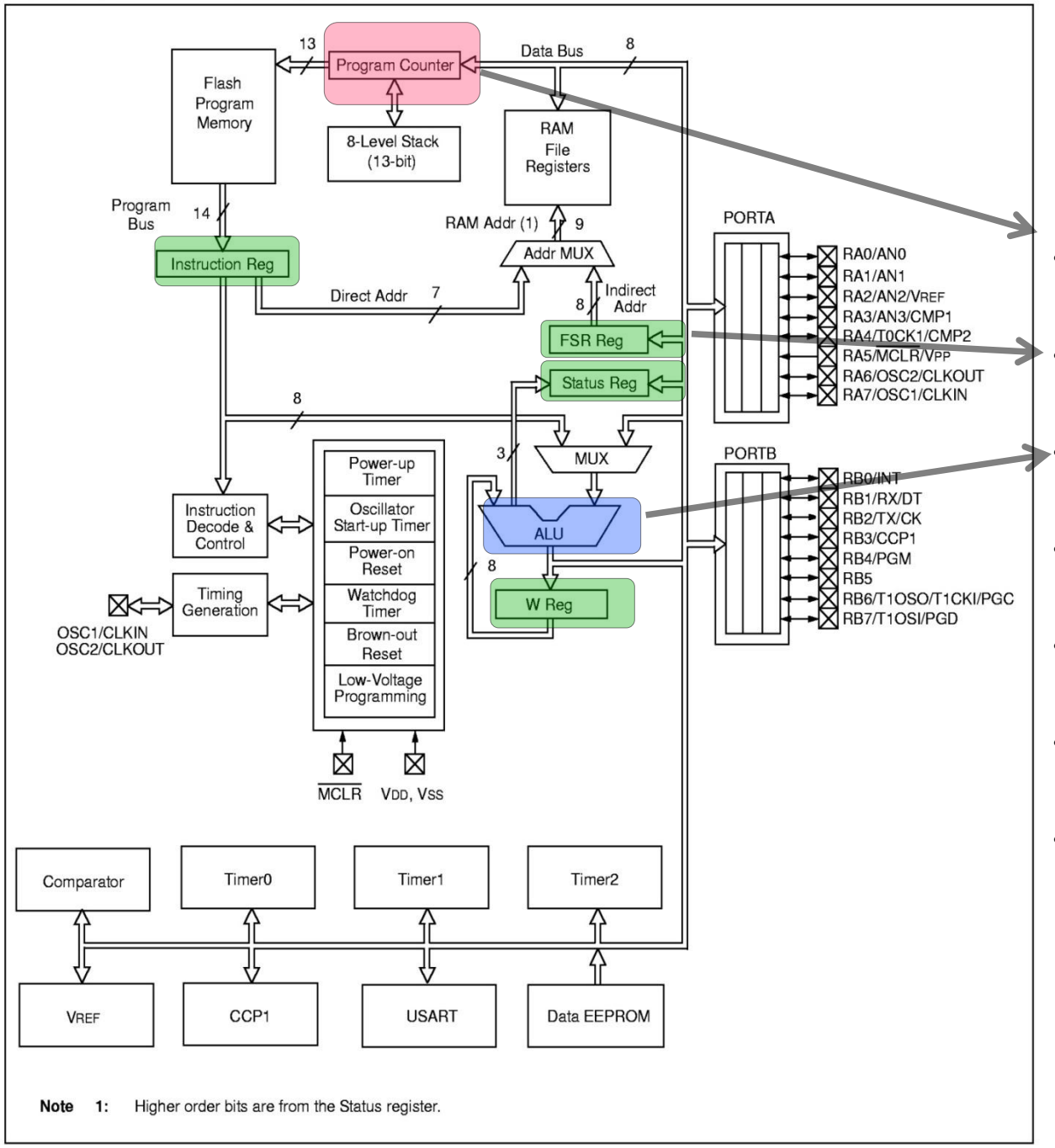
mémoire à lecture seule, programmée à vie.

EEPROM

(Elec. Erasable Programmable Read Only Memory)

mémoire lente qui permet de stocker des données même après coupure de l'alim.

FIGURE 3-1: BLOCK DIAGRAM

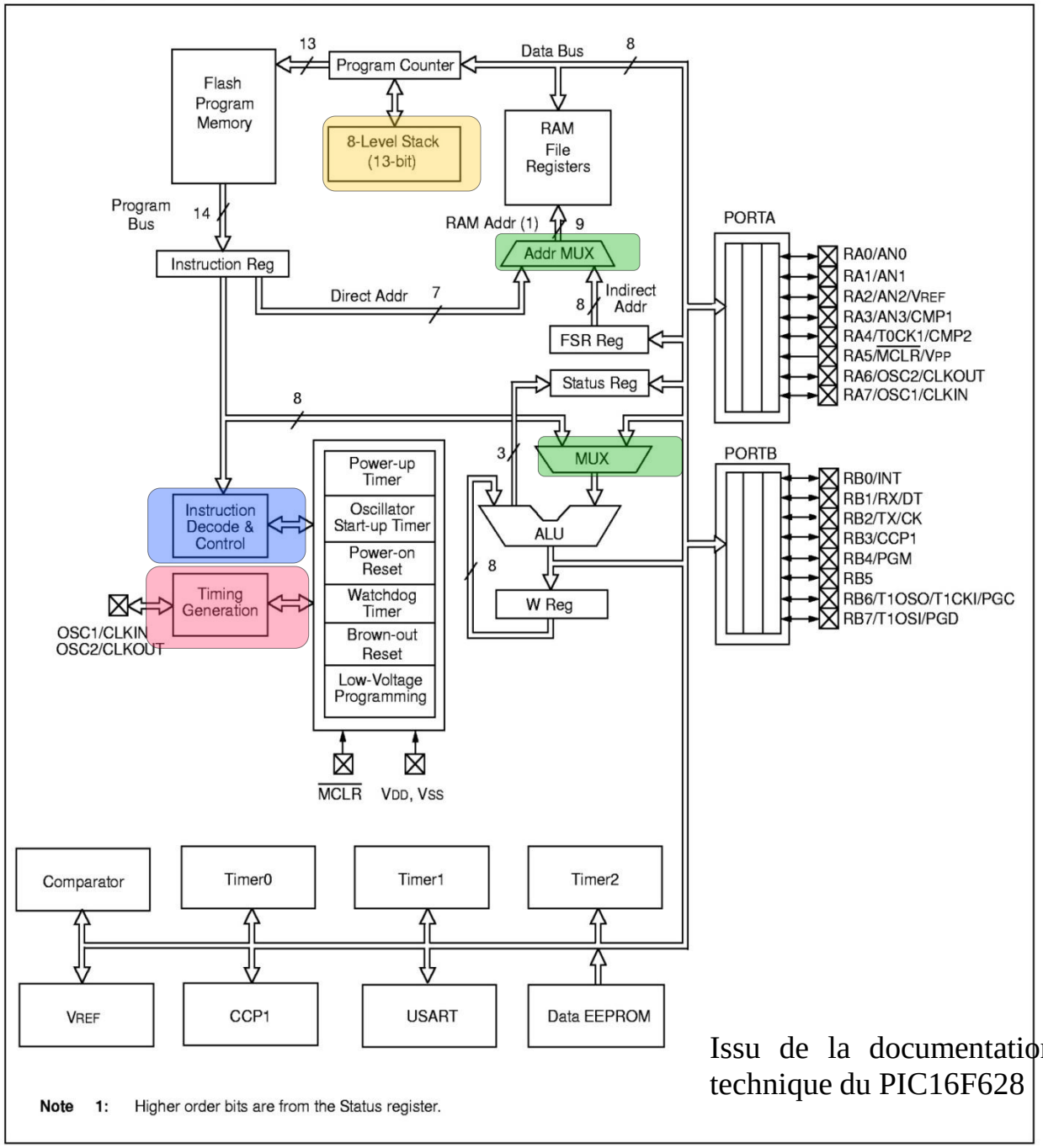


Note 1: Higher order bits are from the Status register.

- **PC** (Program Counter)
- **Registre** (case mémoire)
- **ALU**
- **Multiplexeur**
- **Décodeur d'instructions**
- **horloge**
- **Stack (pile)**

  LIFO (Last In First Out)
  FIFO (First In First Out)

FIGURE 3-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the Status register.

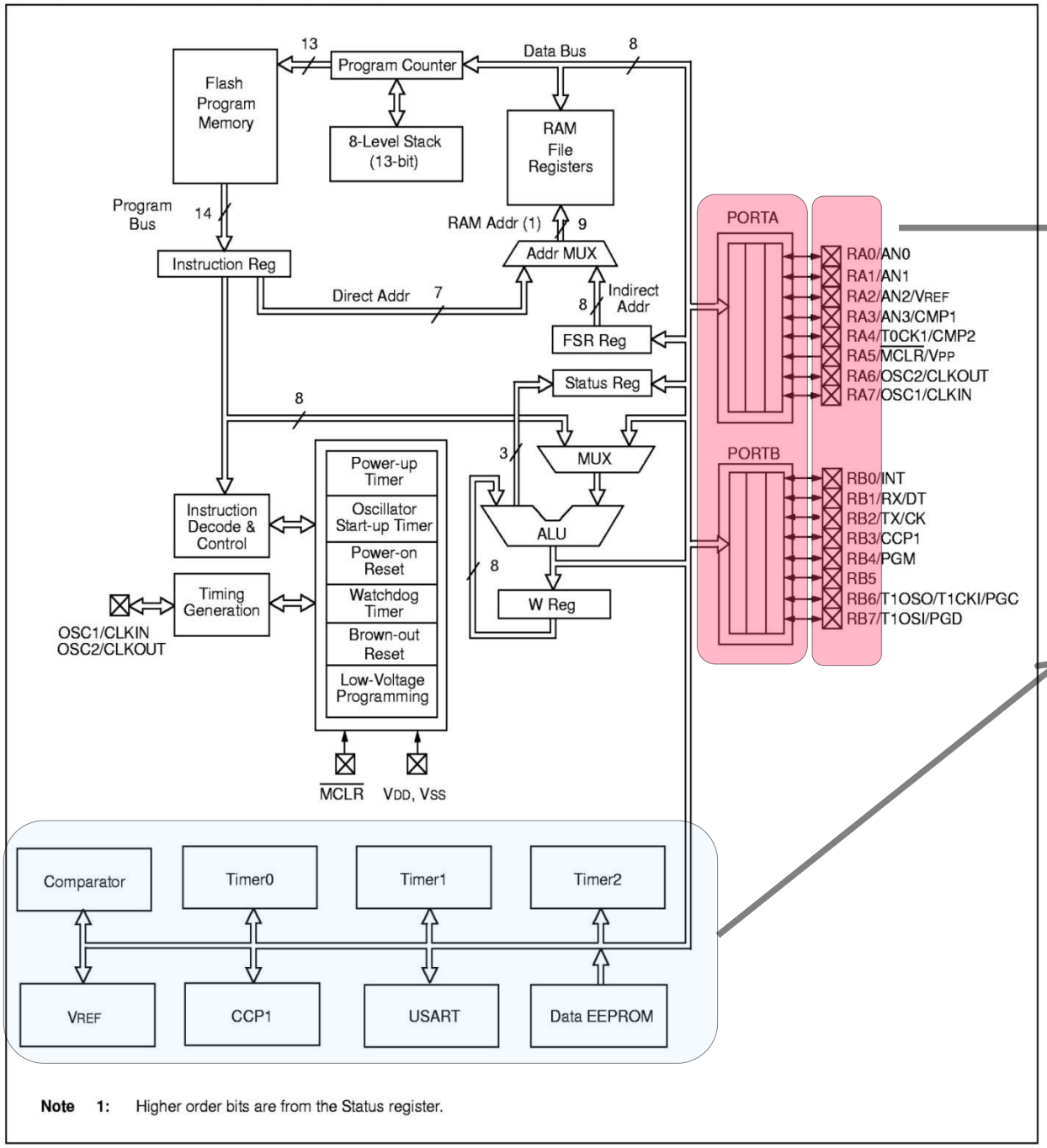
Issu de la documentation technique du PIC16F628

- **Registre** (case mémoire)
- **ALU**
- **PC** (Program Counter)
- **Multiplexeur**
- **Décodeur d'instructions**
- **horloge**
- **Stack (pile)**

LIFO (Last In First Out)

FIFO (First In First Out)

FIGURE 3-1: BLOCK DIAGRAM



Note 1: Higher order bits are from the Status register.

