Microcontrôleurs

Famille PIC 16

Thomas Grenier,
Dominique Tournier.

Insa-GE, DUT+3

Objectifs de ce cours

1 – Présenter la structure générale des ordinateurs et les concepts associés

2 – Donner les potentialités permettant d'analyser et/ou de concevoir des éléments matériels (cartes) ou logiciels (programmes) de systèmes bâtis autour des PIC

3 – Définir les éléments génériques des systèmes informatiques permettant une compréhension très rapide d'un nouveau système indépendamment du microcontrôleur dont il est issu

Organisation de la formation

- En DUT+3 : 15h
  - Cours (7h)
  - TD (8h)
  - Examen (devoir de 2h)
    - Thème: matériel et/ou logiciel
    - Documents autorisés

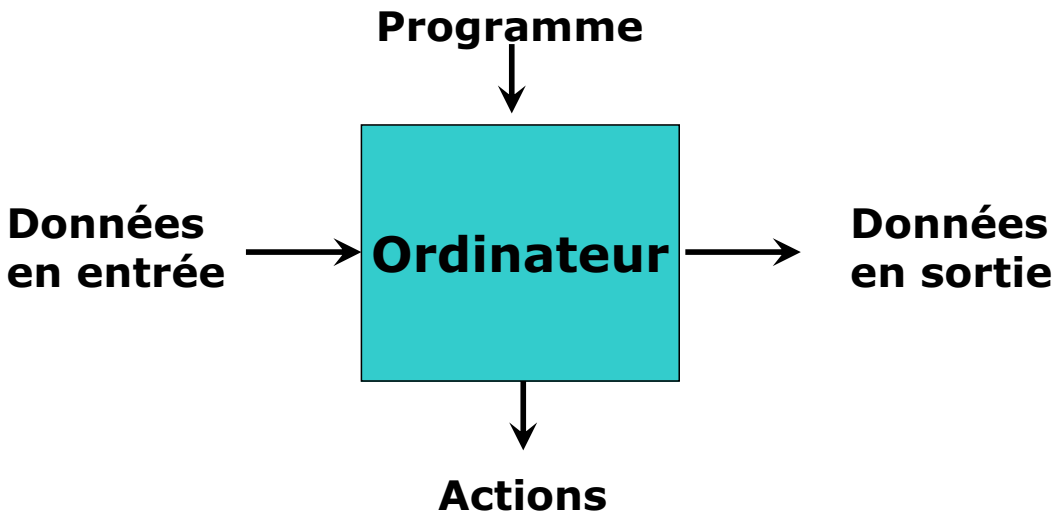
- En 4GE (2ième semestre)
  - Un TP sur les interruptions (4h)
  - Question TP (dans l'examen du module IF3)
  - Projets…

Plan du cours

I. Généralités
   - Structure élémentaire d'un calculateur
II. Les microcontrôleurs PIC16
III. Jeu d'instructions des PIC16
IV. Ports d'entrées/sorties des PIC16
V. Fonctionnalités standards des PIC16
VI. Considérations techniques

I – Description globale

- Schéma fonctionnel d'un système informatique



I – Description globale

Definition : *Un système informatique est un système que fait interagir « du logiciel », (Software) sur « du matériel » (Hardware), en vue d'une finalité codée dans le logiciel.*

Les 5 éléments constitutifs du S.I représentés sont :

- L'ordinateur
- Le programme
- Les données en entrée
- Les données en sortie
- Les actions

I – Description globale

a) L'ordinateur : *Élément matériel central, il comporte essentiellement*

- un calculateur et sa mémoire, un ensemble clavier, écran, des périphériques informatiques conventionnels tels que : unités de sauvegarde, unités d'édition, unités de communication,
- des périphériques spécialisés : carte d'acquisition de son, d'images, des dispositifs de contrôle de processus industriels ...

b) Le programme : *Élément logiciel central, il indique à l'ordinateur*

- le type de traitement à effectuer sur les données d'entrée,
- les données à fournir et les actions à entreprendre en sortie.

c) Les données en entrée : *Elles peuvent être d'origines diverses*

- ensemble de nombres à traiter issus de la gestion bancaire, du traitement des signaux d'images, des bases de données ....
- informations en provenance de capteurs (température, hygrométrie...)
- données issues du clavier, (dialogue homme machine)

I – Description globale

d) Les données en sortie :

Elles représentent le résultat attendu du traitement spécifié par le programme des données en entrée.

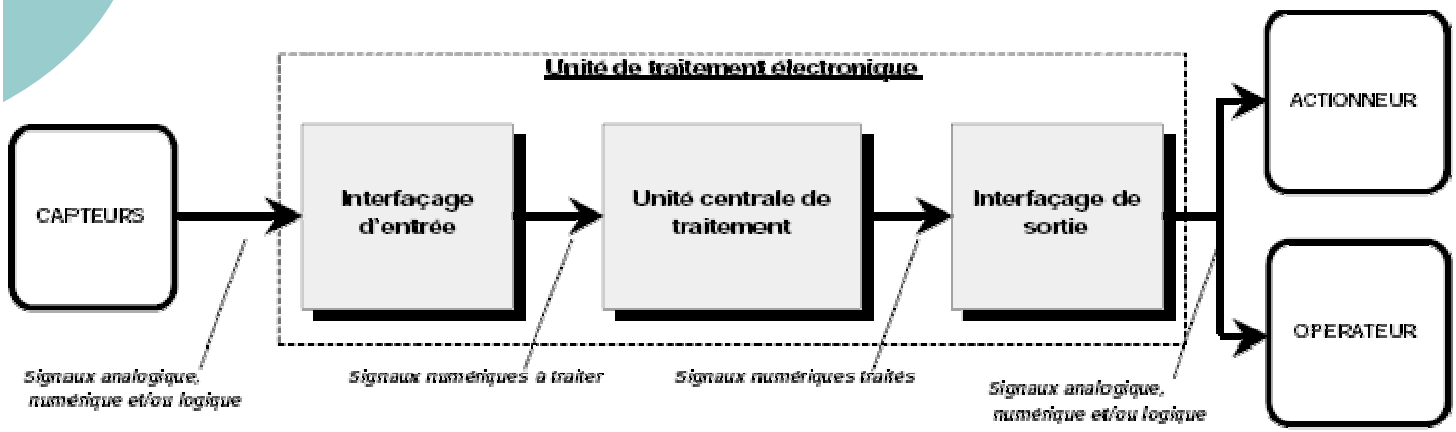
e) Actions :

Elles sont couramment destinées:

- au système informatique lui même (sauvegarde)
- au processus industriel commandé (marche, arrêt, commande de vanne, de moteur...)

