Remise à niveau

Informatique industrielle

Micro-contrôleurs Microchip

Julien Marot julien.marot@fresnel.fr

Equipe Groupe Signaux Multidimensionnels
Institut Fresnel, bureau 237
Saint-Jérôme

Organisation de l'enseignement

Contenu horaire :

- **10 h de cours/TD**

Présentation de l'informatique industrielle, Programmation en langage Assembleur, interruptions, génération d'un signal PWM

- **3 h de travaux pratiques**

Mise en pratique: programme assembleur pour la génération d'un signal PWM sur carte microchip et PIC 18F4520.

Merci d'être à l'heure en cours / TP !

Plan

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 des microcontrôleurs

II Codage, instructions assembleur, algorigrammes
- A Codage binaire et hexadécimal, opérations arithmétiques
- B Premier algorigramme et instructions associées

III Les interruptions, exemple du bouton poussoir
- A Interruptions: définition et gestion
- B Interruption par appui sur bouton poussoir

IV Modules fonctionnels
- A Capture, Compare, PWM
- B Conversion analogique numérique et Watchdog

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 en œuvre une interruption externe*
- *Créer un signal PWM avec un microcontrôleur*

Evaluation

TP: 100%

Plan

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 des microcontrôleurs

II Codage, instructions assembleur, algorigrammes

- A Codage binaire et hexadécimal, opérations arithmétiques
- B Premier algorigramme et instructions associées

III Les interruptions, exemple du bouton poussoir

- A Interruptions: définition et gestion
- B Interruption par appui sur bouton poussoir

IV Modules fonctionnels

- A Capture, Compare, PWM
- B Conversion analogique numérique et Watchdog

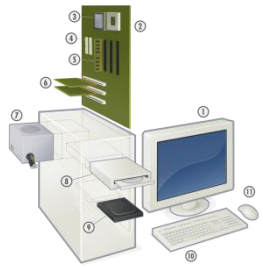
L'informatique industrielle

« 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)

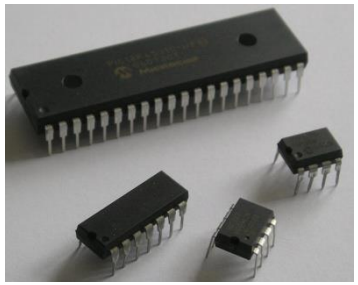
Source : Ascom S.A.

Les systèmes micro-programmés: hiérarchie

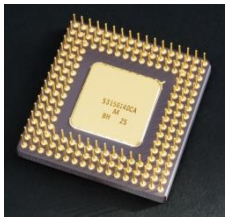
- **Micro-ordinateur:** fixe, portable, single board (Raspberry Pi, banana Pi)



- **Microcontrôleur:** circuit intégré qui rassemble un processeur, des mémoires, des ports entrée sortie, des unités périphériques comme une horloge. *Exemple:* le PIC



- **Processeur (CPU en anglais):** exécute les instructions machine. Un processeur construit en un seul circuit intégré est un microprocesseur. *Exemple*: Intel 80486



Deux types de processeurs

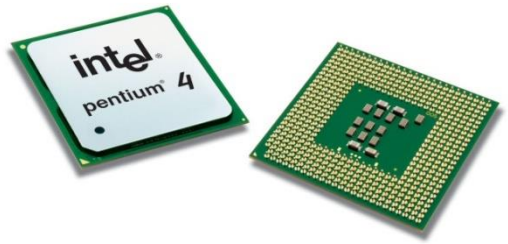
- **CISC** : *Complex Instruction Set Computer*

  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

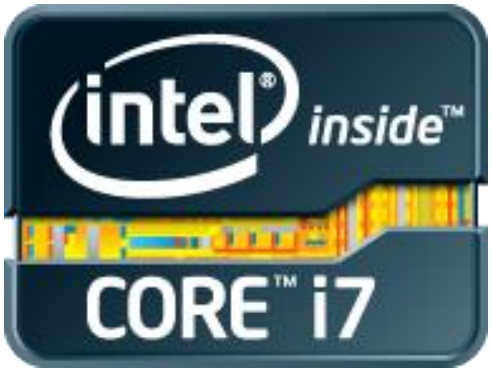
Intel Pentium 4 Northwood C (2002)
42 millions de transistors, gravés en 0,13 µm
architecture interne 32 bits
fréquence d'horloge 2,4/3,4 Ghz
(bus processeur : 200Mhz)
450 MIPS

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

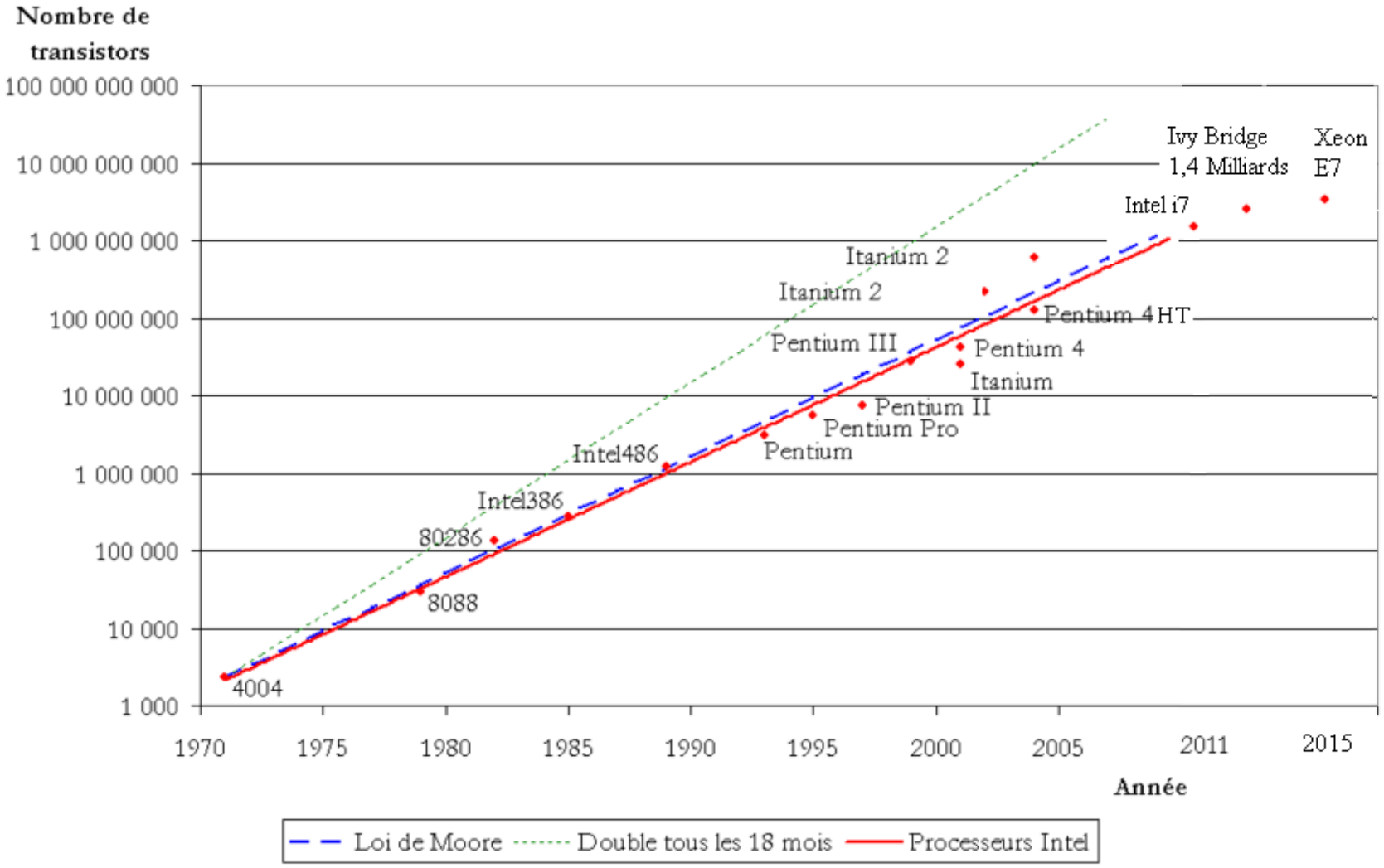
Intel Pentium 4 Northwood C (2002)
42 millions de transistors, gravés en 0,13 µm
architecture interne 32 bits
fréquence d'horloge 2,4/3,4 Ghz
Fréquence de bus: 0,2 GHz
450 MIPS