- **Ports d'entrées/sorties**

- **USART**
  (Universal Synchronous Asynch. ReceiverTransmitter)
  interface de communication série,
- **CCP** (Capture/Compare/PWM)
  Modulation en largeur d'impulsion
- **Timer**
- **Comparateur**
- **CAN/CNA**
- **Référence de tension**
- **Module HF**
- **Liaison USB, ...**

C.4 Composants et fonctionnalités: quels critères de choix ?

Architecture :

- ALU (8, 16, 32, 64 bits)
- Structure du processeur (Harvard, Von Neumann)
- Type de processeur (RISC, CISC)
- Taille des mémoires programme et donnée
- Nombre de ports d'entrée/sortie

Fonctionnalités :

- Fonctions *analogiques* : CAN, CNA, Comparateur, ...
- Fonctions de *timing* : Timer, Watchdog, ...
- Fonctions de *communication* : UART (Communication série), USB, I2C, ...
- Facilité de programmation : In-Circuit Serial Programming, Self Programming, ...

Mise en oeuvre, maintenance :

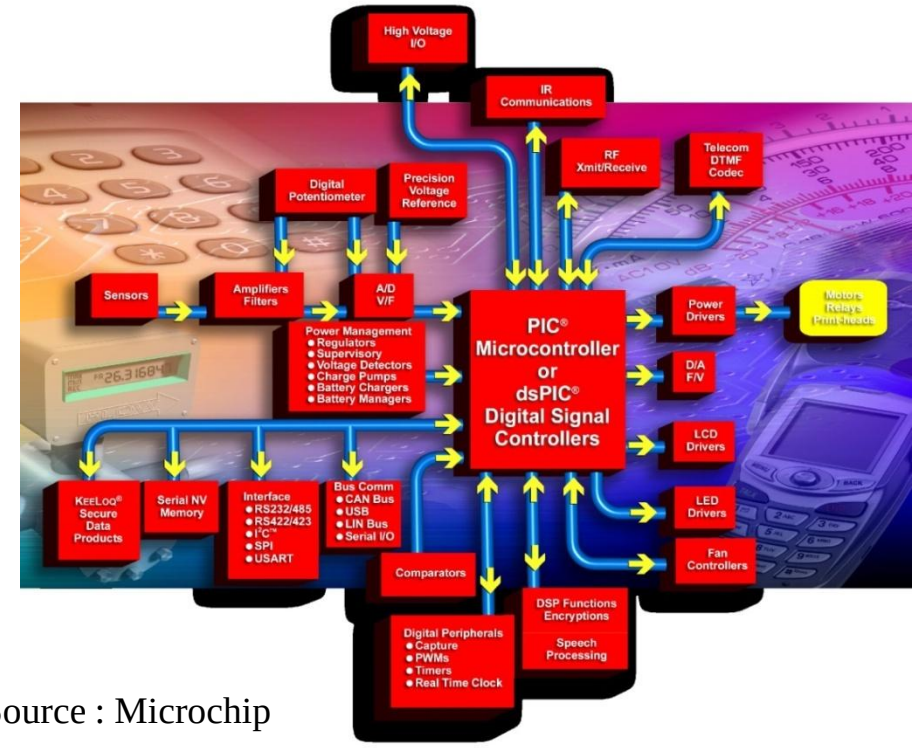
- Coût de développement : outils de développement, formation, ...
- Suivi du micro-contrôleur : production suivie, disponibilité, composant obsolète, ...

Caractéristiques électriques :

- Fréquence d'horloge
- Tensions d'alimentation
- Consommation d'énergie, modes faible consommation d'énergie, ...

Caractéristiques physiques :

- Type de boîtier : DIL, PLCC, ...



Source : Microchip

Plan

Lien avec les TPs

0 Objectifs et plan du cours, lien avec les TPs

I Catégories de systèmes programmables, architecture des microcontrôleurs
- A Présentation de l'informatique industrielle et des systèmes micro-programmés
- B Architecture neuromorphique : comment le cerveau inspire l'informatique industrielle
- C Architecture des microcontrôleurs

II Codage, instructions assembleur, modes d'adressage des instructions
- A Codage binaire et hexadécimal, complément à deux
- B Premier programme assembleur et algorigramme correspondant (TP1)
- C Modes d'adressage des instructions

III Etude du fonctionnement du microcontrôleur
- A Structure des ports entrées sorties
- B Phase de démarrage du microcontrôleur
- C Exécution d'une instruction
- D Les registres
- E Organisation de la mémoire

IV Les interruptions, exemple avec bouton poussoir
- A Interruptions: définition et gestion (TP2)
- B Interruption par appui sur bouton poussoir

Binaire, octal, décimal et hexadécimal

On rappelle tout d'abord les différentes bases qui nous seront utiles :

le **binaire** (base 2) est constitué de 2 chiffres :

0, 1

l'**octal** (base 8), 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** (base 16), 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** (base 10) est constitué de 3 chiffres :

- le chiffre 8 est affecté du poids de 1 (unités)
- le chiffre 2 est affecté du poids de 10 (dizaines)
- le chiffre 1 est affecté du poids de 100 (centaines)

Le nombre peut donc s'écrire

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

Codes pondérés

Le nombre 10 **binaire** (base 2) 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

Remarque : le nombre 10 **binaire** que l'on lit « Un Zéro » *ne s'exprime pas* dix car ceci sous-entendrait que le nombre est exprimé en décimal...

Le nombre peut donc s'écrire

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

Chiffre

Poids

Codes pondérés

Le nombre 1F8 **hexadécimal** (base 16) 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

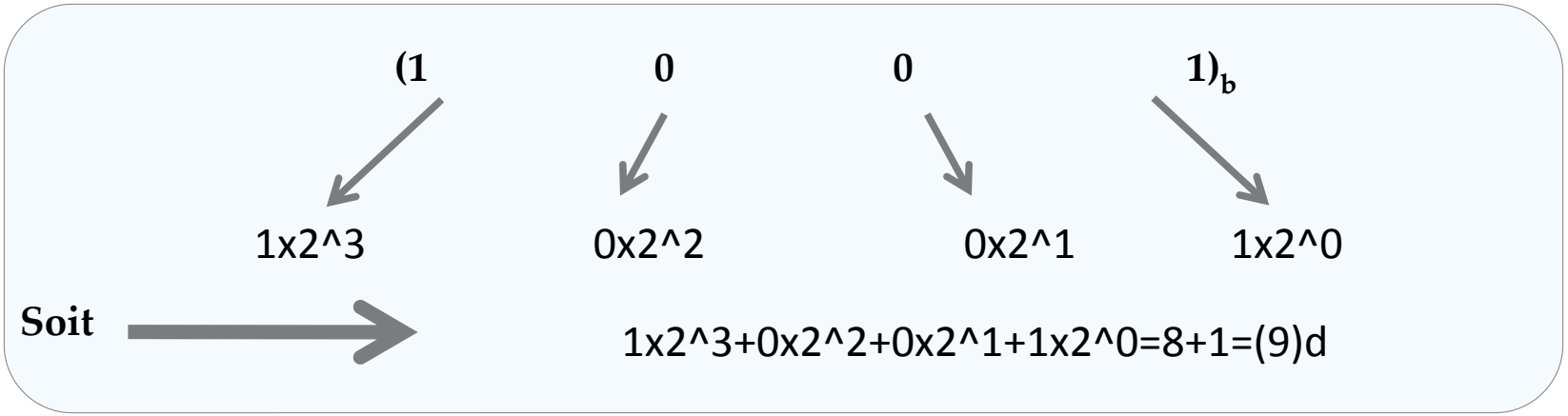
$$1 \times 256 + F \times 16 + 8 \times 1 = (\mathbf{1F8})_h$$

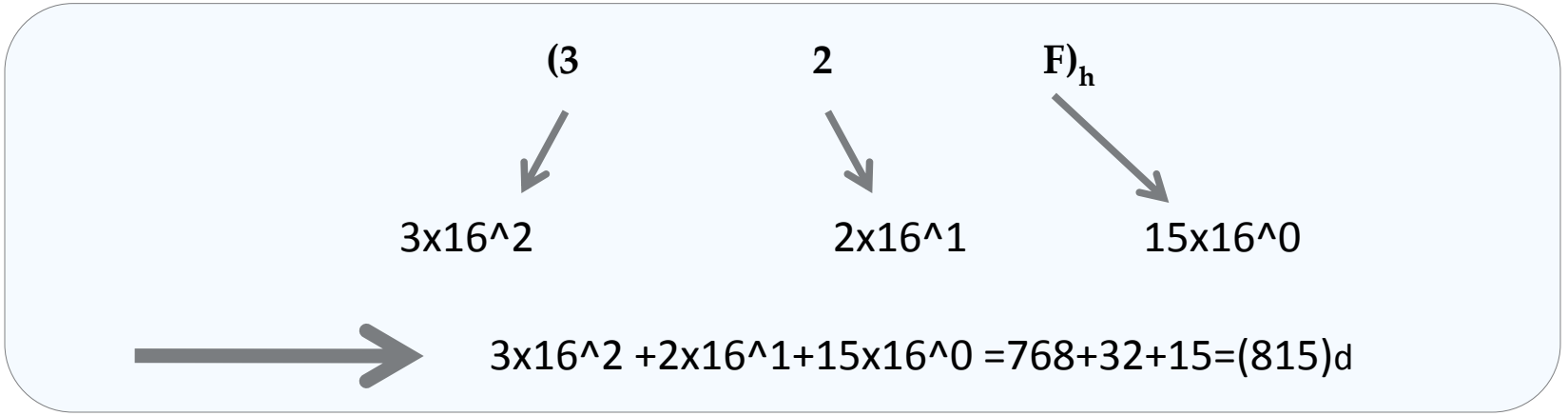
Conversion binaire-hexadécimal : le codage hexadecimal a été créé afin d'alléger l'exploitation des nombres binaires. Il permet en particulier une **conversion simple** par *regroupement des bits par 4 en partant de la droite*, chaque paquet étant alors simple à convertir :

0 0 0 1	1 1 1 1	1 0 0 0	
1	**F**	**8**	= $(\mathbf{1F8})_h$

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

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 = A x B, ***Exemple*** **:** $(0110)_b . (0101)_b = (011110)_b$

NOTEZ BIEN QUE...

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 un par des zéros (et vice-versa), puis ajouter 1.

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

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

Justification, principe et calcul du complément à 2

Un microprocesseur n'effectue que des décalages et des additions.

Pour calculer 'a-b' on ajoute à 'a' le terme (2^n - b) (le complément à 2^n de b).

a-b devient a+(2^n - b) = 2^n + (a-b)

i)complémenter à 1 tous les bits
ii)ajouter 1
Exemple :

010 '2' codé sur n=3 bits

↓ + 101

111 Le complément à 1 permet de passer à 111

↓ + 001

1000 Ajouter 1 permet de passer à 2^n

Le complément à 2^n de $(010)_b$ est donc bien $(101)_b + (001)_b = (110)_b$.

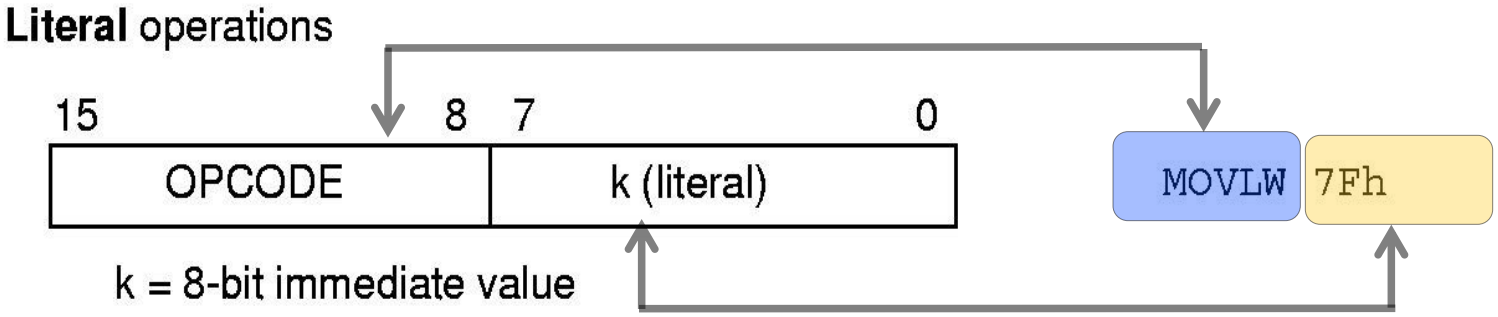
NOTEZ BIEN QUE...