I – Description globale

- Exemple: *système à microcontrôleur*



I – Description globale
➔ II –Architecture

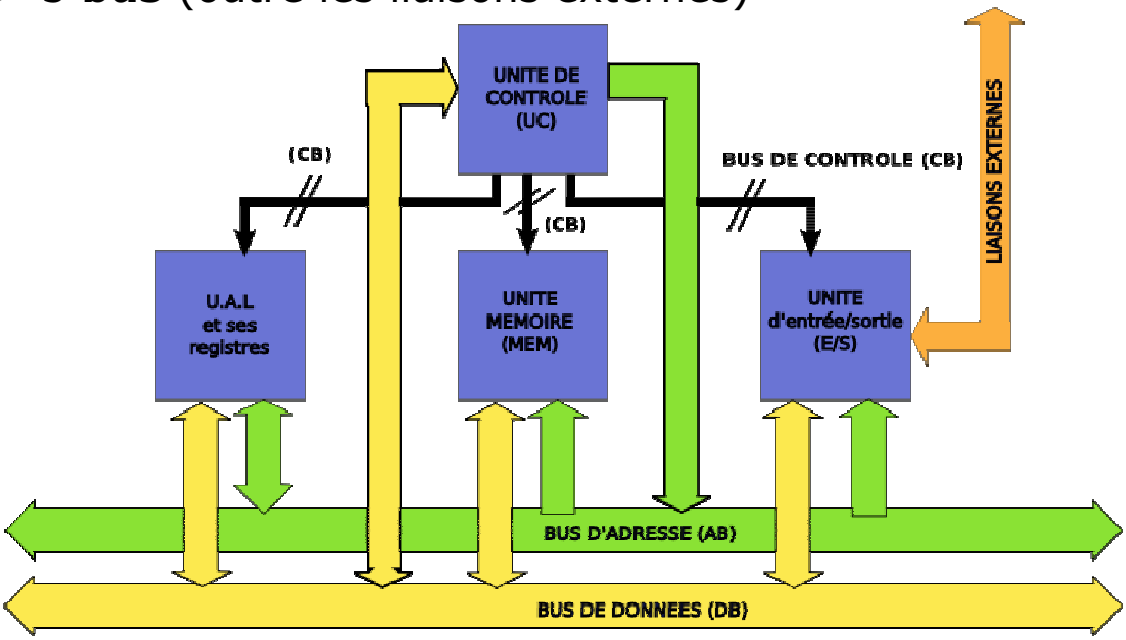
- Description de Von Neumann

II –Architecture de Von Neumann

- **Historique :**
  - Cette architecture a été introduite dans les années 50 à l'Institute for Advanced Study (IAS) de Princeton.
  - Von Neumann a donné son nom à l'architecture de Von Neumann utilisée dans la quasi totalité des ordinateurs modernes, l'apport d'autres collaborateurs de l'EDVAC en est par conséquent grandement minimisé (on citera J. Presper Eckert et John William Mauchly parmi d'autres). Cela est dû au fait qu'il est, en 1944, le rapporteur des travaux pionniers en la matière (First Draft of a Report on the EDVAC).
  - Le modèle de calculateur à programme auquel son nom reste attaché et qu'il attribuait lui-même à Turing, possède une **unique mémoire** qui sert à conserver **les logiciels et les données**. Ce modèle, extrêmement innovant pour l'époque, est à la base de la conception de nombre d'ordinateurs mais a fortement évolué depuis.

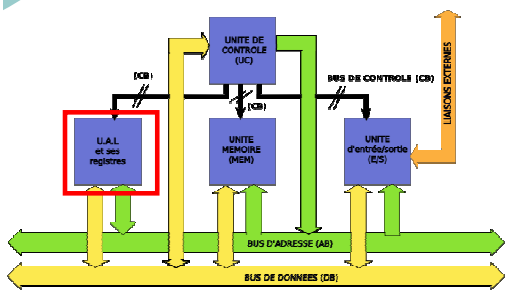
II –Architecture de Von Neumann

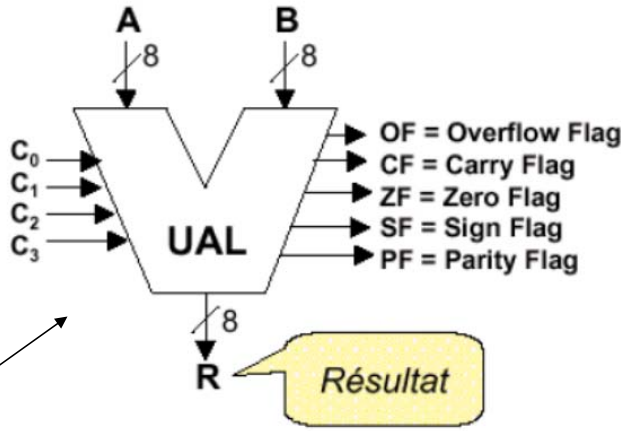
➔ L'architecture de von Neumann décompose l'ordinateur en
- 4 **unités** distinctes
- 3 **bus** (outre les liaisons externes)



II –Architecture de Von Neumann

- **Unité Arithmétique et Logique** (*UAL*)
  - o effectue les opérations arithmétiques ou logiques élémentaires (+, -, *, /, … ; ET, OU, complémentation…)
  - o une batterie de registres « généraux » permet de stoker **temporairement** les opérandes et résultats en cours. L'*accumulateur* désigne le registre stockant l'une des deux opérandes et le résultat (A, AC, ACC, W).





Représentation standard

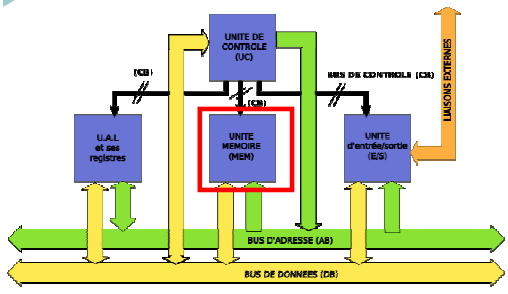
II –Architecture de Von Neumann

- **Unité Mémoire**

2 types de mémoires fonctionnellement et technologiquement différents:

o mémoire de masse (**ROM**, Read Only Memory) stocke un programme figé et/ou une table de constantes

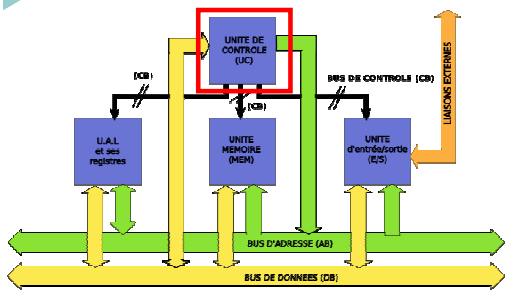
o mémoire vive (**RAM**, Random Access Memory) stocke des variables et des segments de programmes susceptibles d'être écrasés par d'autres après qu'ils aient été exécutés.



➔ Une position mémoire particulière est sélectionnée par le biais de son adresse, puis lue ou écrite via le bus de données

II –Architecture de Von Neumann

- **Unité de contrôle** (*UC*)
  - **chef d'orchestre de l'ensemble des unités**
  - **réalise 3 taches:**



→ **lire** une instruction du programme en cours, la **décoder** et **pointer** vers la suivante*,

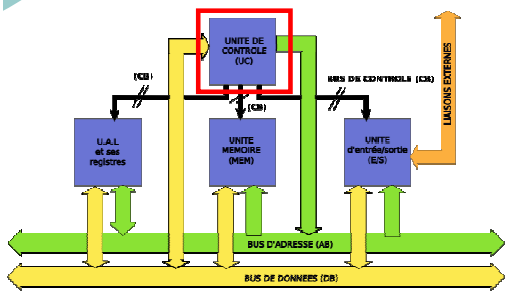
→ **faire exécuter** l'instruction par les différentes unités

→ **gérer les « événement exceptionnels »** susceptible d'intervenir de manière impromptus

*L'UC contient souvent un dispositif d'incrémentation du PC, évitant ainsi de surcharger et de ralentir l'UAL

II –Architecture de Von Neumann

- **Unité de contrôle** (*UC*)
  - **comporte 2 registres principaux :**



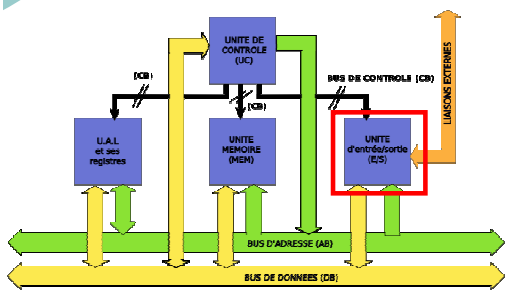
→ **PC** (Programme counter) ou **IP** (Instruction Pointer) qui « pointe » vers l'adresse de la prochaine instruction à traiter

→ **IR** (Instruction Register) qui contient le code de l'instruction courante.