« 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

- **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 :

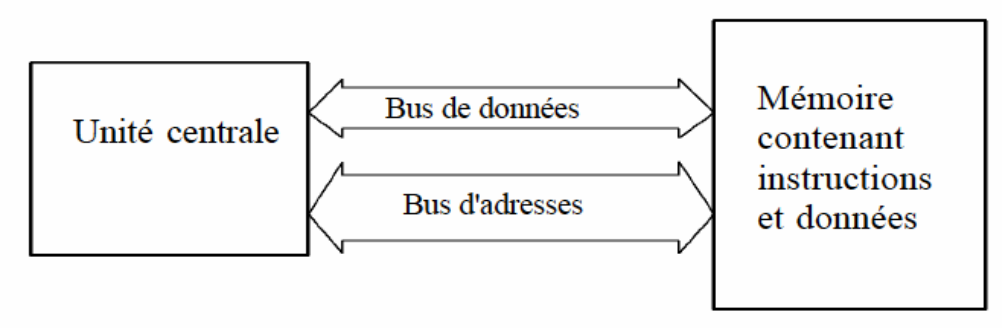
- **Bus de données** : permet de transférer entre composants des données,
  ex. : *résultat d'une opération, valeur d'une variable, etc.*

- **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.*

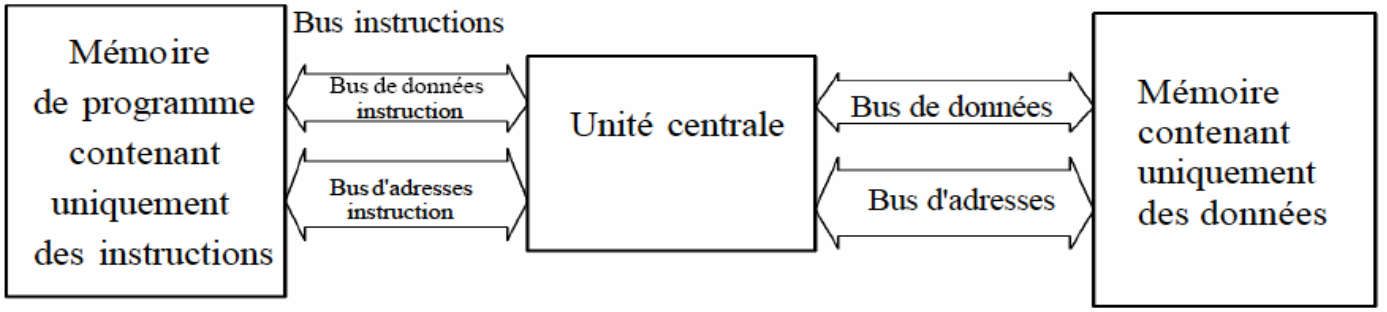
Définition : Une **adresse** est un nombre binaire qui indique un emplacement dans une zone mémoire

- **Structure de Von Neumann**



Extraits du cours intitulé « Les systèmes micro-programmés »