On fixe au départ le nombre de bits qui sont nécessaires pour représenter les nombres considérés.

Une instruction est composée au minimum de deux parties:

Instruction = OPCODE + opérande(s)

OPCODE (Operation CODE) : partie d'une instruction qui précise quelle opération doit être réalisée



Extrait du datasheet (documentation technique) du PIC18F4520.

Les types d'instructions en Assembleur

A. Les instructions propres au micro-contrôleur :

- Les instructions logiques : xorlw, andlw, ...
- Les instructions de transfert : movlw, movf, ...
- Les instructions arithmétiques : decf, addwf, ...
- Les instructions de branchement : bz (*branch if zero*), bra (*branch always*), ...

B. Les instructions pré-processeur

Elles permettent au programmeur de donner des indications au compilateur.

Elles sont destinées au PC et non pas au micro-contrôleur !

- Instructions de contrôle : org = début du programme, end = fin du programme, *etc.* ;
- Instructions conditionnelles : if, else, endif, *etc.* ;
- Instructions relatives aux données : res = réservation d'espace mémoire, *etc.*

Instructions logiques

ANDLW et logique entre un nombre (‘literal’) et le registre w

OU-EXCLUSIF et inversion de valeur bit

A	B	XOR
0	0	0
1	0	1
0	1	1
1	1	0

Instruction correspondante en assembleur:

xorwf REG

Ou exclusif entre le registre ‘w’ et le registre ‘REG’
Le résultat est placé dans ‘REG’

Jeu d'instructions

Un **jeu d'instructions** est un ensemble d'opérations directement réalisables sur un système micro-programmé donné.

Par exemple : le PIC18F4520 (RISC) possède un jeu d'instructions composé de 75 instructions. L'exécution d'une instruction peut nécessiter un ou plusieurs **cycles d'horloges** suivant la complexité de l'instruction.

NOTE : Un **cycle d'horloge** correspond à une période de **l'horloge** (signal de référence temporelle). La **fréquence d'horloge** est le nombre de cycles effectués par une horloge en une seconde.

4 étapes pour l'exécution d'une instruction :

A1 : Fetch, pointer l'instruction dans la mémoire programme
A2 : Decode, décoder dans le décodeur d'instructions
A3 : Execute, calculer dans l'UAL
A4 : Write Back, écrire le résultat dans W

•Exécution séquentielle classique:

A1	A2	A3	A4	A1	A2	A3	A4	…

8 coups d'horloge pour 2 instructions

•Pipeline à 4 étages

(1)	A1	A2	A3	A4	…				
(2)		A1	A2	A3	A4	…			
(3)			A1	A2	A3	A4	…		
(4)				A1	A2	A3	A4	…	

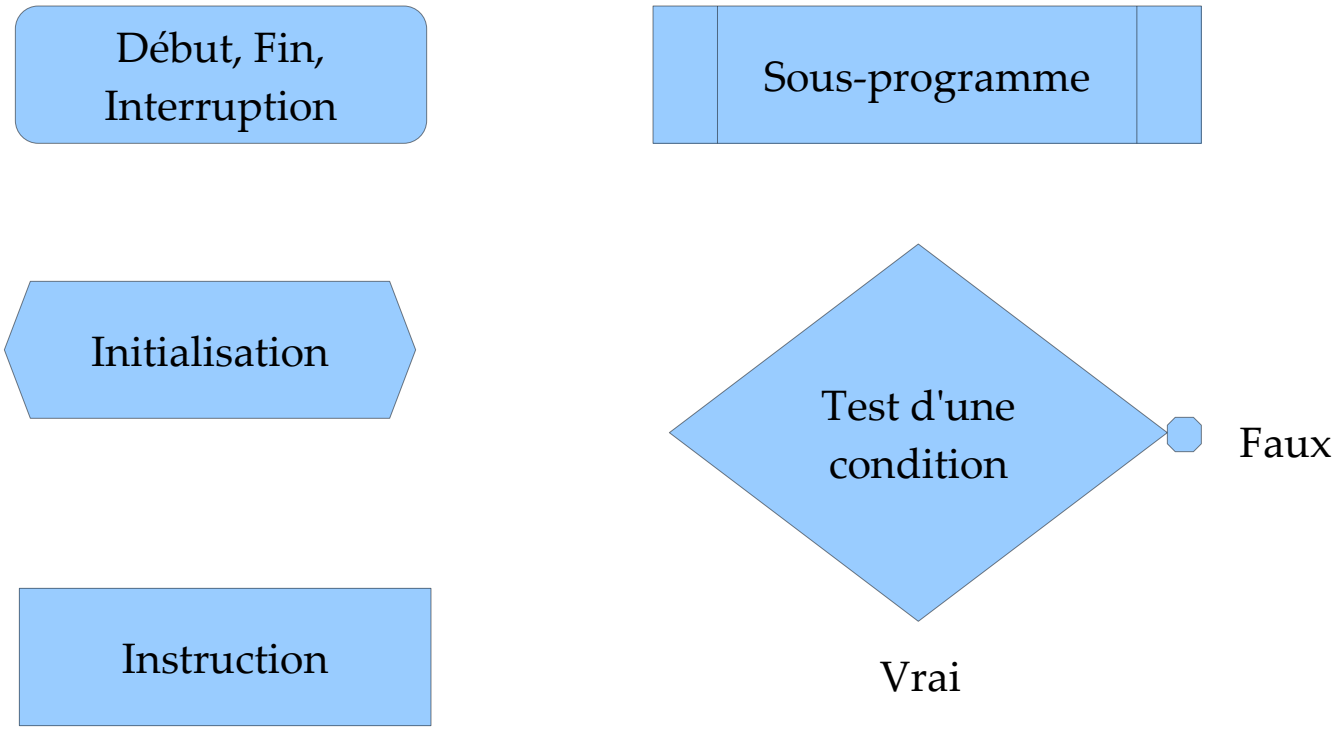
7 coups d'horloge pour 4 instructions

Pipeline à plusieurs étages: exécution plus rapide des instructions
14 à 19 étages pour IvyBridge (version 2014)

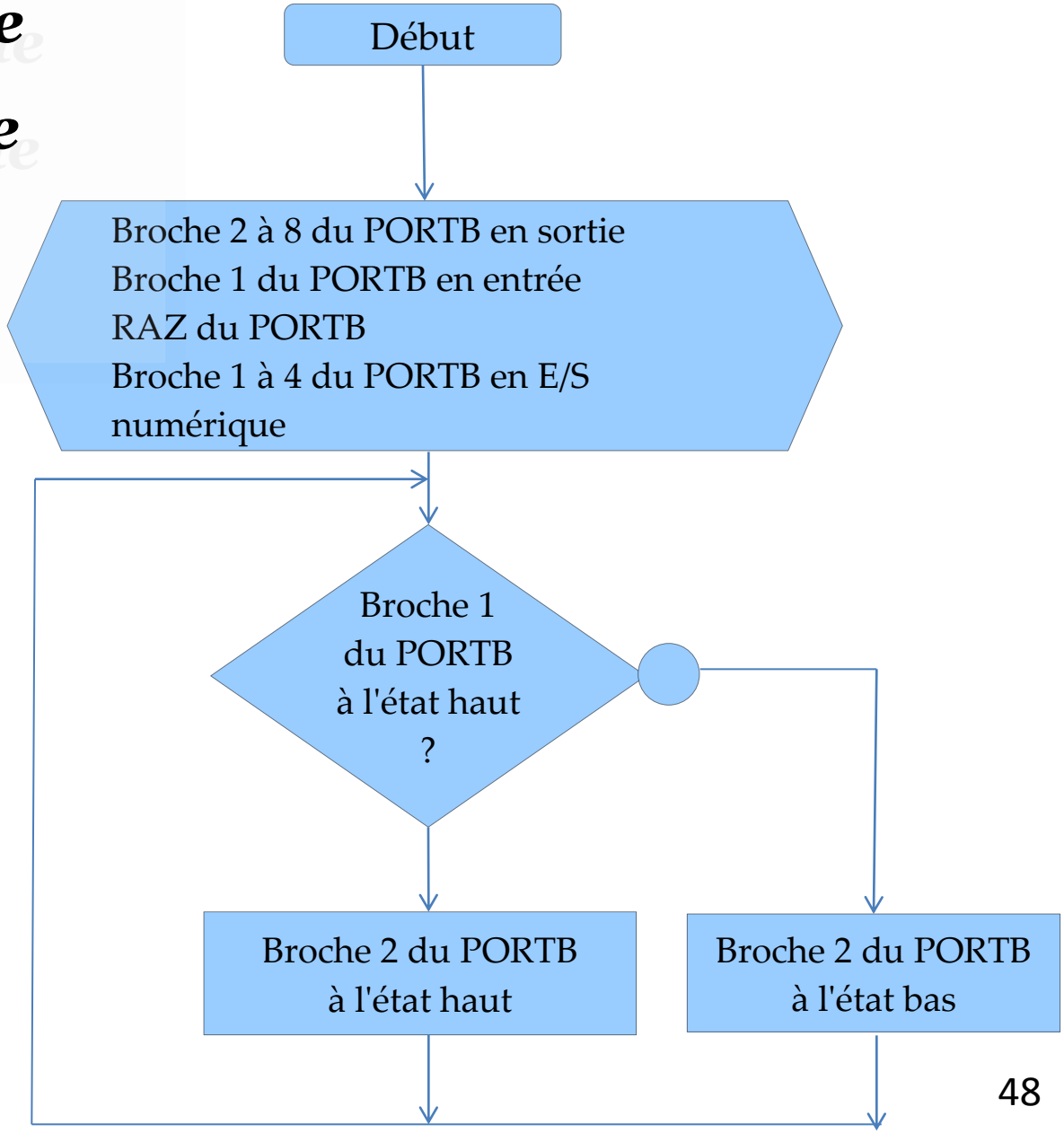
Algorigrammes

La description du programme par un algorigramme permet de :

- **gagner en efficacité** lors de la phase de codage du programme,
- d'**optimiser la structure** du programme,
- de **clarifier le fonctionnement** du programme,
- **le rendre compréhensible** à une personne extérieure.



Premier algorigramme et programme assembleur associé

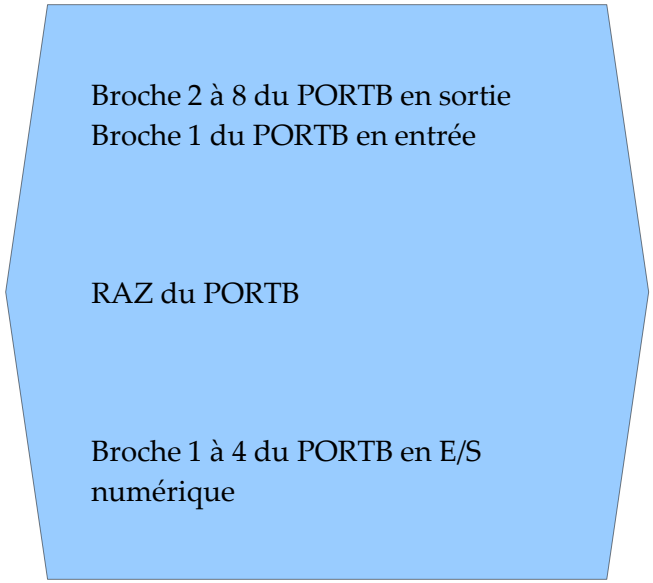


La première opération consiste systématiquement à **initialiser le vecteur RESET**.