→ Cycles instruction

II –Architecture de Von Neumann

- **Unité d'entrées et sorties** (E/S, I/O)
  - Interface entre le monde externe et le calculateur
    → importante diversité des périphériques connectables,
    → nombreux protocoles d'échanges (propre aux périphériques externes)

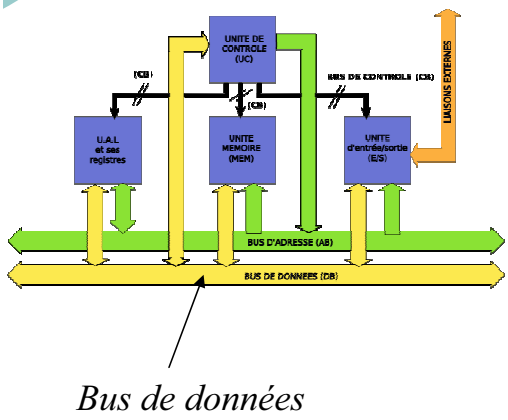


2 types de communications:

- communication série
  les bits sont transmis les uns après les autres
- communication parallèle
  les bits d'un bloc sont transmis simultanément

II –Architecture de Von Neumann

- **Bus de Données** (Data Bus)
  - voie de communication (de données) **bidirectionnelle** pouvant être « écrite » ou « lue » par une unité quelconque.



Bus de données

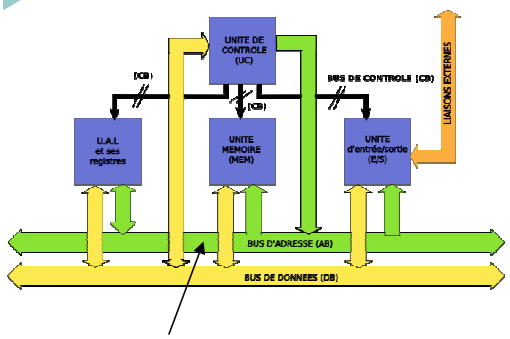
➔ A un instant donné, une seule unité est autorisée à écrire ou lire une information sur *DB*, les autres étant temporairement "déconnectées électriquement" du bus de données.

➔ L'UC coordonne l'utilisation de ce bus par l'intermédiaire du bus de contrôle (*CB*).

➔ Taille de *DB* souvent identique à celle des registres de l'UAL. (représentative de la puissance de calcul de la machine).

II –Architecture de Von Neumann

- **Bus d'Adresses** (Address Bus)
  - o voie de communication **unidirectionnelle**
    → seuls l'*UC* ou l'*UAL* peuvent ecrire
    → seuls les unités mémoires ou d'E/S peuvent lire



Bus d'adresses

Remarques:

• L'*UC* positionne l'adresse de l'instruction à traiter, et généralement l'*UAL* qui contient (soit directement soit à l'issu d'un calcul) l'adresse d'un opérande en mémoire ou en E/S.

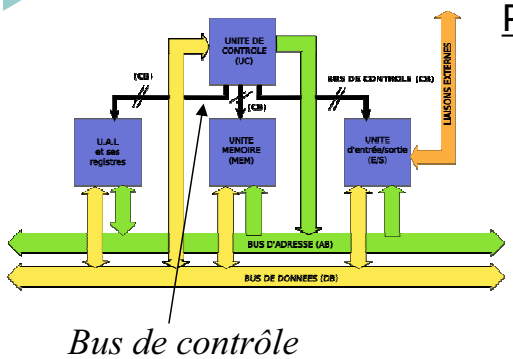
• La taille de ce bus T_{AB} en bit (nombre de fils de *AB*) représente la capacité maximale C_m d'adressage du calculateur, suivant la relation

$$C_m = 2^{T_{AB}}$$

→ Qq ex.

II –Architecture de Von Neumann

- **Bus de Contrôle** (Control Bus)
  ➔ Permet à l'UC de synchroniser et de commander les autres unités (sélection d'adresse (Mem), selection d'une opération (UAL), activation/désactivation des unités, type d'accès (lecture/écriture)…)



Bus de contrôle

Pour l'éxecution d'une instruction:

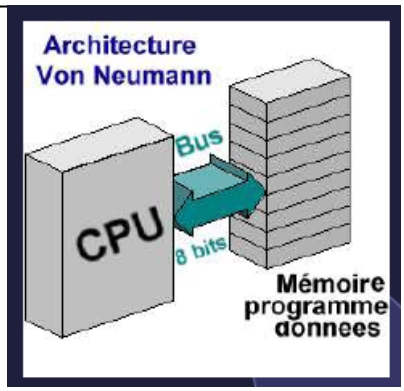
- o sélection des registres/mémoires source et destinataires du ou des opérandes,
- o commande de l'instant des transferts des opérandes,
- o sélection de l'opération particulière à effectuer dans l'UAL et du destinataire du résultat,
- o commande de l'instant du transfert du résultat

Remarques: **➔Découpage d'une instruction en plusieurs cycles**
➔Synchronisation sur signal d'horloge

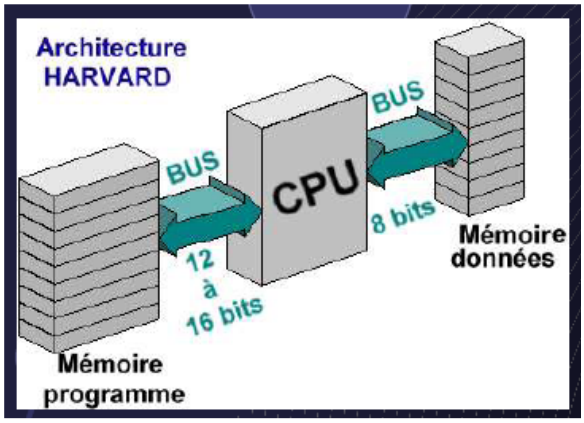
II –Architecture de Von Neumann

➔ **Evolution de l'architecture :**
séparation des espaces mémoires programme et données

→ architecture ***Harvard***



Architecture Harvard :
- mémoires programme et données distinctes,
- bus programme et données distincts,
- transfère simultané des instructions à exécuter et des données.
- modèle plus rapide que Von Neumann,
- structure interne plus complexe complexité interne de la structure.