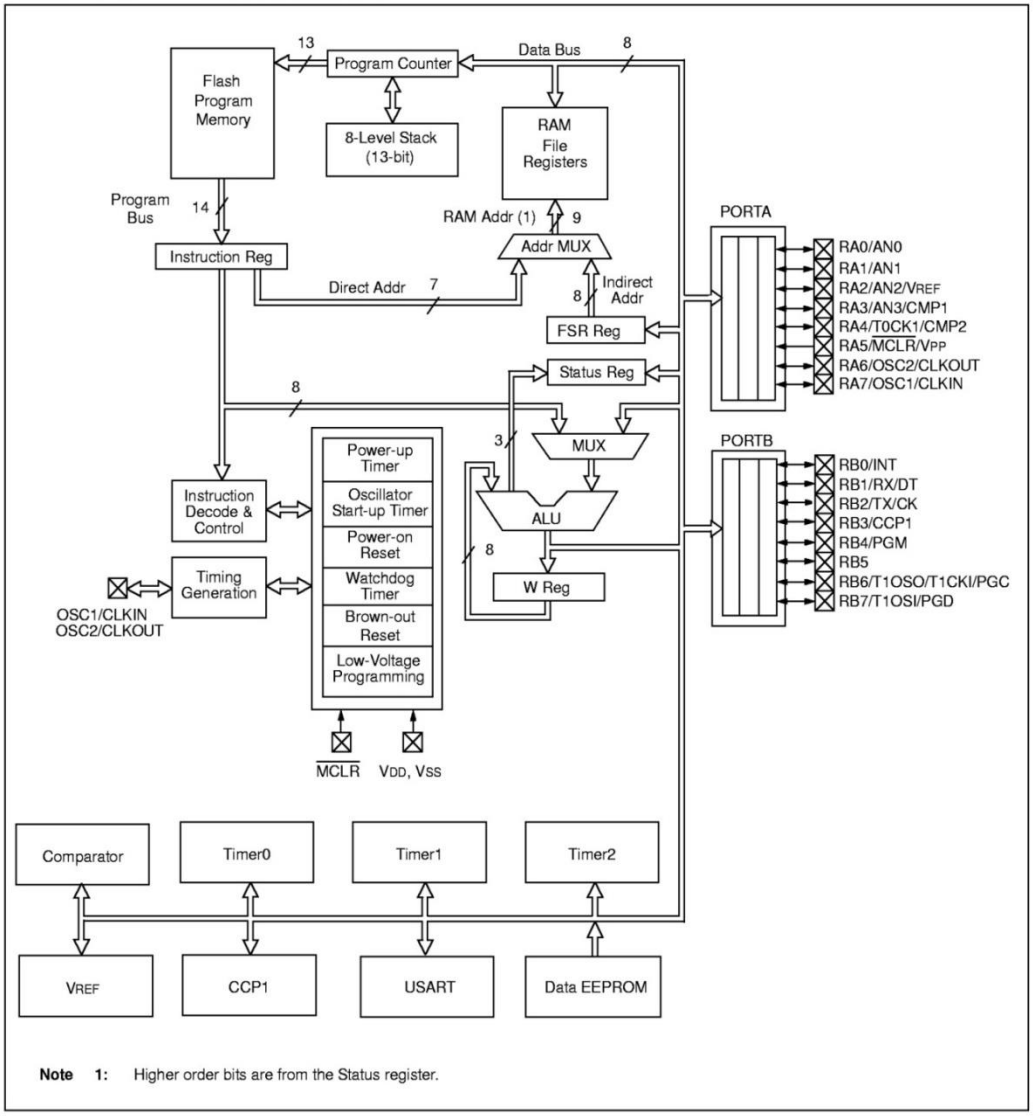
- **Structure de Harvard**



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.