Début

```
org     h'0000'; initialisation du vecteur RESET
goto init
```

La deuxième opération consiste à initialiser le PORTB et sa directionnalité (Entrée ou Sortie)

Algorigramme	Programme assembleur
	

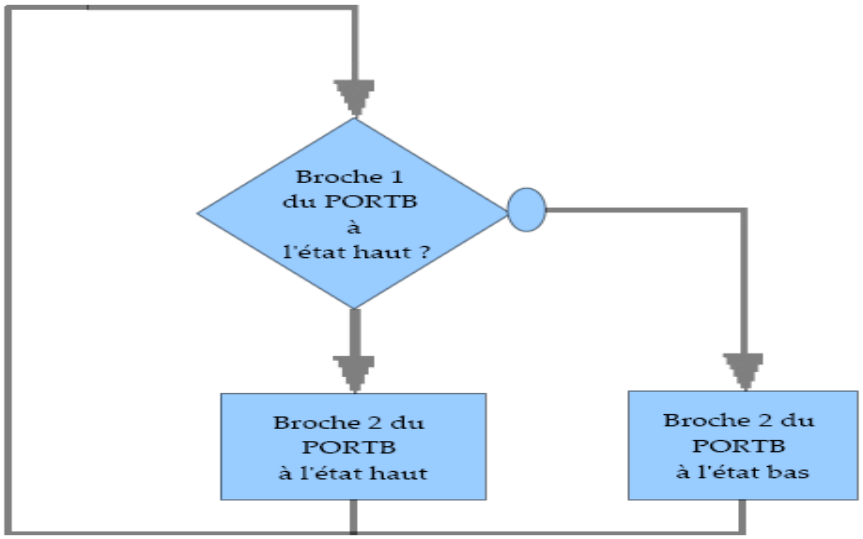

```
init    movlw   b'00000001'
        movwf   TRISB

        clrf    PORTB

        movlw   0Fh
        movwf   ADCON1
```

La troisième partie du programme est dédiée à la réalisation de la fonction principale, c.à.d. la boucle et le test.

Algorigramme	Programme assembleur

	```
	boucle btfss PORTB,0
	
	; btfss: saute l'instruction
	;suivante si état haut
	
	                   goto eteindre
	
	allumer bsf PORTB,1
	                   goto boucle
	
	eteindre bcf PORTB,1
	                   goto boucle
	``

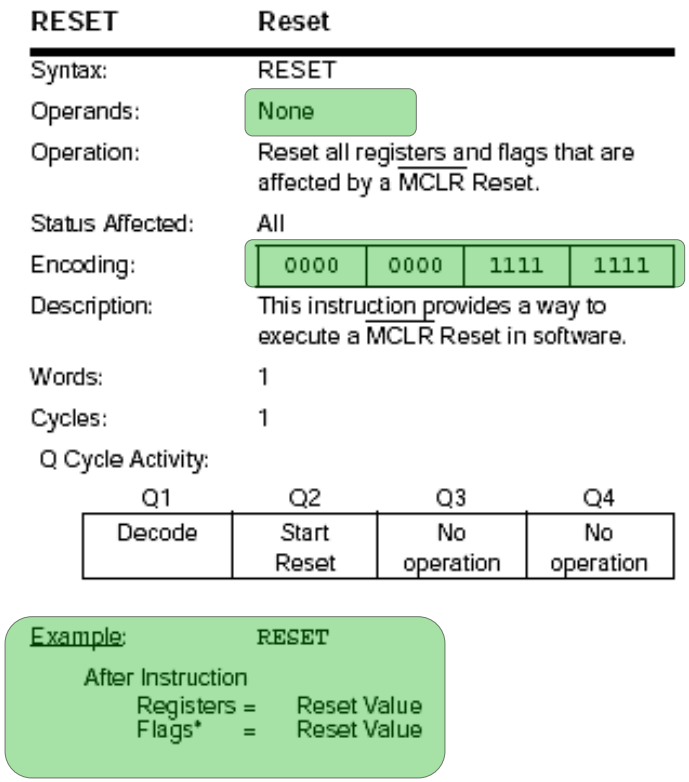
La nature et le nombre d'opérandes qui constituent une instruction déterminent le **mode d'adressage** de l'instruction. *On distingue 4 modes d'adressage principaux.*

L'adressage inhérent : il n'y a pas d'opérande !

ex : NOP, RESET, CLRWDT ;

Description de l'instruction RESET extraite de la notice technique (le « datasheet ») du PIC 18F4520 [micro-contrôleur utilisé en TP].

ATTENTION : Un nombre important d'information utiles figure sur ces fiches...



RESET	Reset
Syntax:	RESET
Operands:	None
Operation:	Reset all registers and flags that are affected by a MCLR Reset.
Status Affected:	All
Encoding:	0000 0000 1111 1111
Description:	This instruction provides a way to execute a MCLR Reset in software.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start Reset	No operation	No operation

Example: RESET

After Instruction
Registers = Reset Value
Flags* = Reset Value

La nature et le nombre d'opérandes qui constituent une instruction déterminent le **mode d'adressage** de l'instruction. *On distingue 4 modes d'adressage principaux.*

L'adressage immédiat (literal en anglais):
l'opérande est une valeur

ex : MOVLW 5Ah ; W=5Ah

Nombres de cycles nécessaires à l'exécution

Exécution de l'instruction (pipeline à 4 niveaux)

MOVLW **Move literal to W**

Syntax:	MOVLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → W
Status Affected:	None
Encoding:	0000 1110 kkkk kkkk
Description:	The eight-bit literal 'k' is loaded into W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: MOVLW 5Ah

After Instruction

W = 5Ah

La nature et le nombre d'opérandes qui constituent une instruction déterminent le **mode d'adressage** de l'instruction. *On distingue 4 modes d'adressage principaux.*

L'adressage direct (étendu) : l'opérande est constituée des bits de poids faible de l'adresse de la donnée dans la page mémoire active.

Exemple 1 :

Movlw 0x55
Movwf 0x40 place 0x55 à l'emplacement RAM 0x40

Exemple 2 :

MOVWF PORTA, ➡ PORTA est une adresse

En **mode direct étendu** : on transmet l'adresse complète

ANDWF **AND W with f**

Syntax:	ANDWF f {,d {,a}}
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(W) .AND. (f) → dest
Status Affected:	N, Z
Encoding:	0001 01da ffff ffff
Description:	The contents of W are AND'ed with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

La nature et le nombre d'opérandes qui constituent une instruction déterminent le **mode d'adressage** de l'instruction. *On distingue 4 modes d'adressage principaux.*

- **l'adressage indirect :** l'opérande est l'adresse d'un registre qui contient l'adresse de la donnée.

Exemple:

```
MOVLW 0x55
LFSR  0, 0x40  ; le registre FSR0 est chargé avec la valeur d'adresse 0x40
MOVWF INDF0  ; INDF0 est l'adresse de FSR0.
                    place 0x55 à l'emplacement RAM sur lequel pointe FSR0 (i.e.
                    l'adresse 0x40)
```

- En **mode indirect indexé**, on ajoute un décalage par rapport à l'adresse.

NOTE : Il existe de nombreux autres modes d'adressage (ex. implicite, relatif) : leur nombre varie en fonction du constructeur et du micro-contrôleur !

Plan

Lien avec les TPs

0 Objectifs et plan du cours, lien avec les TPs

I Catégories de systèmes programmables, architecture des microcontrôleurs
- A Présentation de l'informatique industrielle et des systèmes micro-programmés
- B Architecture neuromorphique : comment le cerveau inspire l'informatique industrielle
- C Architecture des microcontrôleurs

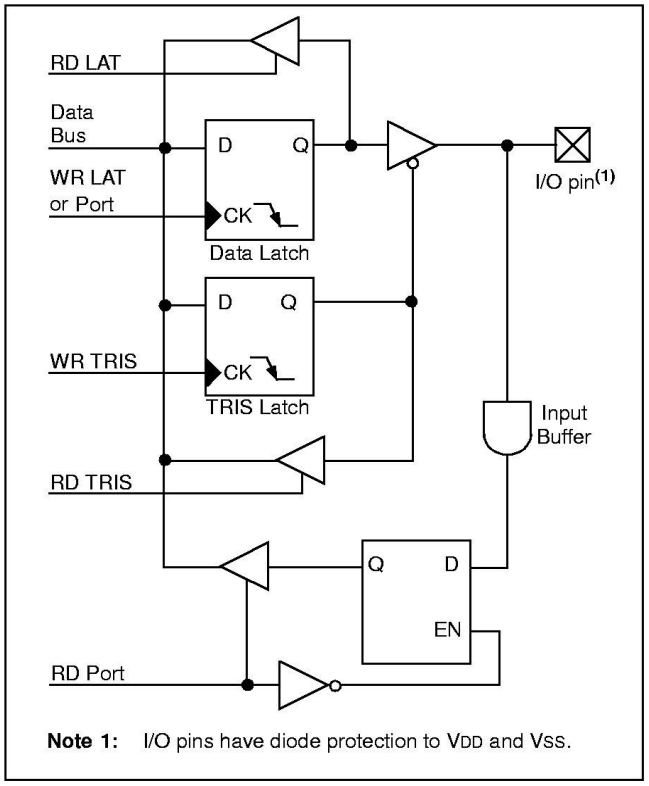
II Codage, instructions assembleur, modes d'adressage des instructions
- A Codage binaire et hexadécimal, complément à deux
- B Premier programme assembleur et algorigramme correspondant (TP1)
- C Modes d'adressage des instructions

III Etude du fonctionnement du microcontrôleur
- A Structure des ports entrées sorties
- B Phase de démarrage du microcontrôleur
- C Exécution d'une instruction
- D Les registres
- E Organisation de la mémoire

IV Les interruptions, exemple avec bouton poussoir
- A Interruptions: définition et gestion (TP2)
- B Interruption par appui sur bouton poussoir

Un port d'entrées/sorties est par définition un port bidirectionnel.

De fait, il est donc nécessaire de configurer **la direction** du port (*in* ou *out*). Dans le microcontrôleur, des registres spécifiques sont dédiés à la gestion de ces ports...



EXAMPLE 10-3: INITIALIZING PORTC