II –Architecture de Von Neumann
➔ III –Communication par bus

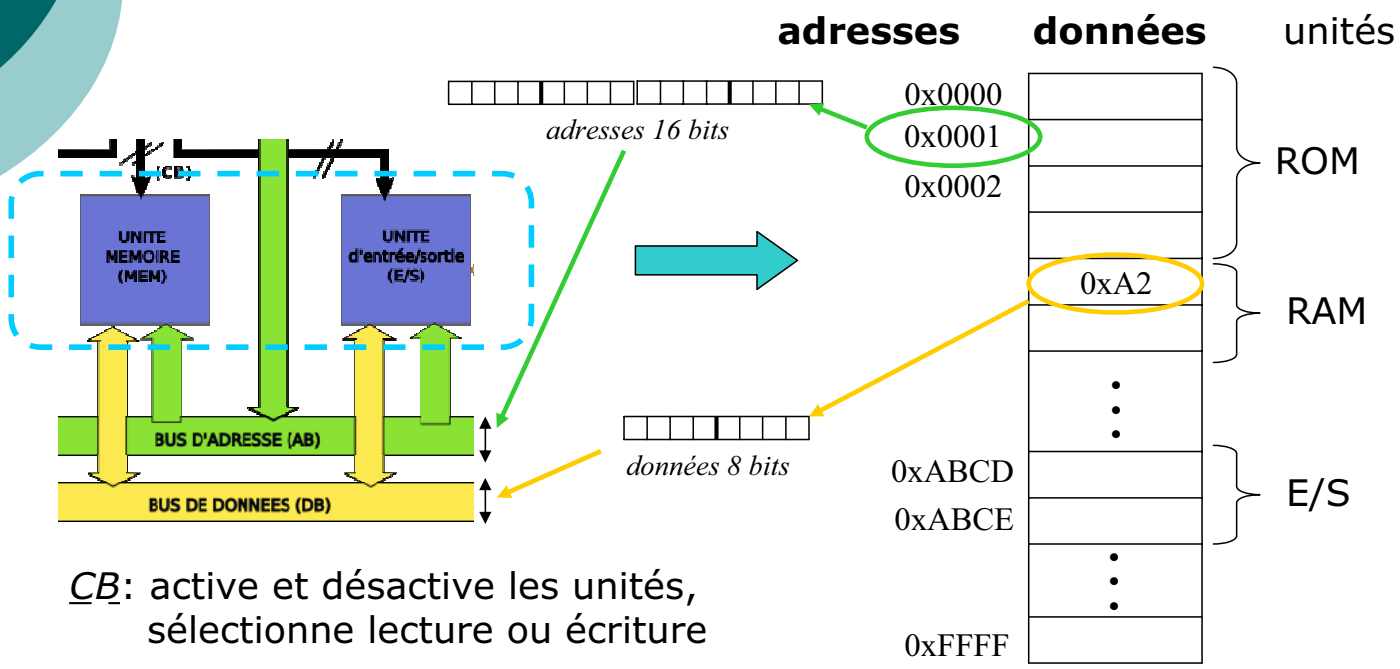
gérer les accès aux bus bidirectionnels

⇔

activer une unité (ou boitier)
et désactiver les autres

III –Communication par bus
1- Représentation *plan mémoire*

➔ Exemple:
bus d'adresse 16 bits, bus de données 8 bits



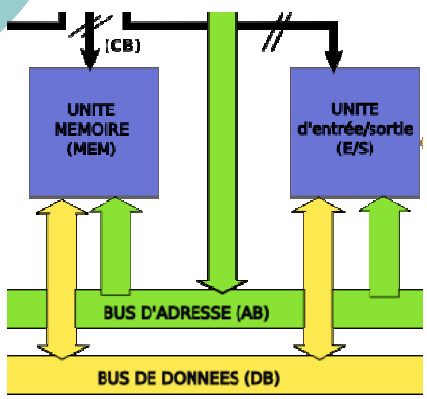
CB: active et désactive les unités, sélectionne lecture ou écriture

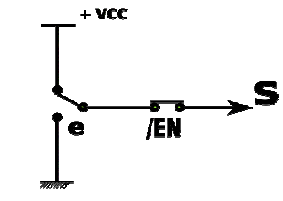
III –Communication par bus
2- Logique 3 états

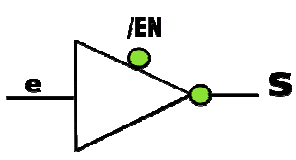
- Communication bidirectionnelle :
Comment désactiver des unités ?

➔ **Logique 3 états**

o On ajoute aux deux états un 3ième état noté HZ (haute impédance)
o Un circuit dans l'état HZ est déconnecté électriquement des autres (insensible aux modifications de ces entrées et n'influe pas sur l'états des autres circuits)





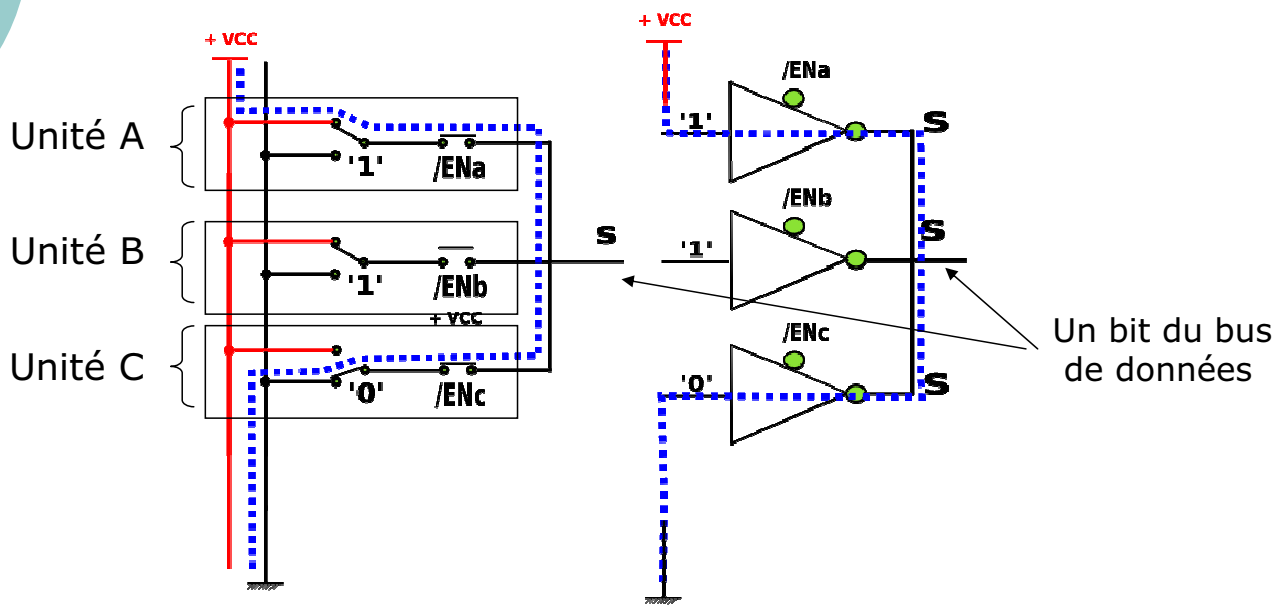


/EN	e	S	impédance
0	0	0	faible
0	1	1	faible
1	0	Z	forte
1	1	Z	forte

III –Communication par bus
2- Logique 3 états

➔ A chaque instant, au plus une unité doit être active *en écriture* sur un bus bidirectionnel, sinon: risque de court circuit!