B.2 Schéma bloc d'un microcontrôleur

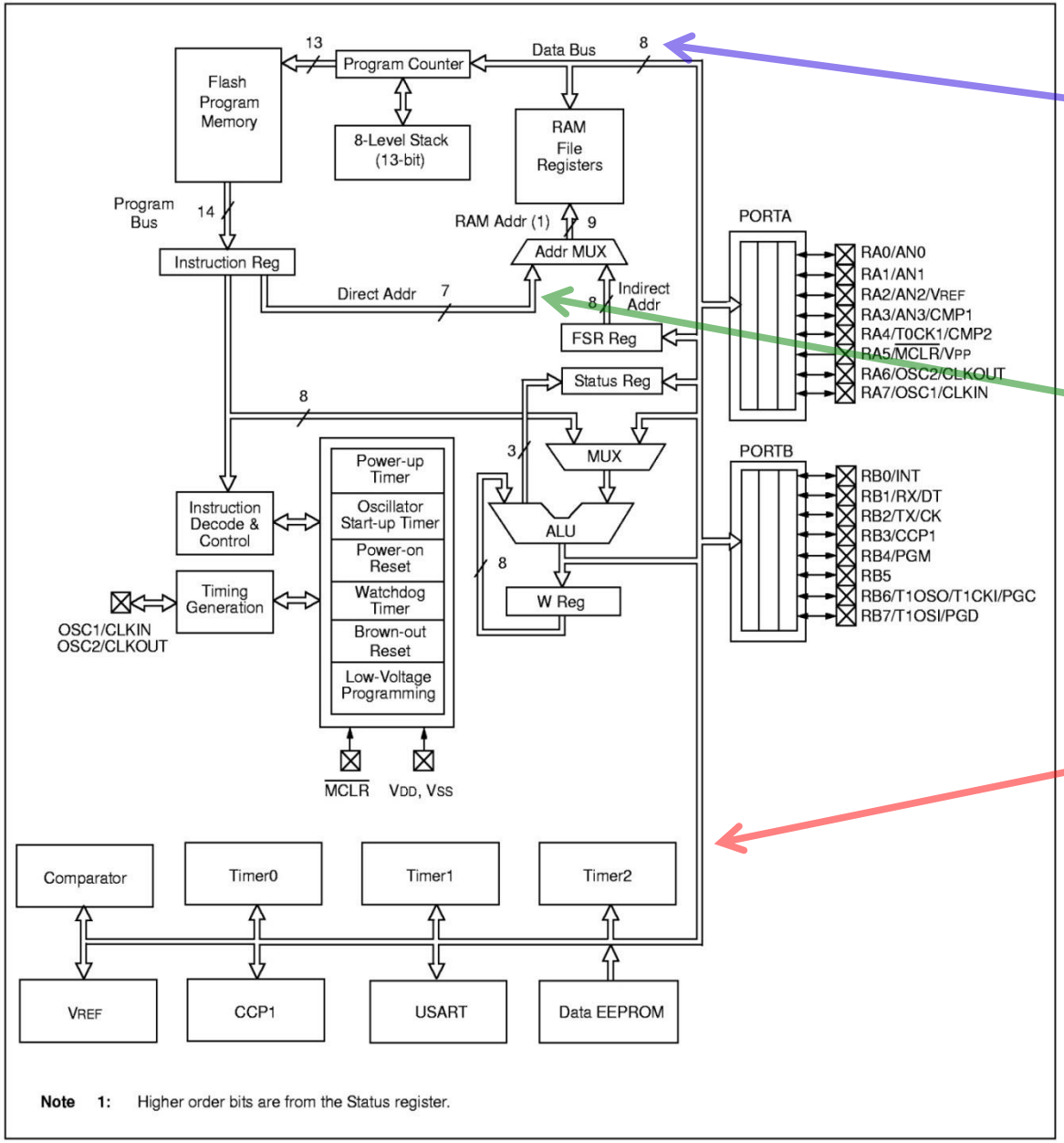
FIGURE 3-1: BLOCK DIAGRAM



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 »**

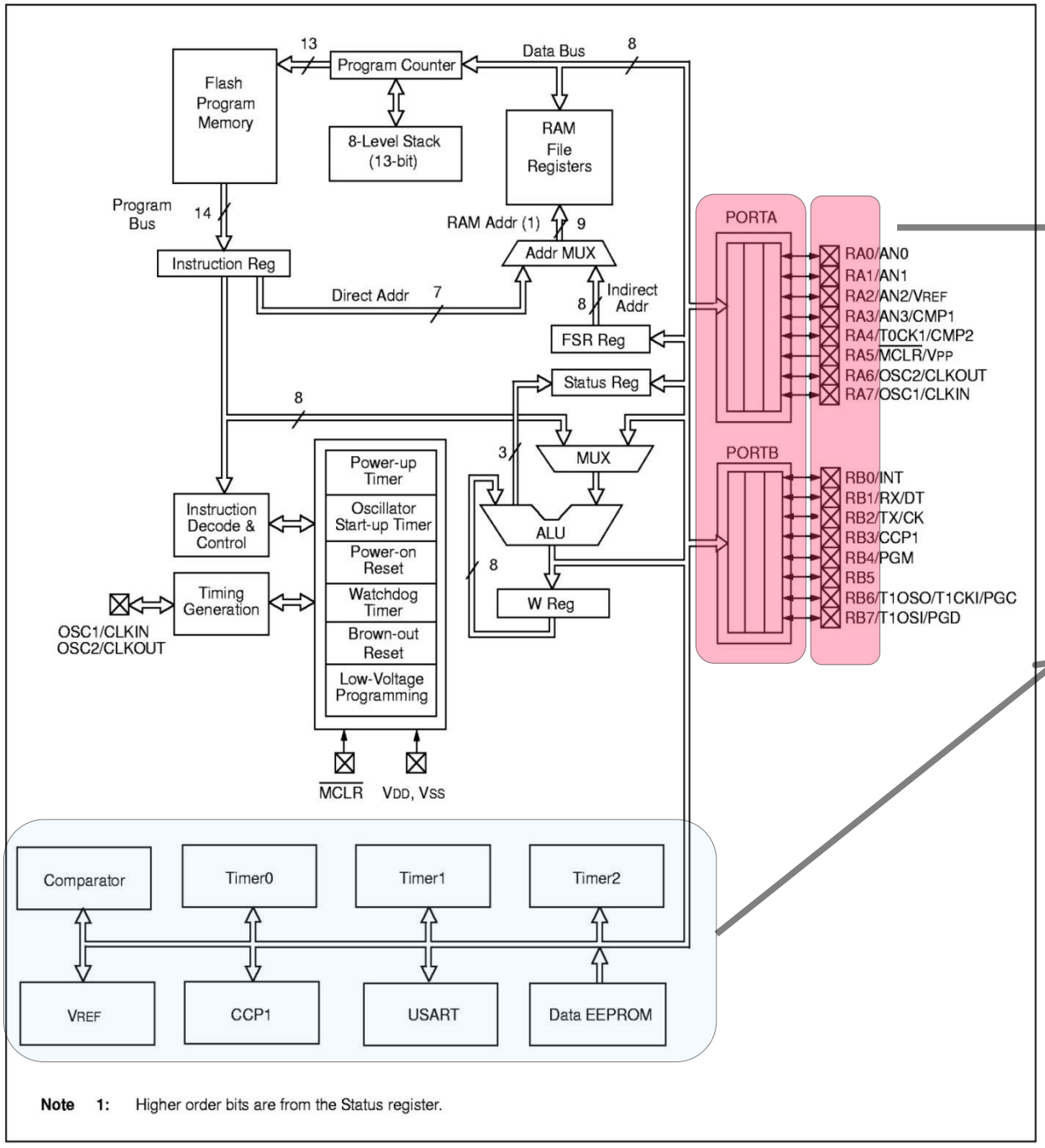
8

- **Unidirectionnel**

- **Bidirectionnel**

Issu de la documentation technique du PIC16F628

FIGURE 3-1: BLOCK DIAGRAM



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

- **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, ...**

Plan

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 des microcontrôleurs

II Codage, instructions assembleur, algorigrammes
- A Codage binaire et hexadécimal, opérations arithmétiques
- B Premier algorigramme et instructions associées

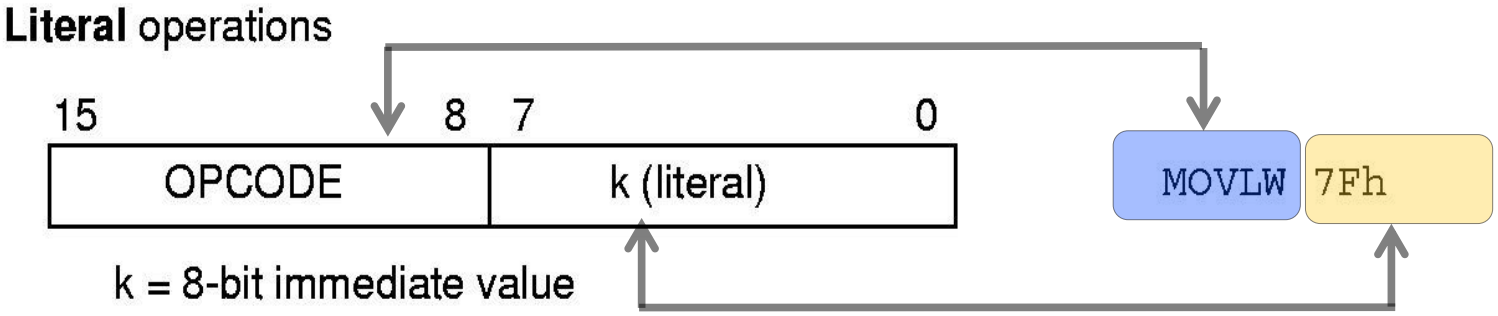
III Les interruptions, exemple du bouton poussoir
- A Interruptions: définition et gestion
- B Interruption par appui sur bouton poussoir

IV Modules fonctionnels
- A Capture, Compare, PWM
- B Conversion analogique numérique et Watchdog

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 microcontrôleur :

- Les instructions logiques : andlw, xorwf, ...
- 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 préprocesseur

Elles permettent au programmeur de donner des indications au compilateur.

Elles sont destinées au PC et non pas au microcontrôleur !