```
CLRF    PORTC   ; Initialize PORTC by
                ; clearing output
                ; data latches
CLRF    LATC    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISC   ; Set RC<3:0> as inputs
                ; RC<5:4> as outputs
                ; RC<7:6> as inputs
```

Extrait de la documentation technique du PIC18F4520 de Microchip.

Suite à une opération de remise à zéro (RESET), le micro-contrôleur effectue une phase de démarrage :

1/ RESET : il peut être déclenché par la mise sous tension du micro-contrôleur, la réception d'un signal sur la broche RESET du micro-contrôleur, une instruction de RESET, ...

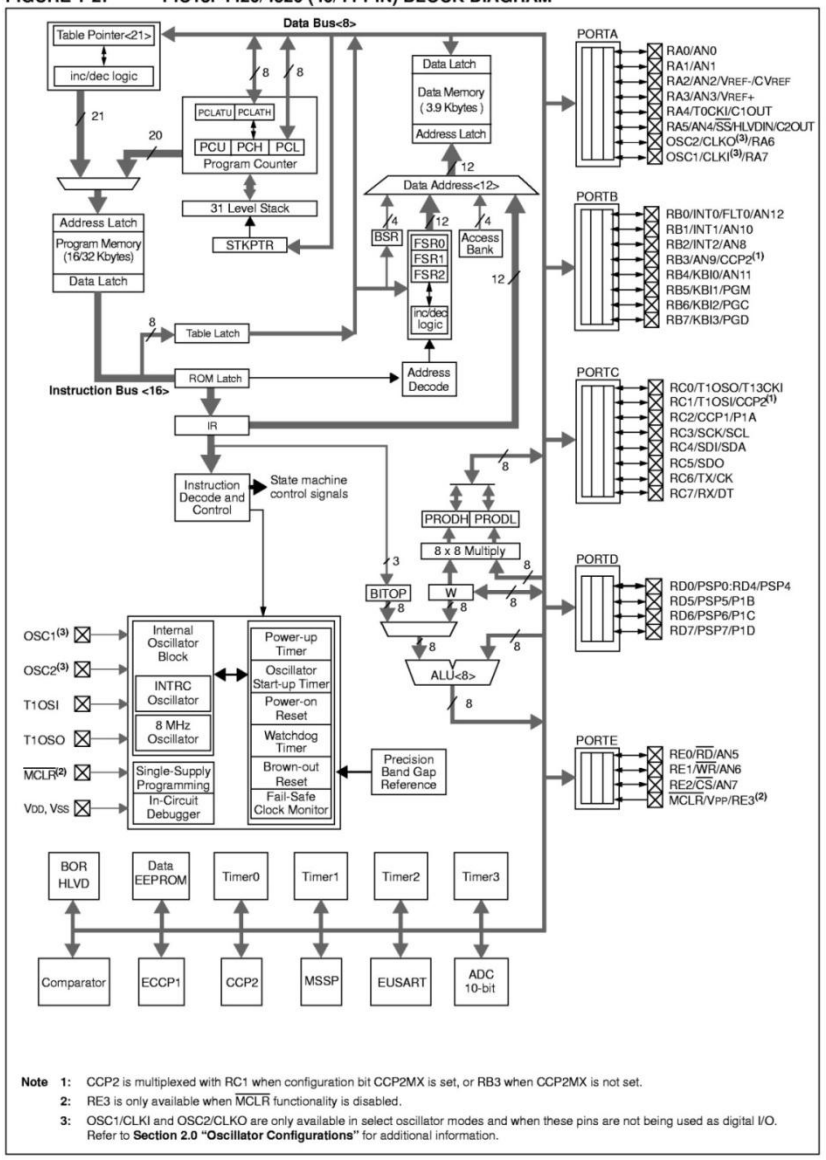
2/ Initialisation du micro-contrôleur : le micro-contrôleur effectue une temporisation afin de garantir la stabilité des signaux d'horloge.

3/ Effacement des registres : le micro-contrôleur efface le contenu des registres (variable en fonction du « mode de RESET » que vous effectuez).

4/ Lecture du vecteur RESET
Le micro-contrôleur lit l'adresse du programme principal dans la mémoire programme.

5/ Début de l'exécution du programme principal.

FIGURE 1-2: PIC18F4420/4520 (40/44-PIN) BLOCK DIAGRAM



Note 1: CCP2 is multiplexed with RC1 when configuration bit CCP2MX is set, or RB3 when CCP2MX is not set.
2: RE3 is only available when MCLR functionality is disabled.
3: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

Adressage inhérent

L'instruction ne comporte pas d'opérande et agit implicitement sur un registre.

Exemples : SLEEP, RESET, NOP

Adressage immédiat

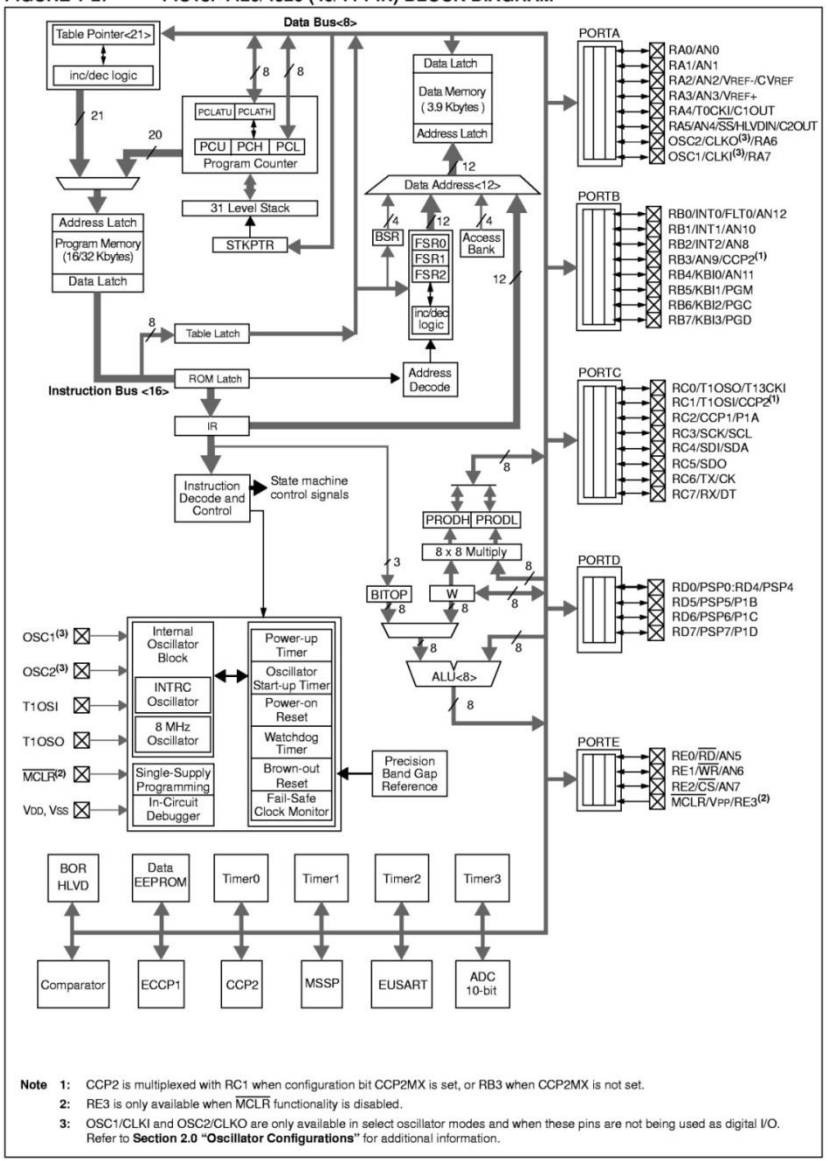
L'instruction comporte une opérande et agit explicitement sur un registre

Exemples : ADDLW, MOVLW

Déroulement:

(1) Le compteur programme indique l'adresse de l'instruction suivante dans la mémoire programme. (2) L'instruction est lue et stockée dans le registre d'instruction. (3) Puis elle est décodée par le module de décodage et de contrôle des instructions. (4) Finalement elle est exécutée.

FIGURE 1-2: PIC18F4420/4520 (40/44-PIN) BLOCK DIAGRAM



Note 1: CCP2 is multiplexed with RC1 when configuration bit CCP2MX is set, or RB3 when CCP2MX is not set.
2: RE3 is only available when MCLR functionality is disabled.
3: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

Adressage direct (étendu)

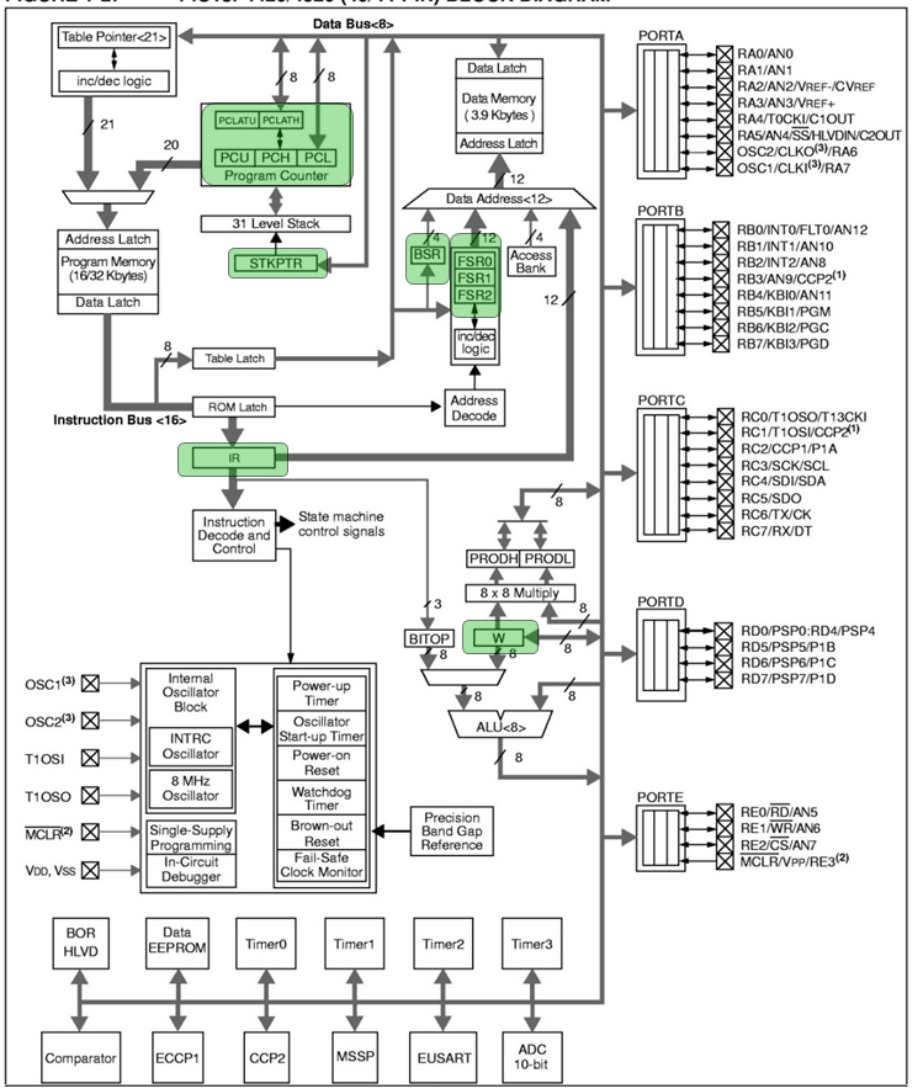
L'instruction comporte une opérande qui indique l'adresse mémoire sur laquelle s'effectue l'opération.

Exemples : CLRF (direct), MOVFF (étendu)

Déroulement:

(1) Lecture de l'instruction dans la mémoire programme à l'adresse pointée par le compteur programme. (2) Lecture de l'instruction et décodage. (3) Pour l'adressage direct, l'opérande constitue la partie basse de l'adresse mémoire sur laquelle s'effectue l'opération, la partie haute est complétée avec le registre BSR. (3') Pour l'adressage étendu, l'opérande est l'adresse complète de la case mémoire sur laquelle s'effectue l'opération. (4) Finalement l'instruction est exécutée sur la case mémoire pointée.

FIGURE 1-2: PIC18F4420/4520 (40/44-PIN) BLOCK DIAGRAM



Un **registre** 8 bits est synonyme d'un ensemble de 8 cases mémoire. De nombreux registres sont utilisés pour gérer le microcontrôleur.

Le registre W (accumulateur)

Le compteur programme (PC)

Le registre d'état (Flags)

Les registres de configuration :

les registres de directions pour les ports d'entrées/sorties (TRIS, SFR), les registres de gestion des interruptions, de gestion de la mémoire (BSR, GPR, *etc.*)

ATTENTION : tous les registres du microcontrôleur ne sont pas représentés sur le schéma...

Par exemple : le registre d'état

Le registre d'état (Status Register) contient des bits d'informations sur les opérations arithmétiques menées par l'ALU (ex., le dépassement de format après avoir demandé l'addition de deux valeurs 8 bits).

STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative (ALU MSB = 1).