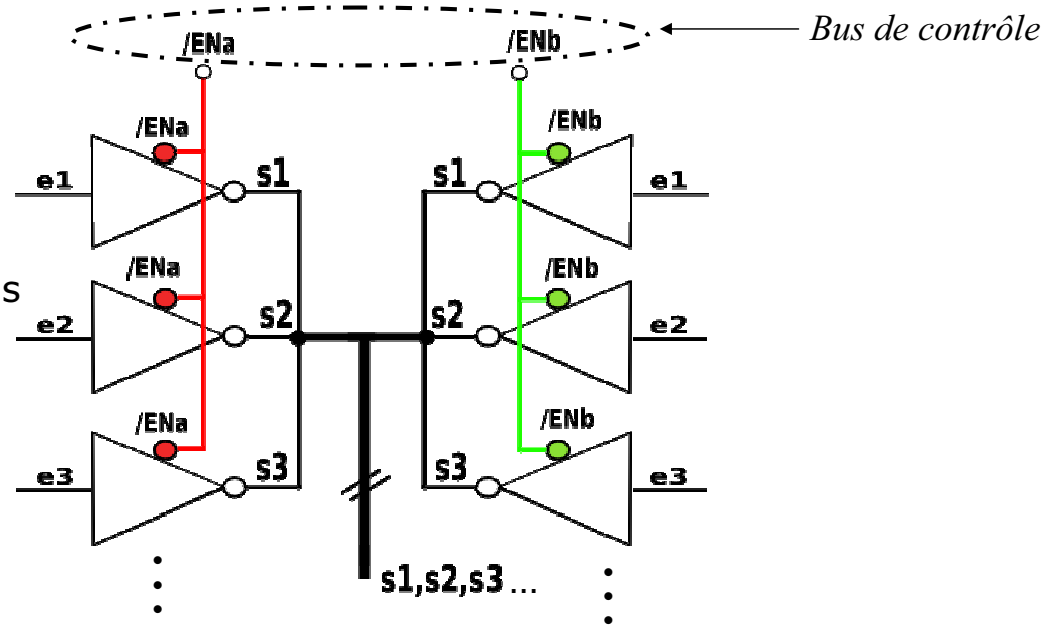
généralement, en plus de l'UC: une seule unité active



III –Communication par bus
2- Logique 3 états

➔ Le bus de contrôle gère l'activation des unités (ou circuits)

Ex: écriture sur Bus données

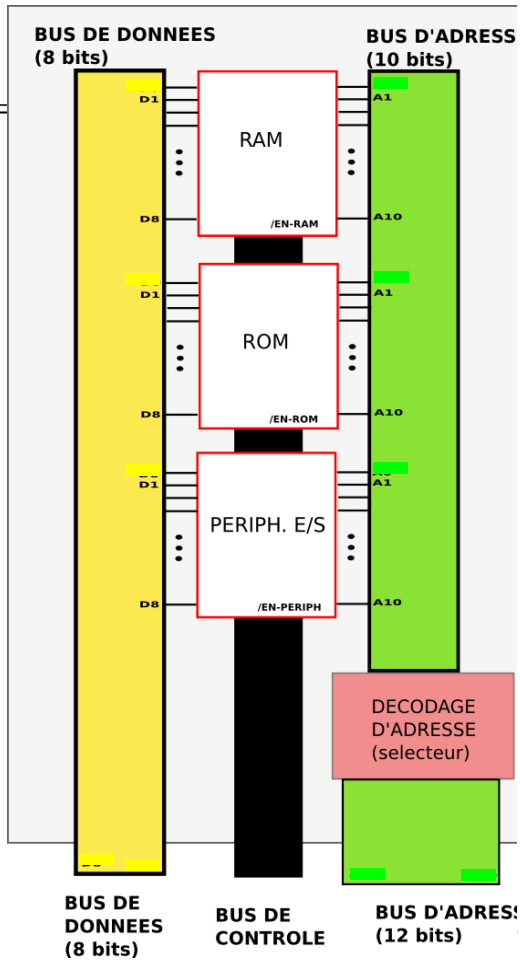


➔ Accès R/W ?

III –Communication par bus

3- Exemple

→Bus de données : 8 bits
→Bus d'adresses : 12 bits
→RAM : 1 ko
→ROM : 1 ko
→E/S : 1 ko



Bus d'adresses (12bits)

A12	A11	A<10:1>	sélection	zone adressée	Plage d'adresse
0	0	0000000000	/ENRAM	RAM	000h
0	0	1111111111			3FFh
0	1	0000000000	/ENROM	ROM	400h
0	1	1111111111			7FFh
1	0	0000000000	/ENPERIPH	PERIPH. E/S	800h
1	0	1111111111			BFFh
1	1	0000000000	libre		C00h
1	1	1111111111			FFFh

→ Activation/désactivation

IV – Diversité et applications

IV – Diversité et applications

- Caractéristiques d'un microcontrôleur
  - Périphériques: CAN/CNA(PWM), ports d'entrée/sortie, ports séries (protocole I2C, usb…), compteur, IRQ,…
  - Mémoires
    - programme ROM (qq ko à qq 100ko)
    - données RAM (qq ko à qq 100ko)
    - données non volatile (qq ko à qq Mo)
  - Registres
    - largeurs: 8, 16, 32 bits…
    - type (opérande): entiers, virgule fixe/flottantes
  - Fonctionnalité internes
    - instructions: fréq. exécution, nombres et types (maths, spécialisées)
    - compteurs/timers, Watch Dog, …
    - débogage
  - Alimentation
    - plage de tension de fonctionnement
    - consommation électrique

IV – Diversité et applications

- Fabricants et familles
  - Intel (80C51)
  - Motorola (Série 68HC11)
  - Atmel (Série AVR)
    - Comfile Technology (Cubloc, basé sur uP Atmel)
  - Parallax (Série Basic Stamp)
  - Arizona Microchip (Série PIC)
  - Rabbit Semiconductor (Série Rabbit)
  - Cypress Microsystem (PSoC)
  - …

IV – Diversité et applications

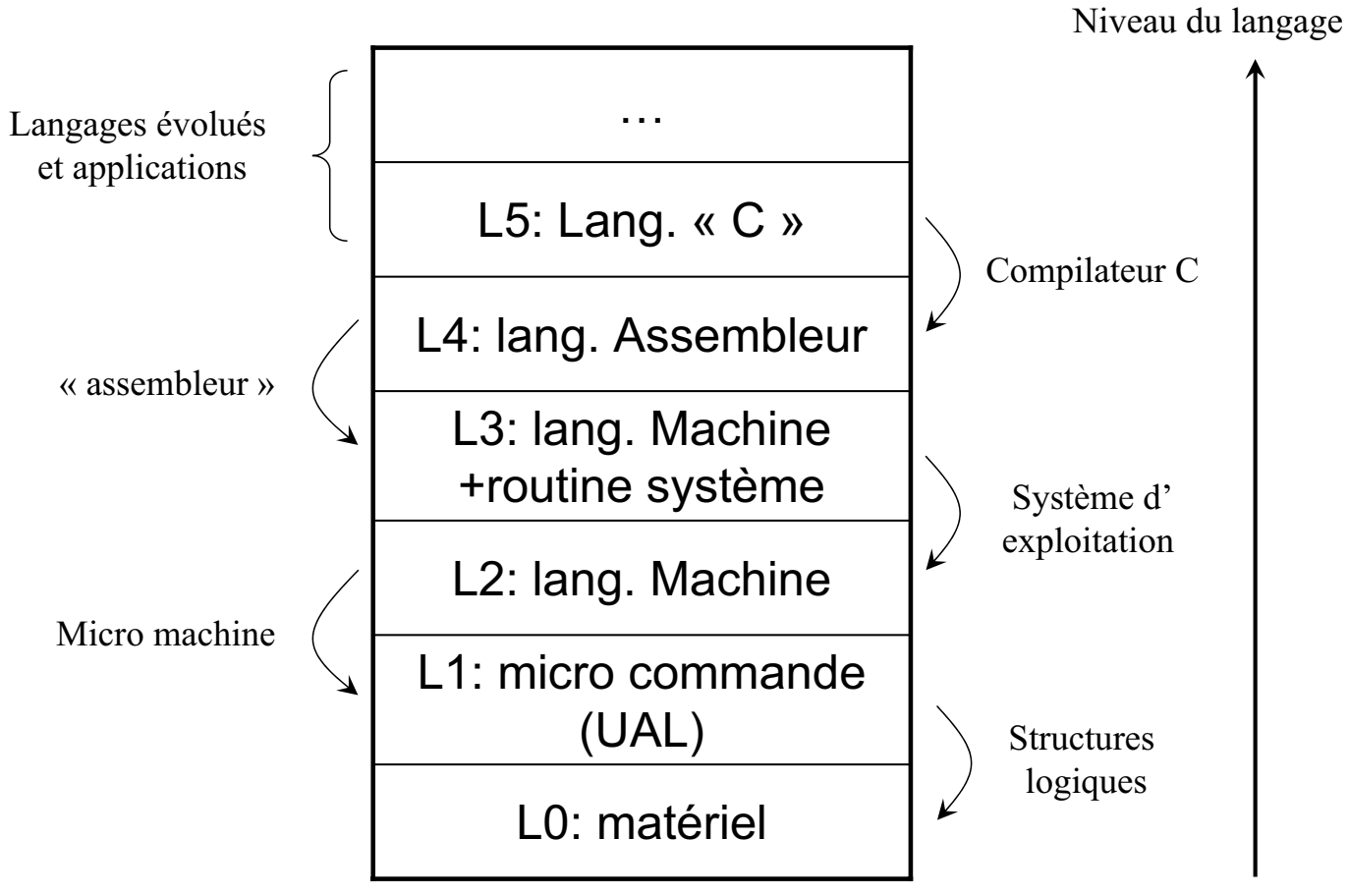
- interfaçage ?