- 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

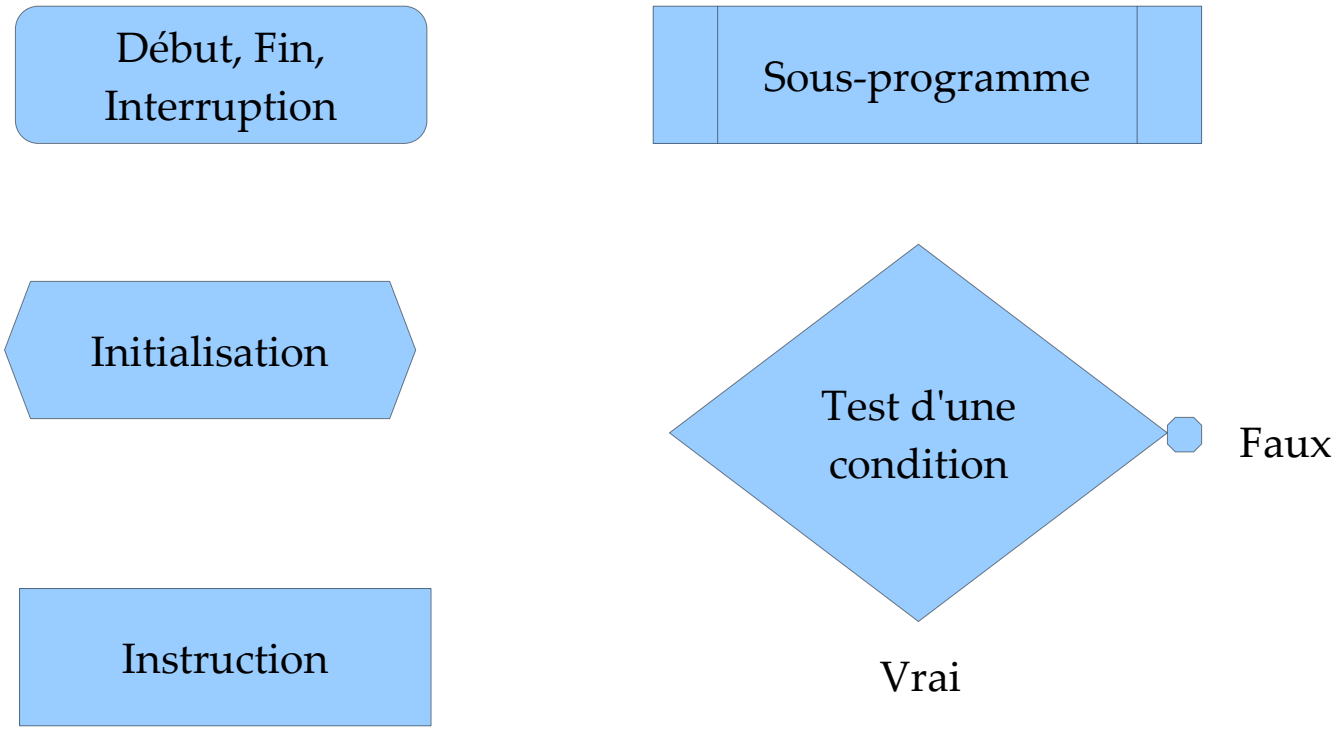
xorwf REG
ou exclusif entre le registre ‘w’ et le registre ‘REG’
Le résultat est placé dans ‘REG’

OU-EXCLUSIF et inversion de valeur bit

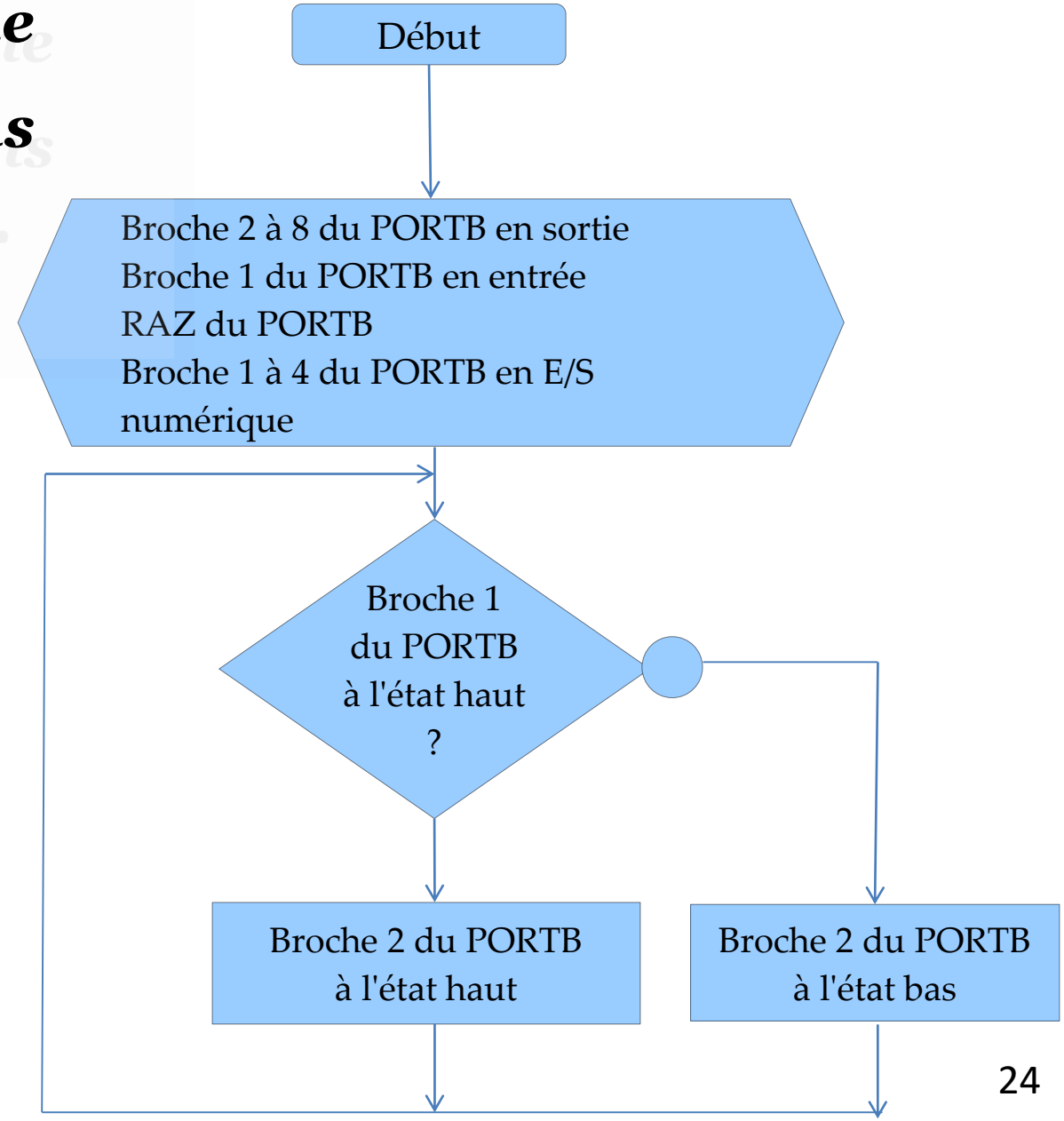
A	B	XOR
0	0	0
1	0	1
0	1	1
1	1	0

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 instructions assembleur associées