1 = Result was negative
0 = Result was positive

bit 3 **OV:** Overflow bit
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude which causes the sign bit (bit 7 of the result) to change state.
1 = Overflow occurred for signed arithmetic (in this arithmetic operation)
0 = No overflow occurred

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

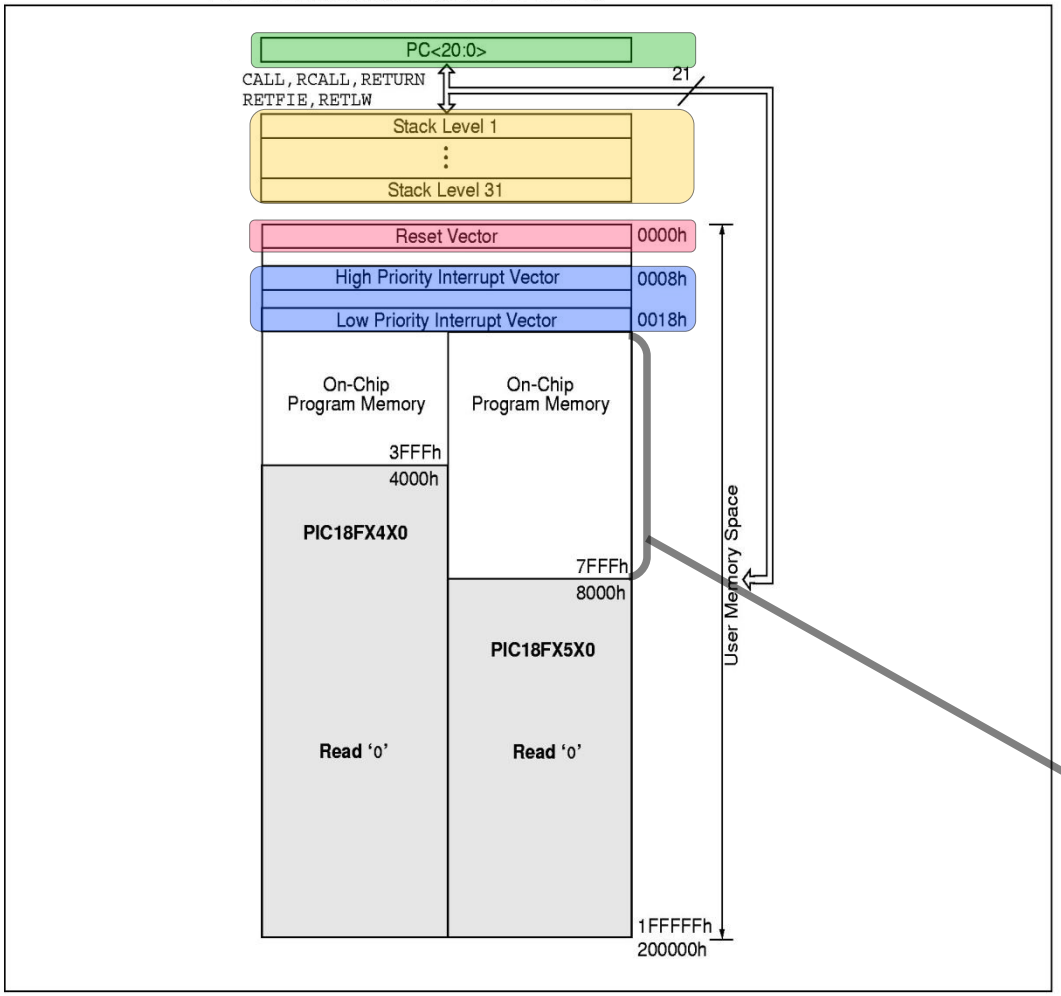
bit 1 **DC:** Digit Carry/$\overline{\text{borrow}}$ bit
For ADDWF, ADDLW, SUBLW and SUBWF instructions:
1 = A carry-out from the 4th low-order bit of the result occurred
0 = No carry-out from the 4th low-order bit of the result

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the 2's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either bit 4 or bit 3 of the source register.

bit 0 **C:** Carry/$\overline{\text{borrow}}$ bit
For ADDWF, ADDLW, SUBLW and SUBWF 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

E.1 Mémoire programme

FIGURE 5-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F2420/2520/4420/4520 DEVICES



Compteur de programme (PC)

le compteur de programme

Pile (Stack)

une pile pour gérer les appels programmes et les interruptions

Vecteur Reset

pointeur vers l'adresse mémoire du début du programme principal

Vecteurs d'interruption

pointeur vers l'adresse mémoire du programme à exécuter en cas d'interruptions

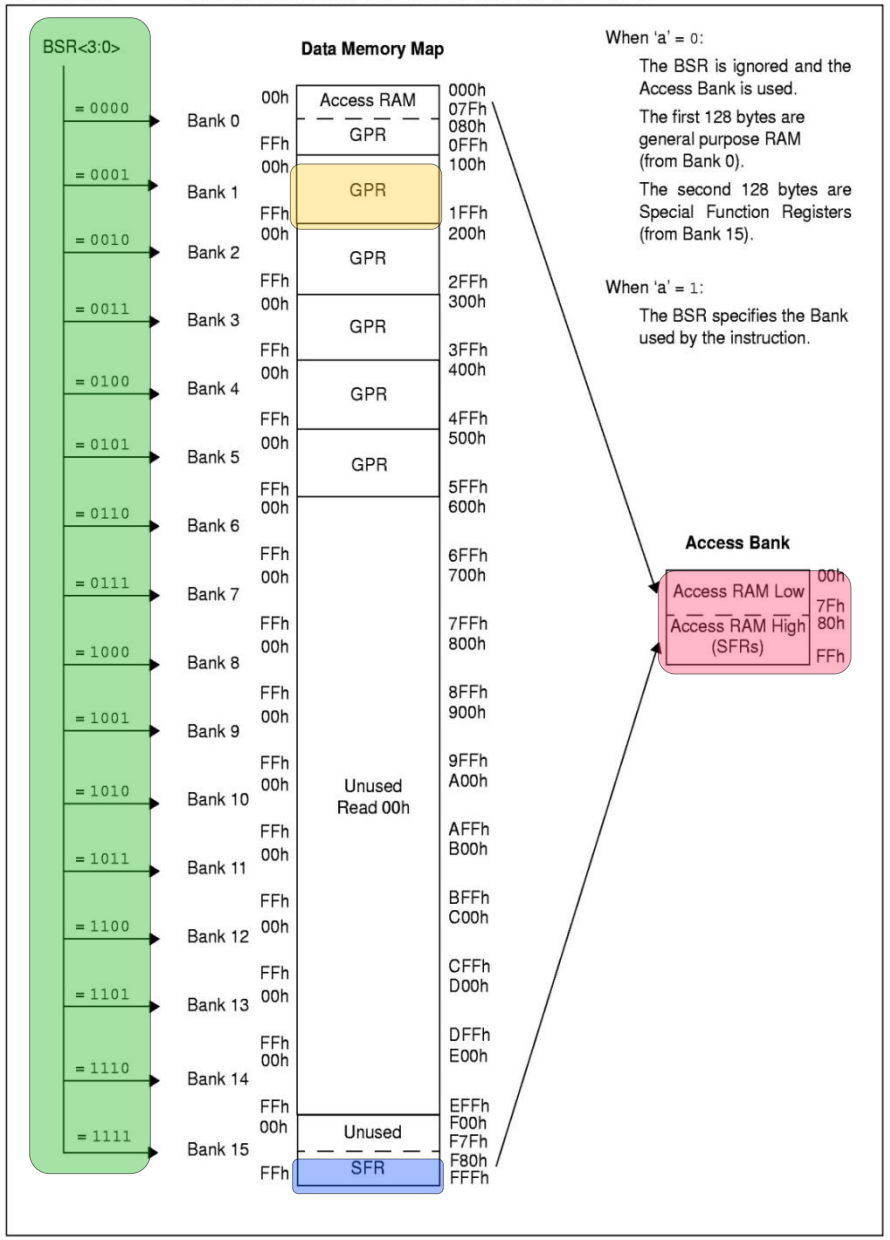
Mémoire programme

zone mémoire réservée au stockage des programmes écrits par l'utilisateur

Remarque : Un **pointeur** est une variable contenant une adresse mémoire.

A vérifier: cohérence avec l'espace mémoire renseigné sur le schéma du PIC.

FIGURE 5-6: DATA MEMORY MAP FOR PIC18F2520/4520 DEVICES



BSR (*Bank Select Register*)

Permet de pré-sélectionner la page pour un accès mémoire plus rapide.
=> notion de **pagination de la mémoire**

GPR (*General Purpose Registers*)

Espaces mémoires qui permet le stockage de données temporaires (variable, ...)

Access Bank

pointeurs vers des zones mémoires

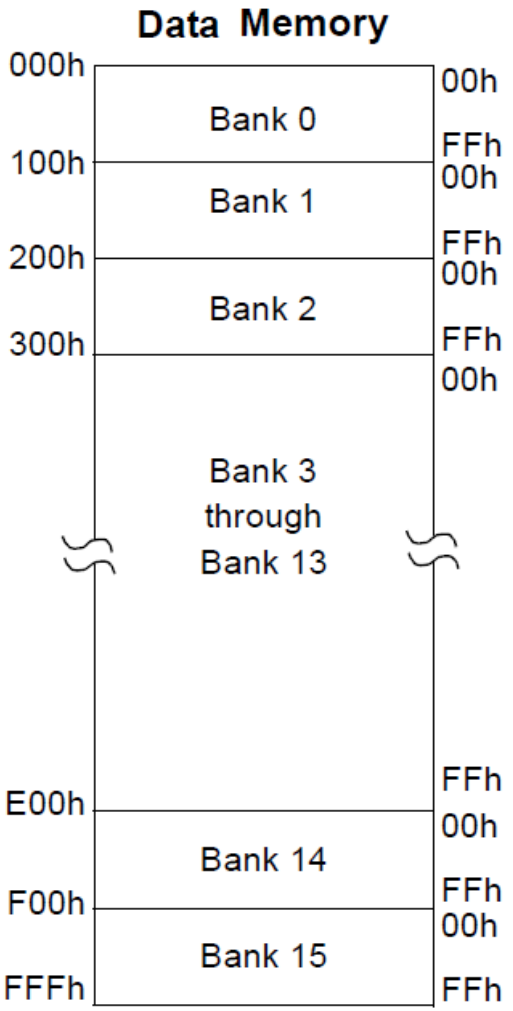
SFR (*Special Function Registers*)

Registres de contrôle et d'état pour les périphériques (WREG, T0CON, …)

« La pagination de la mémoire consiste à diviser la mémoire en pages (Banks) de longueur fixe. » (Source : Comment Ça Marche)

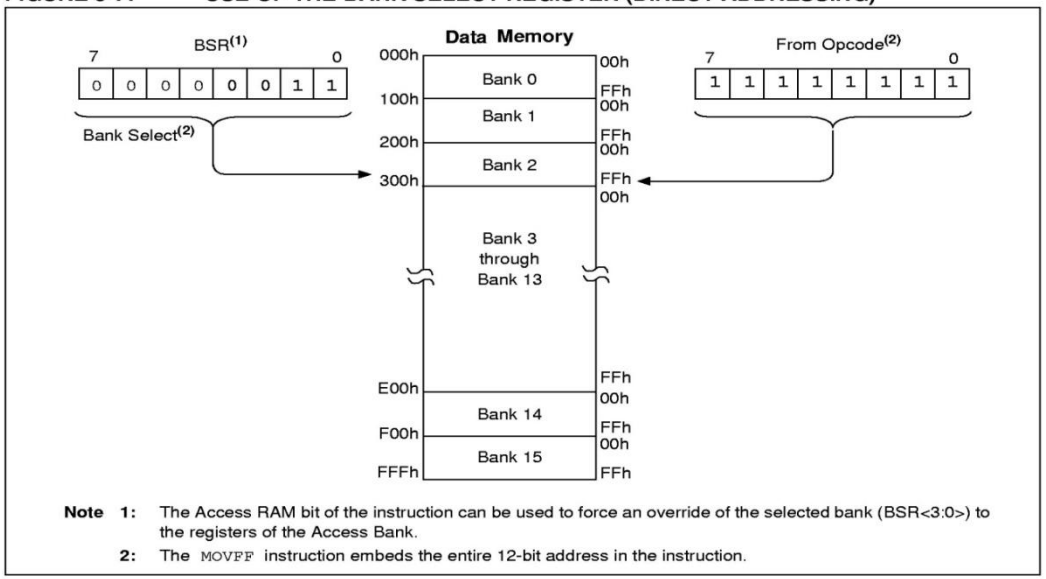
Une adresse mémoire est alors divisée en deux parties :

Partie haute	Partie Basse



Dans le cas d'une instruction avec adressage direct, on transmet seulement la partie basse de l'adresse. Le micro-contrôleur utilise le registre BSR pour compléter l'adresse.

FIGURE 5-7: USE OF THE BANK SELECT REGISTER (DIRECT ADDRESSING)



Note 1: The Access RAM bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

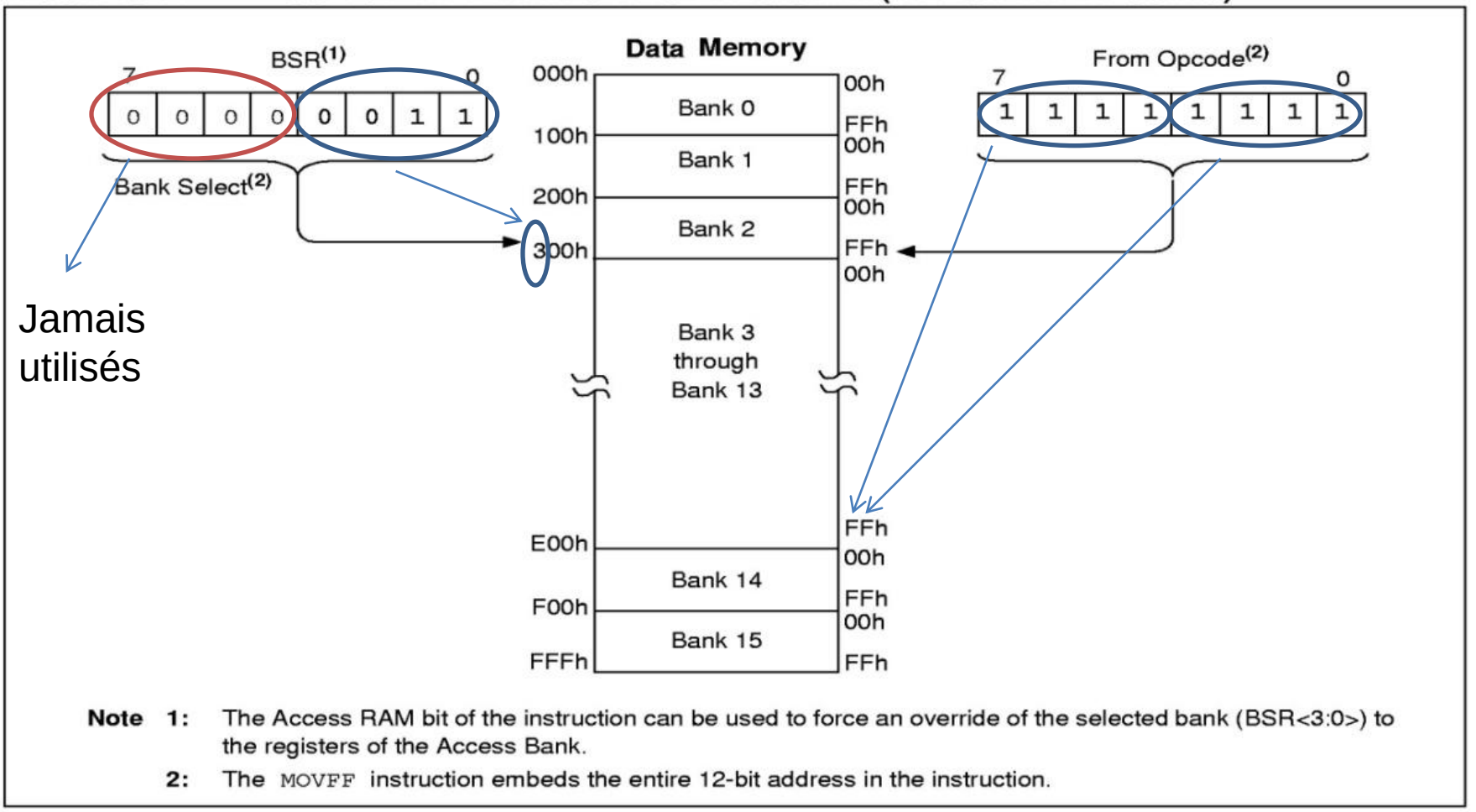
2: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

III Etude du fonctionnement du microcontrôleur

E Organisation de la mémoire

E.4 Accès à la mémoire en adressage direct avec le BSR

FIGURE 5-7: USE OF THE BANK SELECT REGISTER (DIRECT ADDRESSING)



Note 1: The Access RAM bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

2: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

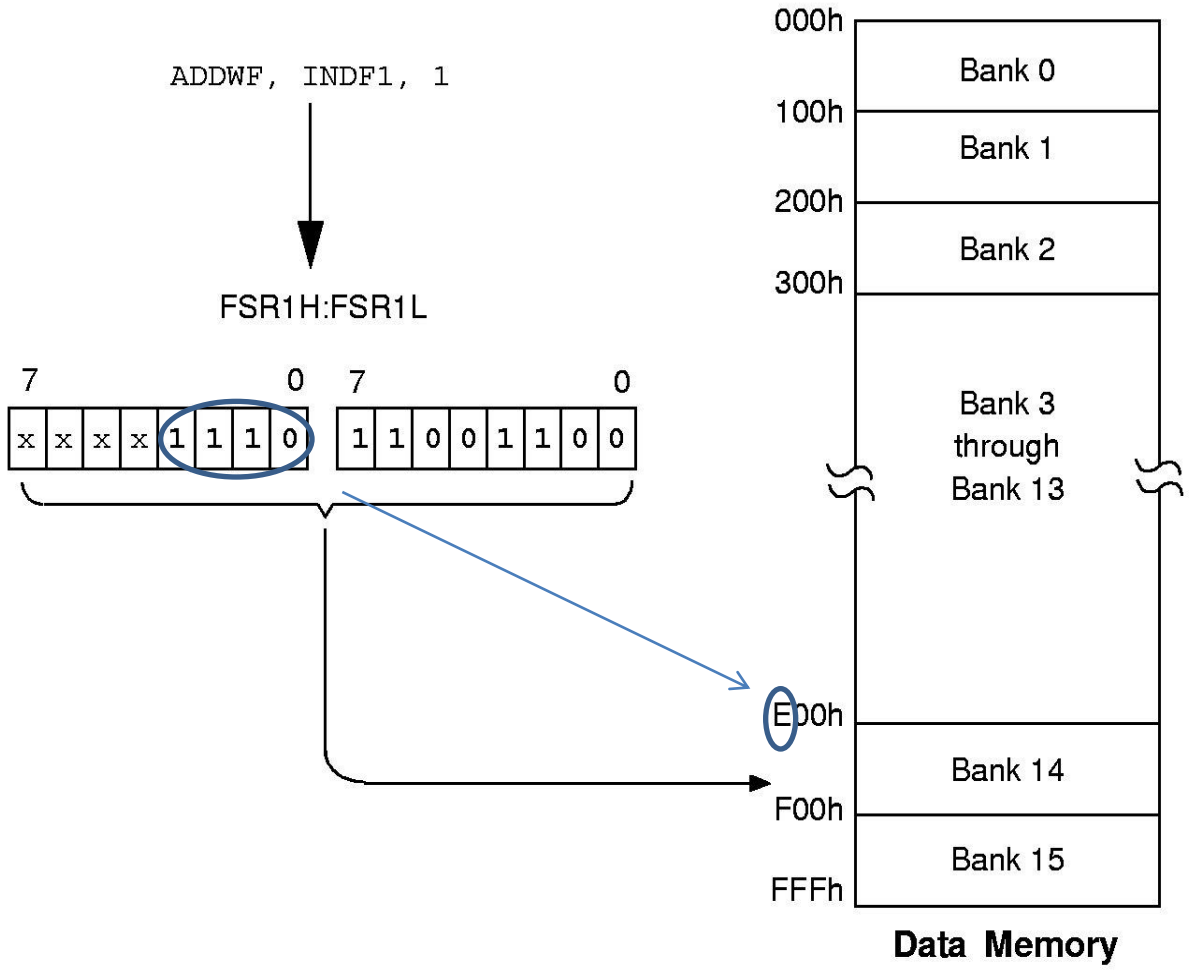
FIGURE 5-8: INDIRECT ADDRESSING

Using an instruction with one of the indirect addressing registers as the operand....

...uses the 12-bit address stored in the FSR pair associated with that register....

...to determine the data memory location to be used in that operation.

In this case, the FSR1 pair contains ECCh. This means the contents of location ECCh will be added to that of the W register and stored back in ECCh.



Plan

Lien avec les TPs

0 Objectifs et plan du cours, lien avec les TPs

I Catégories de systèmes programmables, architecture des microcontrôleurs
- A Présentation de l'informatique industrielle et des systèmes micro-programmés