  ➔ *microcontrôleur !!*

- Utilisation en augmentation
  - miniaturisation des cartes
  - système autonome: alimentation (faible consommation), communication, …
  - puissances de calcul diverses: mise en forme de signal, pilotage d'automate → applications complexes avec calcul

Programme

Structuration en couches logicielles



Plan

I. Généralités

II. Les microcontrôleurs PIC

I. Familles PICs

II. Caractéristiques des PICs

Mémoire, E/S standards et fonctionnalités intégrées

III. Architecture PIC16

Structure, Registres, Mémoires RAM / ROM

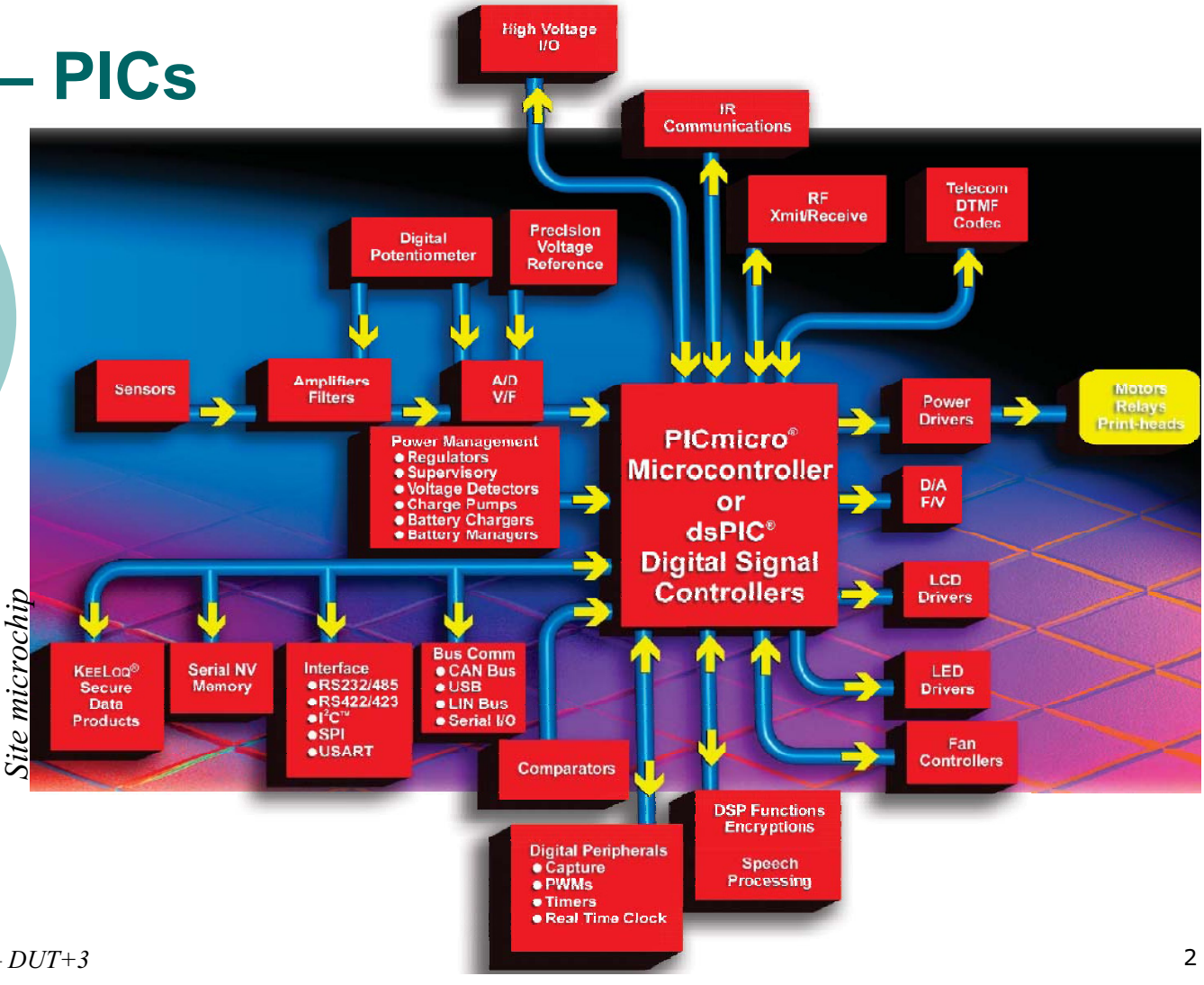
III. Jeu d'instructions des PIC16

IV. Ports d'entrées/sorties des PIC16

V. Fonctionnalités standards des PIC16

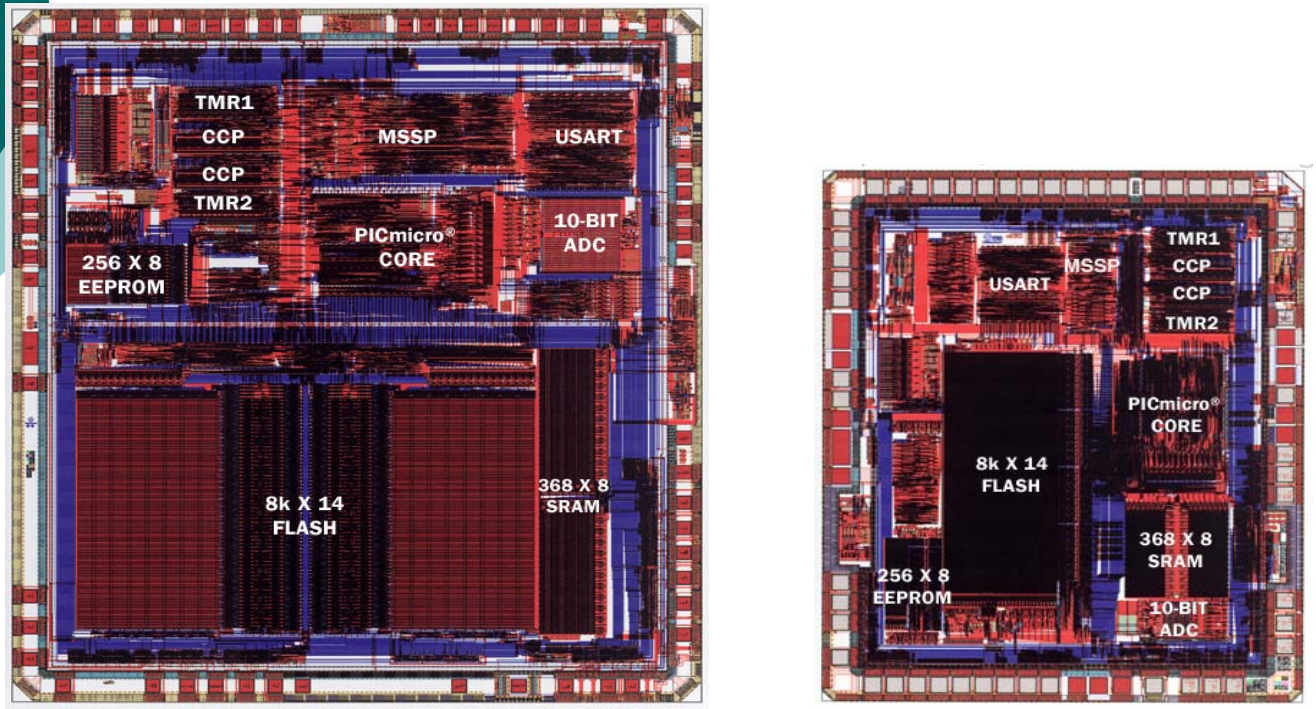
VI. Considérations techniques

I – PICs



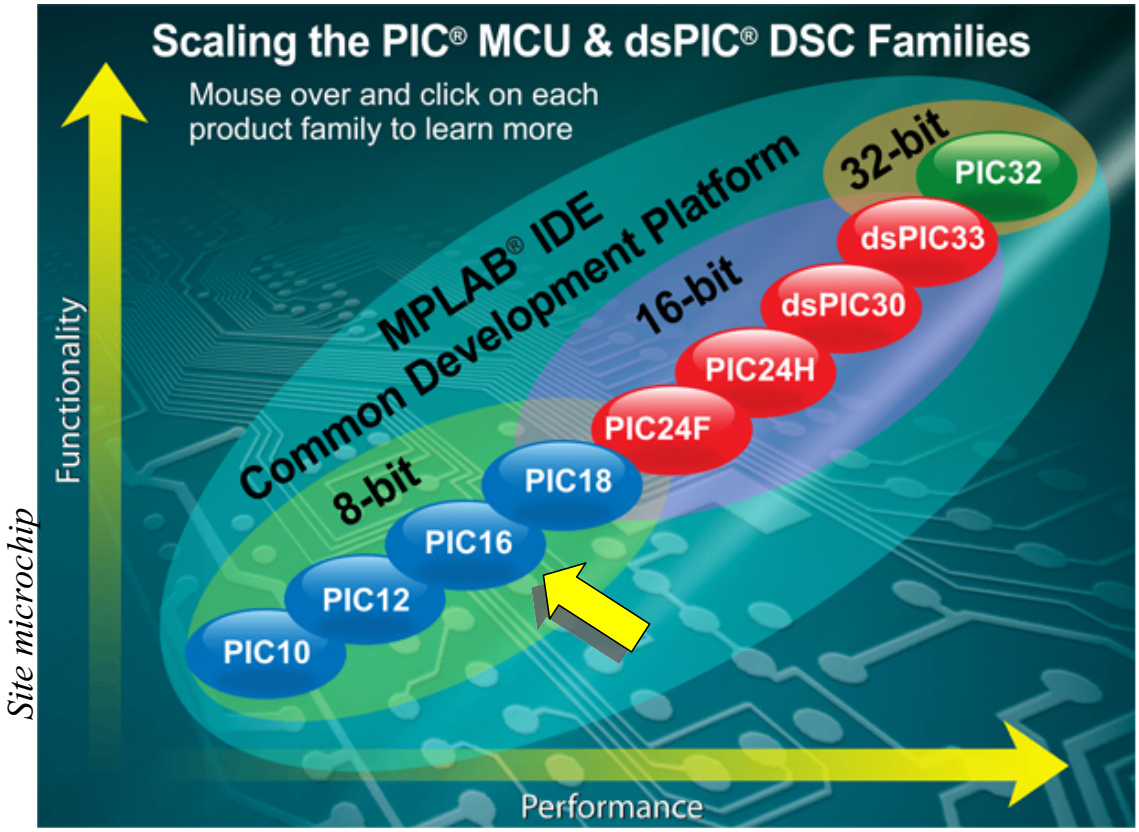
Site microchip

I – Famille PICs: intégration



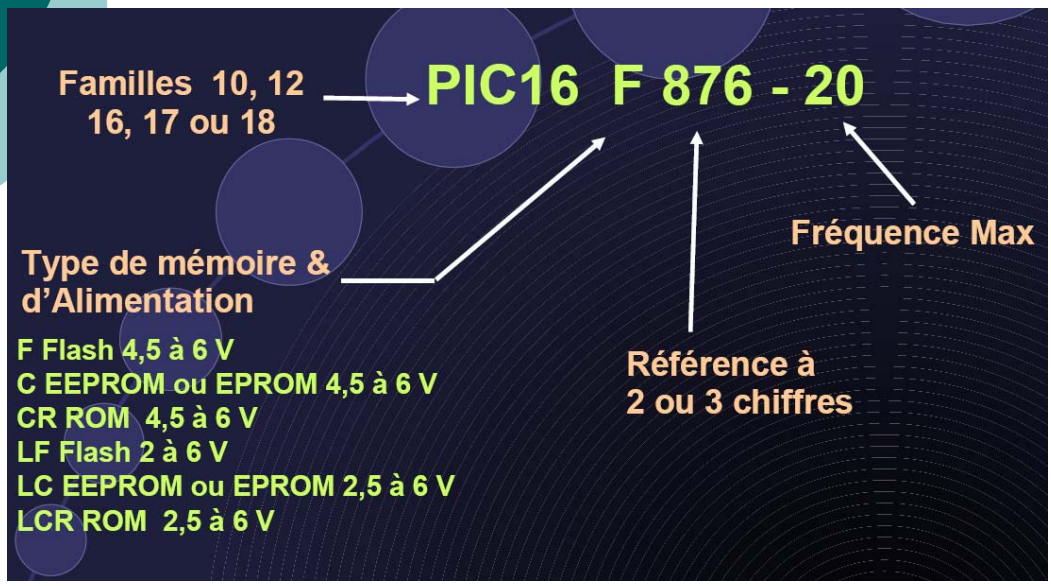
PIC 16F87x

I – Familles PICS



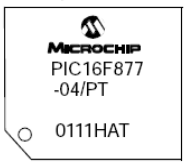
Site microchip

I – Familles PICs: identification



Example

Example



Example

II – Caractéristiques

8-bit Microcontrollers

Family	Devices In Family	Pin Count	Flash (Kbytes)	EEPROM (Bytes)	RAM (Bytes)	ADC Ch	Comp	Timers	CCP/ ECCP	Interfaces	Other Features
PIC10 – 500 ns Instruction Execution, 33 Instruction, 8 MHz											
PIC10F222	6	6	0.75/0.375	–	24/16	2x8-bit	1	1x8-bit	–	–	Internal Bandgap reference
PIC12 – 500 ns Instruction Execution, 33 Instruction, 20 MHz											
PIC12F509	2	8	1.5/0.75	–	41/25	–	–	1x8-bit	–	–	
PIC12F510	1	8	1.5	–	38	3x8-bit	1	1x8-bit	–	–	Internal Bandgap reference
PIC12F635	1	8	1.75	128	64		1	1x8-bit, 1x16-bit	–	–	KEELOQ®, WUR
PIC12F675	2	8	1.75	128	64	4x10-bit	1	1x8-bit, 1x16-bit	–	–	
PIC12F683	1	8	3.5	256	128	4x10-bit	1	1x16-bit, 2x8-bit	1/0	–	
PIC16 – 100-200 ns Instruction Execution, 35 Instruction, 20 MHz											
PIC16F50X	2	14	1.5	–	72 / 67	3x8-bit	2	1x8 bit	–	–	Internal Bandgap reference
PIC16F676	2	14	1.75	128	64	8x10-bit	2	1x8 bit, 1x16 bit	–	–	