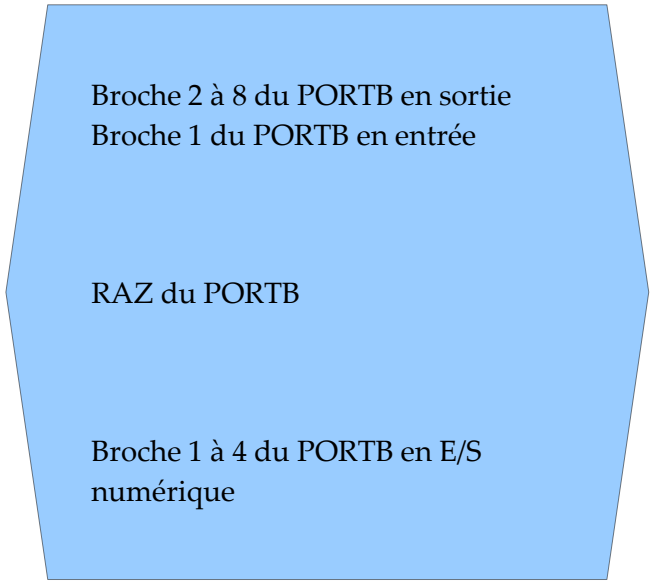


La première opération consiste systématiquement à **initialiser le vecteur RESET**.

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
	

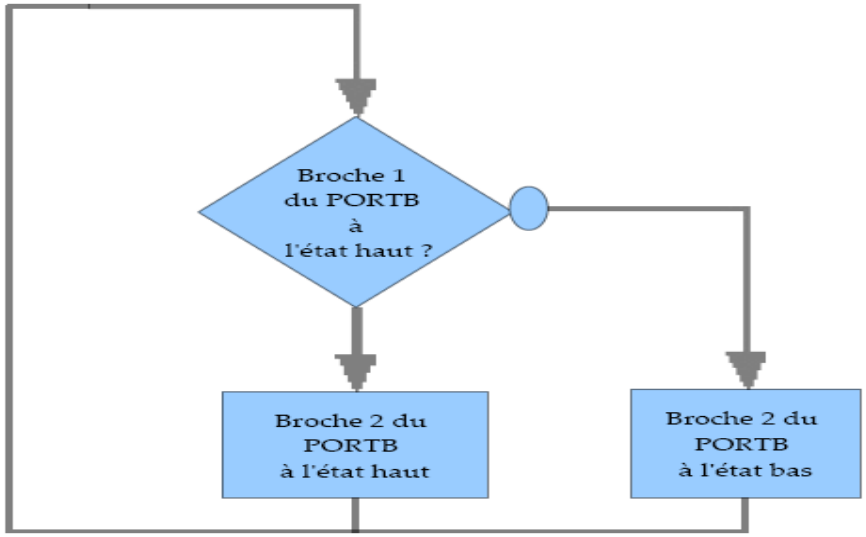

```
init    movlw   b'00000001'
        movwf   TRISB

        clrf    PORTB

        movlw   0Fh
        movwf   ADCON1
```

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

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
	``

Plan

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 des microcontrôleurs

II Codage, instructions assembleur, algorigrammes
- A Codage binaire et hexadécimal, opérations arithmétiques
- B Premier algorigramme et instructions associées

III Les interruptions, exemple du bouton poussoir
- A Interruptions: définition et gestion
- B Interruption par appui sur bouton poussoir

IV Modules fonctionnels
- A Capture, Compare, PWM
- B Conversion analogique numérique et Watchdog

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.

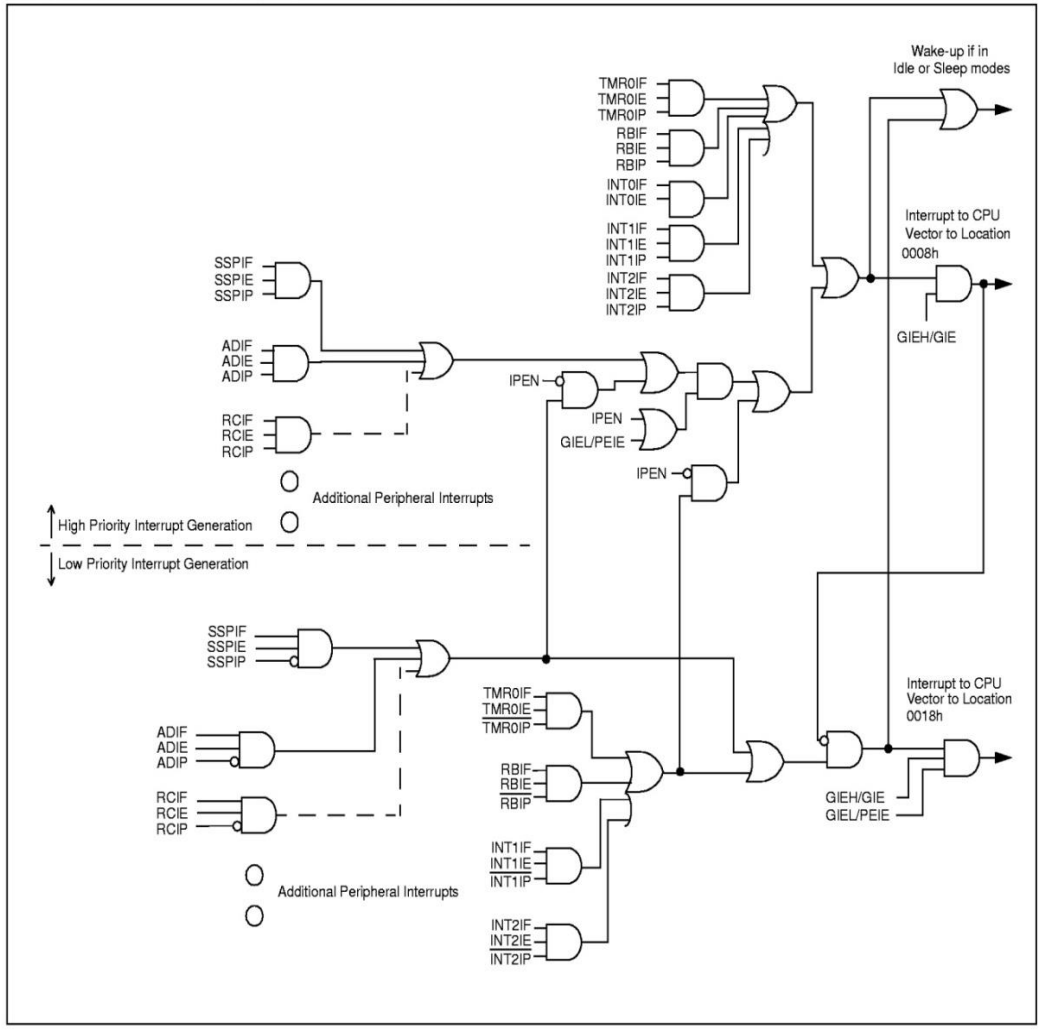
- **Un bit de validation (Enable)**

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.