- B Architecture neuromorphique : comment le cerveau inspire l'informatique industrielle
- C Architecture des microcontrôleurs

II Codage, instructions assembleur, modes d'adressage des instructions
- A Codage binaire et hexadécimal, complément à deux
- B Premier programme assembleur et algorigramme correspondant (TP1)
- C Modes d'adressage des instructions

III Etude du fonctionnement du microcontrôleur
- A Structure des ports entrées sorties
- B Phase de démarrage du microcontrôleur
- C Exécution d'une instruction
- D Les registres
- E Organisation de la mémoire

IV Les interruptions, exemple avec bouton poussoir
- A Interruptions: définition et gestion (TP2)
- B Interruption par appui sur bouton poussoir

PIC: Programmable *Interrupt* Controller

« Une interruption est un arrêt temporaire de l'exécution normale d'un programme informatique par le microprocesseur afin d'exécuter un autre programme (appelé routine d'interruption).

Les interruptions matérielles sont utilisées lorsqu'il est nécessaire de pouvoir réagir en temps réel à un événement asynchrone, ou bien, de manière plus générale, afin d'économiser le temps d'exécution lié à une boucle de consultation (polling loop).» (Source : *Wikipédia*)

Une interruption peut avoir différentes sources : périphérique d'entrée/sortie, *timer*, *watchdog* (*cf.* explications plus loin), ...

Les interruptions sont utilisées pour avertir le micro-contrôleur quand une condition est remplie. En utilisant les interruptions, on évite que le micro-contrôleur reste en attente inutilement (*polling-loop*), elles permettent de gérer les événements asynchrones.

3 bits de contrôle et gestion des interruptions

- **Un bit de Flag**

indique qu'une interruption a été déclenchée et indique la source.

- **Un bit de validation (Enable)**

permet à l'utilisateur d'activer ou non une interruption.

• **Un bit** 'IPEN' de **priorité**

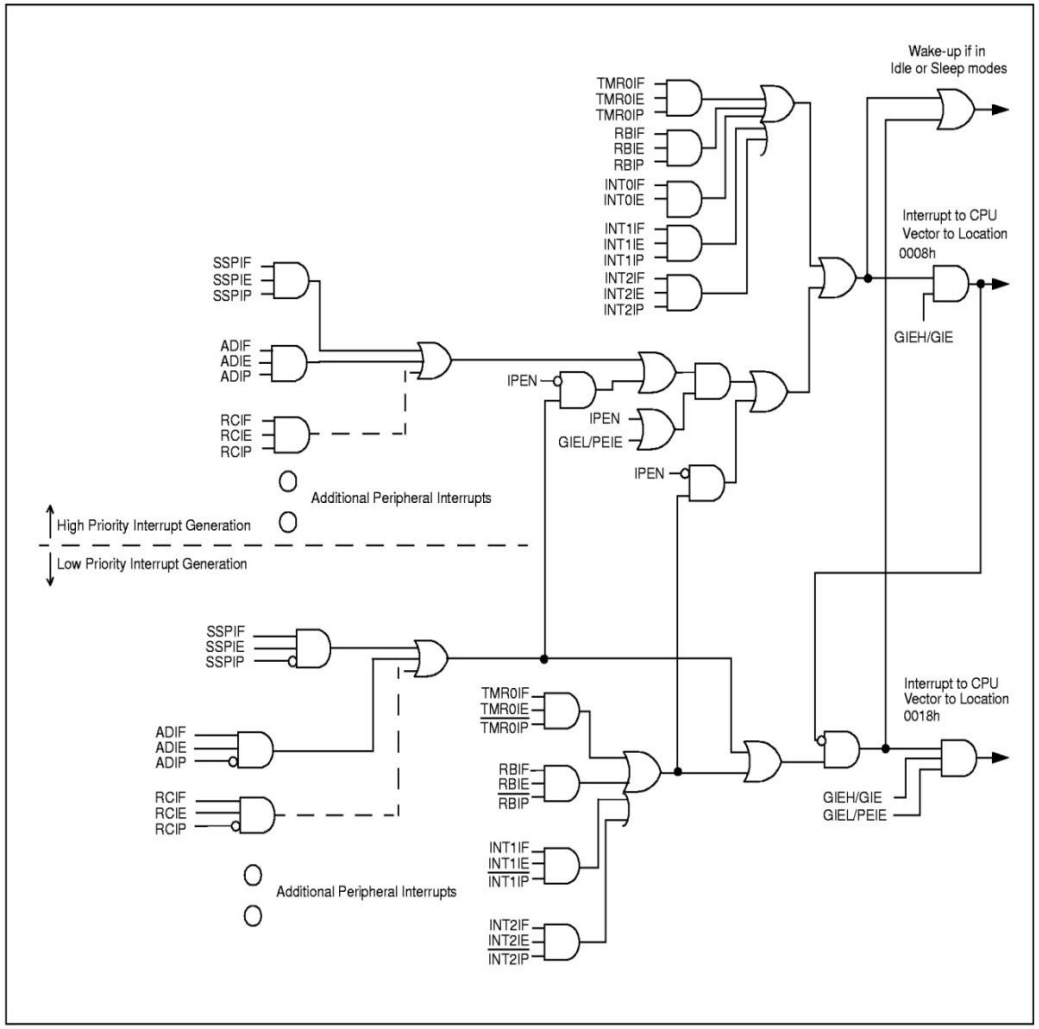
permet de sélectionner la priorité (haute 1/basse 0) de l'interruption. **Il existe des interruptions de priorité hautes et basses**. A chaque type de priorité correspond un **vecteur d'interruption** et une valeur de bit.

IV Les interruptions, exemple avec bouton poussoir

A Interruptions: définition et gestion (TP2)

A.3 Schéma de la logique d'interruption

FIGURE 9-1: PIC18 INTERRUPT LOGIC



A repérer sur le schéma:

TMR0IF (Flag)
TMR0IE (Enable)
IPEN (gestion des priorités)

On notera notamment que si une interruption de haute priorité est en concurrence avec une interruption de basse priorité, l'interruption de haute priorité « prend la main ».

A.4 Déroulement d'une interruption

(1). **Réception de l'interruption :** le micro-contrôleur reçoit une interruption.

(2). **Sauvegarde des données (sauvergarde du contexte) :** le micro-contrôleur sauve une partie variable (en fonction du type d'interruption) de son état interne dans la pile, notamment l'adresse dans la mémoire programme où le micro-contrôleur s'est arrêté.

(3). **Lecture de l'adresse du vecteur d'interruption et chargement dans le PC.**

(4). **Exécution de la routine d'interruption,**

Attention !! L'utilisateur doit penser à effectuer une **sauvegarde de données** du programme principal pour ne pas les effacer pendant la routine d'interruption et également à **supprimer le flag d'interruption** qui a déclenché l'interruption.

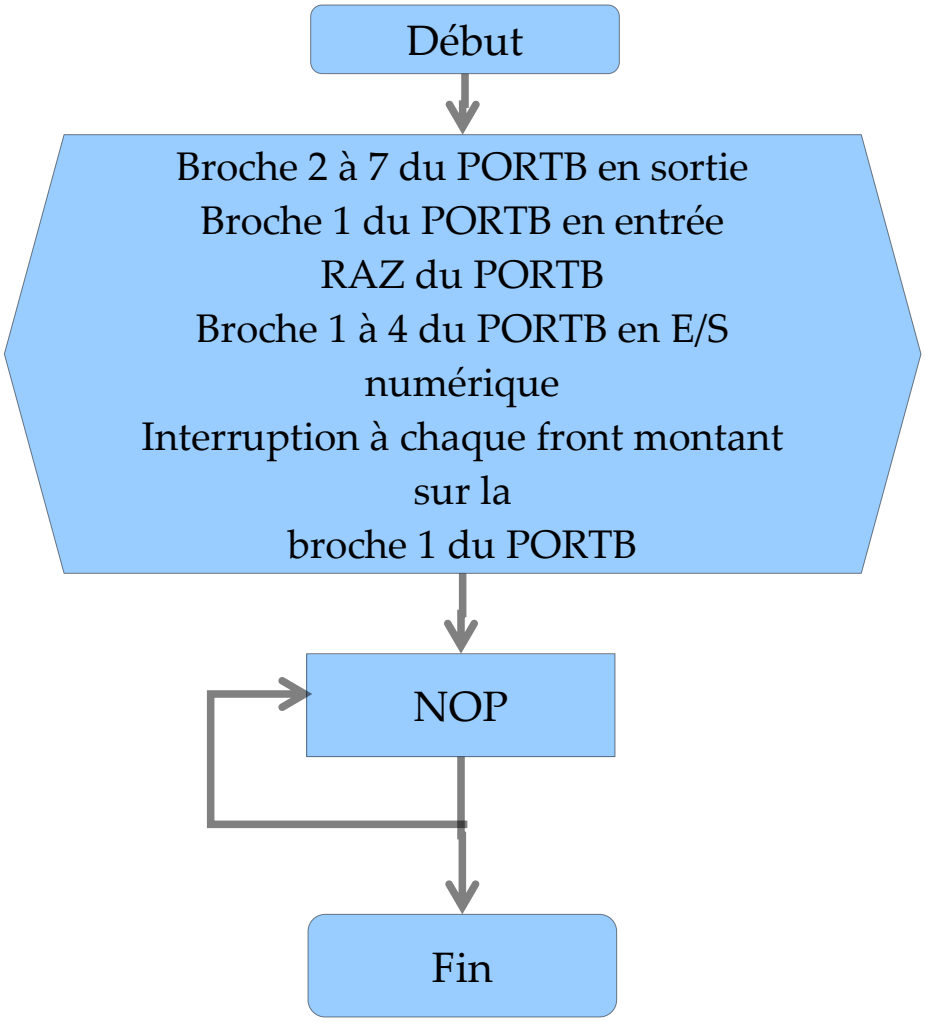
(5). **Rétablissement des données :** le micro-contrôleur rétablit les données stockées dans la pile.

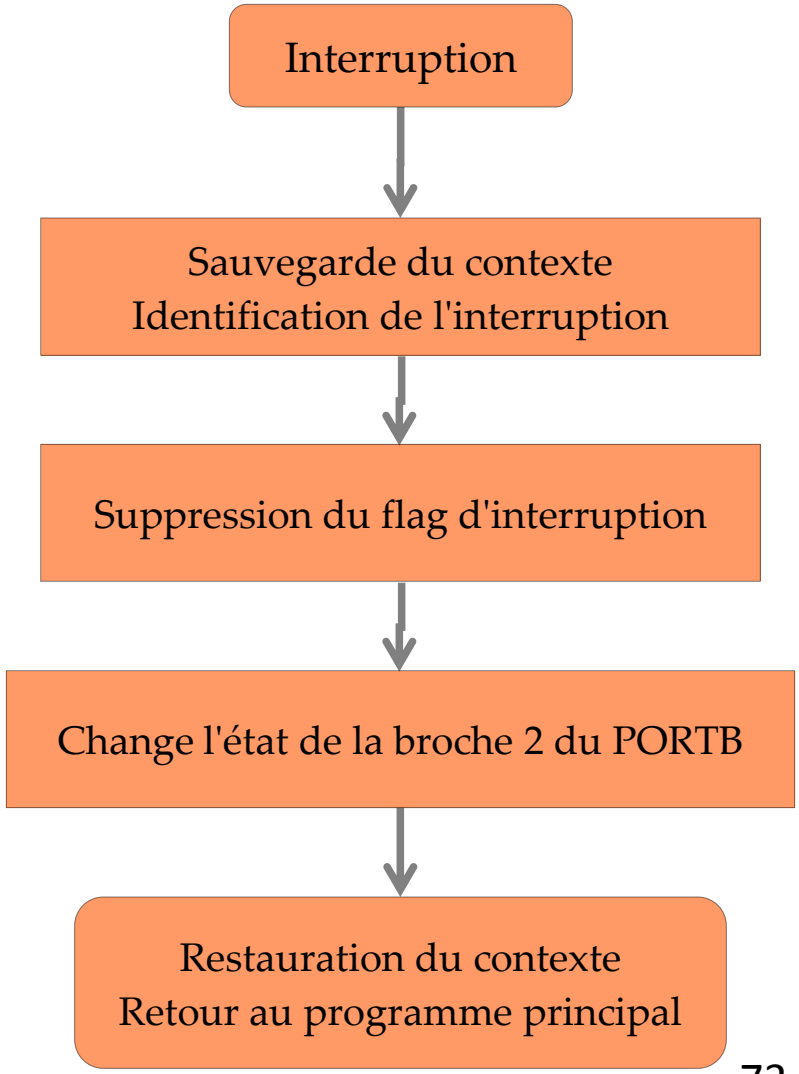
(6). **Le micro-contrôleur reprend son fonctionnement normal...**

IV Les interruptions, exemple avec bouton poussoir

B Interruption par appui sur bouton poussoir

B.1 Algorigrammes: programme principal et programme d'interruption





! Directives de **réservation d'emplacements mémoire** pour la sauvegarde du contexte !