PIC16F684	1	14	3.5	256	128	8x10-bit	2	1x16 bit, 2x8 bit	0/1	–	
PIC16F688	1	14	7	256	256	8x10-bit	2	1x8 bit, 1x16 bit	–	EUSART	
PIC16F716	1	18	3.5	–	128	4x10-bit	–	1x16 bit, 2x8 bit	0/1	–	
PIC16F690	6	20	7/3.5/1.75	256/128	256/128/64	12x10-bit	2	1x16 bit, 2x8 bit	0/1	EUSART, I²C/SPI	Internal Bandgap reference
PIC16F785	1	20	3.5	256	128	12x10-bit	2	1x16 bit, 2x8 bit	1/0	–	2x Op amp, Int Shunt Reg
PIC16F946	1	64	14	256	336	8x10-bit	2	1x16 bit, 2x8 bit	2/0	AUSART/I²C/SPI	**LCD Control - 96 segments**
PIC16F819	2	20/18	3.5/1.75	256	256/128	5x10-bit	–	1x16 bit, 2x8 bit	1/0	I²C/SPI	
PIC16F88	2	20/18	3.5	256	368/256	7x10-bit	2	1x16 bit, 2x8 bit	1/0	AUSART/I²C/SPI	
PIC16F5X	3	40/28/18	3/0.75	–	134/25	–	–	1x8 bit	–	–	
PIC16F77	5	44/28	14/7/3.5	–	368/192	8 x 8-bit	–	1x16 bit, 2x8 bit	2/0	USART, I²C/SPI	PSP
PIC16F777	4	44/28	14/7	–	368	14x10-bit	2	1x16 bit, 2x8 bit	3/0	AUSART, MI2C/SPI	PSP
PIC16F877A	7	44/28	14/7/3.5	256	368/192	8x10-bit	2	1x16 bit, 2x8 bit	2/0	AUSART, MI²C/SPI	PSP
PIC16F917	4	44/28	14/7	256	352	8x10-bit	2	1x16 bit, 2x8 bit	2/0	AUSART/I²C/SPI	**LCD Control - 96 segments**

II – Caractéristiques

8-bit Microcontrollers

Family	Devices In Family	Pin Count	Flash (Kbytes)	EEPROM (Bytes)	RAM (Bytes)	ADC Ch	Comp	Timers	CCP/ ECCP	Interfaces	Other Features
PIC18 – 100 ns Instruction Execution, 77 Instruction, 40 MHz											
PIC18F1320	2	20/18	8/4	256	256	7x10-bit	–	3x16 bit, 2x8 bit	0/1	EUSART	
PIC18F1330	2	20/18	8/4	128	256	4x10-bit	3	2x16 bit	–	EUSART	**Motor Control PWMs**
PIC18F4431	4	44/28	16/8	256	768/512	9x10-bit	–	3x16 bit, 2x8 bit	2/0	EUSART, MI²C/SPI	**Motor Control PWMs**
PIC18F4523	4	44/28	32/16	256	1536/768	13x12-bit	2	3x16 bit, 2x8 bit	1/1	EUSART, MI²C/SPI	
PIC18F4550	6	44/28	32/24/16	256	2048/768	13x10-bit	2	3x16 bit, 2x8 bit	1/1	MI²C/SPI, EUSART	**Full Speed USB 2.0**
PIC18F4620	16	44/28	64/48/32/16	1024	3968/1536	13x10-bit	2	3x16 bit, 2x8 bit	1/1	EUSART, MI²C/SPI	PSP
PIC18F4680	8	44/28	64/48/32/16	1024	3328/1536	11x10-bit	2	3x16 bit, 2x8 bit	1/1	EUSART, MI²C/SPI	**CAN 2.0B**