FIGURE 9-1: PIC18 INTERRUPT LOGIC



A repérer sur le schéma:

INT0IF (Flag)
INT0IE (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.

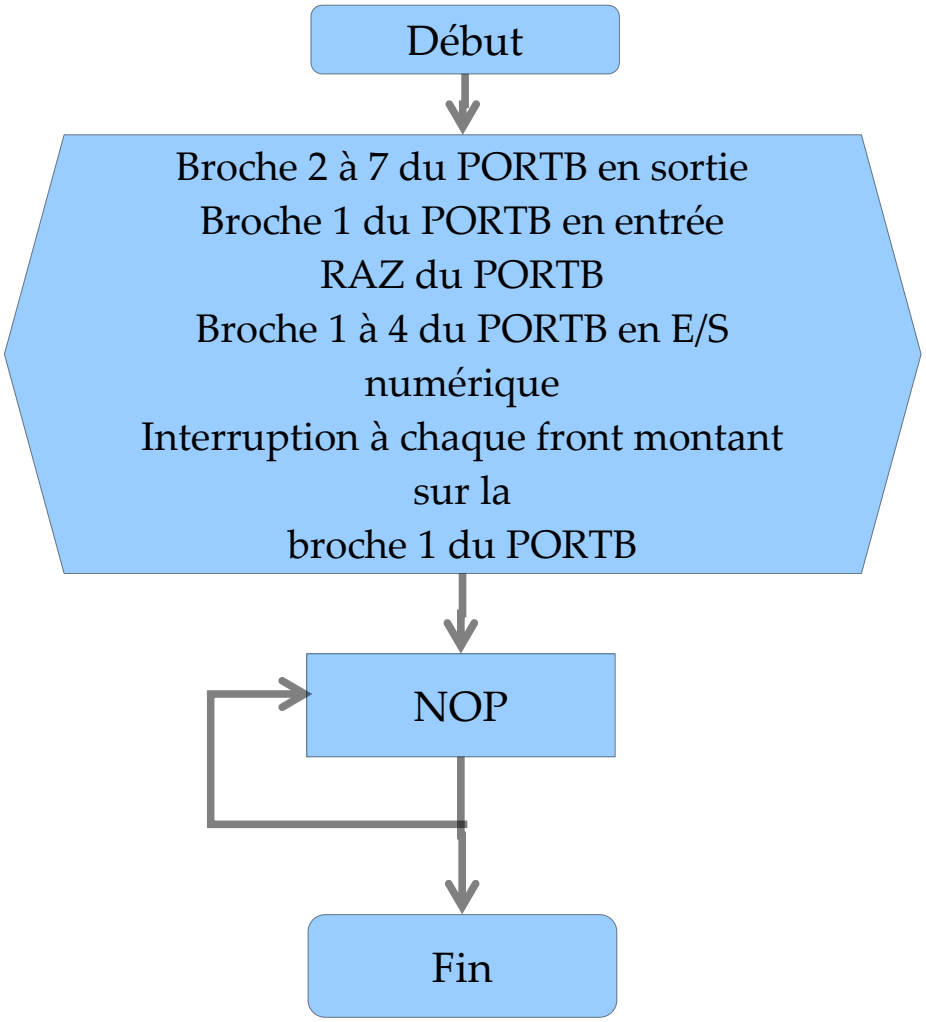
(5). **Rétablissement des données :** le micro-contrôleur rétablit les données stockées dans la pile.

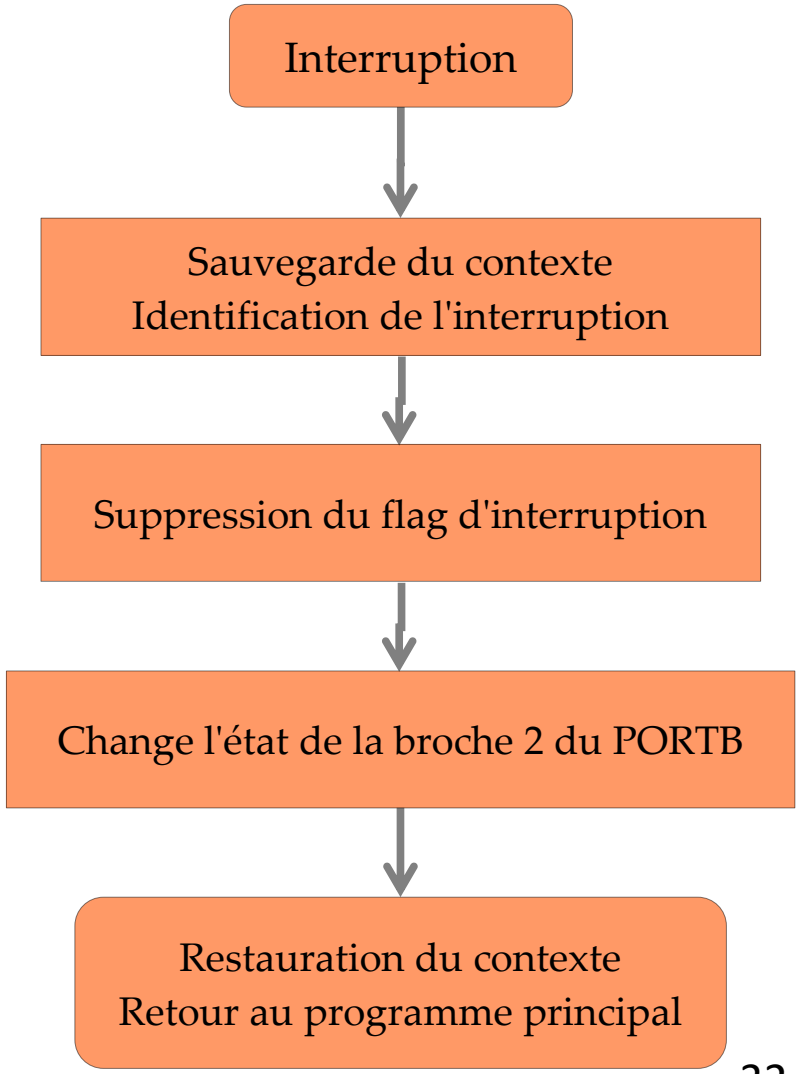
(6). **Le micro-contrôleur reprend son fonctionnement normal...**