```
; Filename : premier_programme_interruption.asm
; Change l'état de la broche 2 du PORTB à chaque front
; montant sur la broche 1 du PORTB (gestion par interruption

list p=18f4520
; Définition du micro-contrôleur utilisé

#include <p18f4510.inc>
; Définitions des emplacements mémoires des registres
; et configurations matérielles par défaut

#include <MA_CONFIG.inc>
; Modification des configurations matérielles par défaut
```

```
W_TEMP          RES     1; Réservation d'un octet en mémoire
STATUS_TEMP     RES     1; Réservation d'un octet en mémoire
BSR_TEMP        RES     1; Réservation d'un octet en mémoire
```

Programme principal: initialisation du vecteur RESET et du PORT B
Parties propres aux interruptions :
initialisation du **vecteur d'INTERRUPTION**,
initialisation du **registre d'INTERRUPTION** **INTCON**.

```
org             h'0000'         ; Init. du vecteur RESET
                goto    init

                org     h'0008' ; Init. du vecteur INTERRUPTION
                goto    routine_interruption

init            clrf    PORTB
                movlw   b'00000001'
                movwf   TRISB   ; Config. de la dir. du PORTB
                clrf    LATB
                movlw   0Fh
                movwf   ADCON1          ; Broche 1à4 du PORTB en E/S num.
                movlw   b'10010000'     ; 0x90 -> w
                movwf   INTCON  ; w -> INTCON (Init. du reg. d'interrup.)

boucle nop
                goto    boucle
                END
```

Le registre d'interruption INTCON permet, *d'une part* d'activer les interruptions (bit 7), et *d'autre part* d'activer le mode interruption externes INT0 (bit 4). Dans ce cas, l'interruption sera détectée sur la broche 0 du port B (*cf. datasheet*).

REGISTER 9-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 GIE/GIEH: Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 PEIE/GIEL: Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 TMR0IE: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 INT0IE: INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 RBIE: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 TMR0IF: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 INT0IF: INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

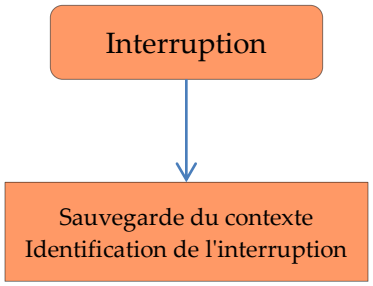
bit 0 RBIF: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Le déclenchement d'une interruption conduit le microcontrôleur à sauver l'adresse de l'instruction courante dans la pile, puis à charger le vecteur d'interruption dans le PC.

Dès lors, il est **systématiquement** nécessaire de (1) **sauvegarder le contexte** et (2) **identifier l'origine de l'interruption**.

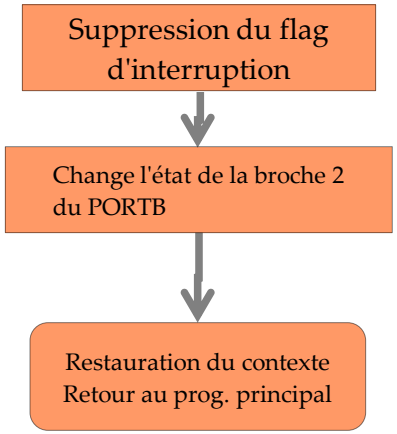



```
routine_interruption

; Sauvegarde du contexte
  movwf   W_TEMP                  ;Sauvegarde de W
  movff   STATUS, STATUS_TEMP     ;Sauvegarde de STATUS
  movff   BSR, BSR_TEMP           ;Sauvegarde de BSR

; identification de l'origine de l'interruption
  btfsc   INTCON,1
  goto    interruption_INT0
  bra             restauration_contexte
```

Il faut ensuite **systématiquement** (3) mettre à **zéro le bit d'interruption** puis, (4) **exécuter la fonction** pour laquelle l'interruption a été prévue, et enfin (4) faire la **restauration du contexte** (5) et retourner au programme principal.




```
interruption_INT0

        bcf     INTCON,1        ; Suppression du flag d'interruption

        movlw   0x02            ; 0x02 -> w
        xorwf   PORTB           ; w xor PORTB -> PORTB
        goto    restauration_contexte

;       Restauration du contexte
restauration_contexte
        movff   BSR_TEMP, BSR           ; Restauration de BSR
        movff   W_TEMP, W               ; Restauration de W
        movff   STATUS_TEMP, STATUS     ; Restauration de STATUS

        retfie
```

That’s all folks