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
```

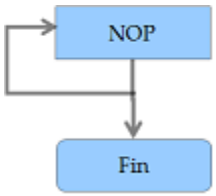
Début

```
org     h'0008' ; Init. du vecteur d'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 registre d'INTERRUPTION
```

Initialisation: Broche 2 à 7 du PORTB en sortie…

```
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

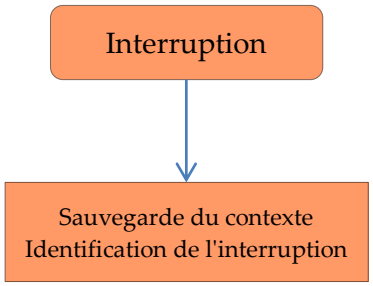
Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

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**
(2) **identifier l'origine de l'interruption**.

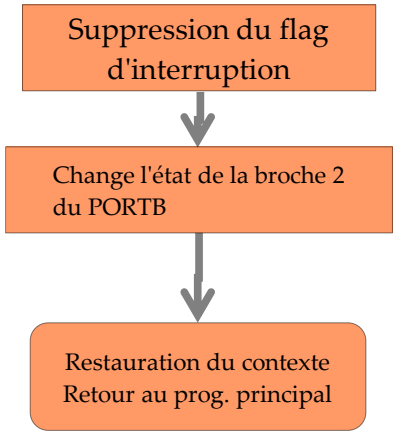
```
routine_interruption
```




```
; 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
```

Plan

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 des microcontrôleurs

II Codage, instructions assembleur, algorigrammes
- A Codage binaire et hexadécimal, opérations arithmétiques
- B Premier algorigramme et instructions associées

III Les interruptions, exemple du bouton poussoir
- A Interruptions: définition et gestion
- B Interruption par appui sur bouton poussoir

IV Modules fonctionnels
- A Capture, Compare, PWM
- B Conversion analogique numérique et Watchdog

Les modules Capture, Compare, PWM (CCP) possèdent trois modes de fonctionnement :

.capture

Le mode *capture* déclenche une action si un événement pré-déterminé apparaît (ex : changement d'état sur une broche). Utilisé avec les timers, ce module peut compter les temps d'arrivées.

.compare

Le mode *compare* effectue une comparaison permanente entre le contenu d'un timer et une valeur donnée pour déclencher une action si ces contenus sont égaux.

.Pulse width modulation (PWM)

Le mode PWM génère un signal rectangulaire de fréquence et de rapport cyclique choisis par l'utilisateur.

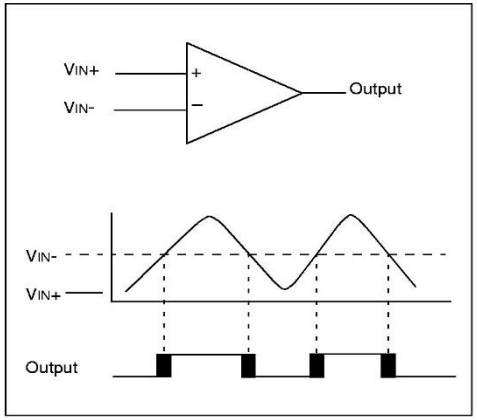
Les comparateurs

Les comparateurs permettent de comparer le signal analogique présent sur un broche du micro-contrôleur à une *valeur de référence*.

Cette valeur de référence peut être

- **soit un signal *analogique*** dont on fait l'acquisition sur une autre broche du micro-contrôleur (*convertisseur analogique - numérique*),
- **soit une *tension de référence*** générée en interne par le micro-contrôleur à l'aide du module de génération de tension de référence.

FIGURE 20-2: SINGLE COMPARATOR



Ce principe de fonctionnement décrit ci-contre permet typiquement d'effectuer une commande de type tout-ou-rien (TOR).

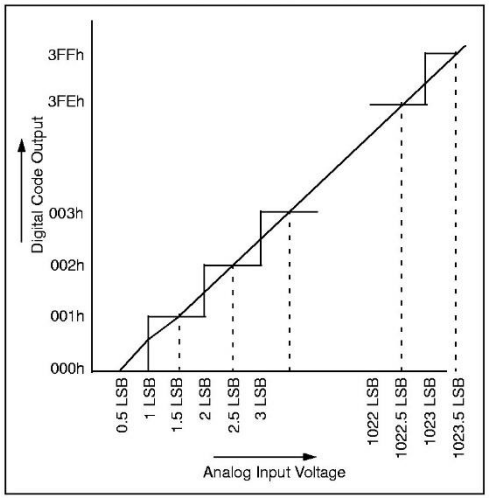
Module de conversion analogique/numérique

Le module de conversion analogique/numérique permet de convertir le signal analogique présent sur une broche du micro-contrôleur en un signal numérique.

Les paramètres à prendre en compte pour la numérisation d'un signal sont

- **la pleine échelle** du module de conversion A/N
  Indique la plage de tension admissible en entrée du module.
- **la dynamique**
  Indique le nombre de bits utilisée pour coder une valeur analogique en numérique.
- **la fréquence d'échantillonnage**

FIGURE 19-2: A/D TRANSFER FUNCTION



Notes : (1) la pleine échelle et la dynamique permettent de calculer la *résolution en tension* du module de conversion A/N ; (2) la fréquence d'échantillonnage doit respecter le (fameux) « *théorème de Shannon* » ; cf cours de traitement du signal.

Le « chien de garde » (Watchdog)

Une **watchdog** (WDT) est un dispositif de protection pour éviter que le micro-contrôleur ne se bloque. Une watchdog effectue un redémarrage du système (RESET) si une action définie n'est pas effectuée dans un délai donné.

Concrètement, l'utilisateur affecte une valeur à un registre (*Watchdog Postscaler*), qui définit une durée temporelle (**timeout**). Périodiquement le micro-contrôleur va incrémenter un registre (*Watchdog counter*). Si ce registre est plein (*overflow*), le micro-contrôleur effectue un re-démarrage.

Pour que le micro-contrôleur ne redémarre pas, le programme doit périodiquement ré-initialisé le registre (Watchdog counter).

That’s all folks