PIC18F45J10	4	44/28	32/16	–	1024	13x10-bit	2	3x16 bit, 2x8 bit	1/1	2xEUSART, 2xMI²C/SPI	PSP
PIC18F8490	4	80/64	16/8	–	768	12x10-bit	2	3x16 bit, 2x8 bit	2/0	MI²C/SPI, 2 x USART	**LCD: up to 192 segments**
PIC18F8680	4	80/64	64/48	1024	3328	16x10-bit	2	3x16 bit, 2x8 bit	1/1	EUSART, MI²C/SPI	**CAN 2.0B**, EMA
PIC18F8722	12	80/64	128/64/32/16	1024	3936/2048	16x10-bit	2	3x16 bit, 2x8 bit	2/3	2xEUSART, 2xMI²C/SPI	PSP, EMA
PIC18F87J10	10	80/64	128/96/64/ 8/32	–	3936/2048	15x10-bit	2	3x16 bit, 2x8 bit	2/3	2xEUSART, 2xMI²C/SPI	PSP, EMA
PIC18F97J60	9	100/80/64	128/96/64	–	3808/2048	16x10-bit	2	3x16 bit, 2x8 bit	2/3	2xEUSART, 2xMI²C/SPI	**10 BASE-T Ethernet**

II – Caractéristiques

16-bit Microcontrollers and Digital Signal Controllers

Pins	Flash Memory (Kbytes)	SRAM Kbytes	Timers 16-bit	Input Capture	Output Comp/PWM	Analog	Communications Serial I/O	Additional Features
PIC24F Family – 16 MIPS, Lowest Cost, General Purpose								
28/44	32-64	8	5	5	5	10-13x 10-bit (500 ksps), 2 comparators	UART w/IrDA* (2), SPI (2), I²C (2)	JTAG, Parallel Master Port (PMP), Real Time Clock Calendar (RTCC)
64-100	64-128	8	5	5	5	16x 10-bit (500 ksps), 2 comparators	UART w/IrDA* (2), SPI (2), I²C (2)	JTAG, Parallel Master Port (PMP), Real Time Clock Calendar (RTCC)
PIC24HJ Family – 40 MIPS, Highest Performance, General Purpose								
64-100	64-256	8-16	9	8	8	User selectable 12-bit A/D (500 ksps) or 10-bit A/D (1.1 Msps), 16 ch.	UART w/IrDA* (2), SPI (2), I²C, CAN (0,1,2)	JTAG, 8 ch. DMA
dsPIC30F Sensor Family – 20, 30 MIPS, Digital Signal Controllers								
18-28	12 or 24	1/2	3	2	2	8/10 ch. 12-bit A/D (200 ksps)	UART (1,2), SPI, I²C	SOIC, PDIP, QFN (6x6 mm) packages
dsPIC30F General Purpose Family – 20, 30 MIPS, Digital Signal Controllers								
40-80	24-144	2-8	3-5	2-8	2-8	13/16 ch. 12-bit A/D (200 ksps)	UART (2), SPI (1,2), I²C, CAN (0,1,2)	Codec Interface - AC97/I²S
dsPIC33F General Purpose Family – 40 MIPS, Digital Signal Controllers								
64-100	64-256	8-30	9	8	8	User selectable 12-bit A/D (500 ksps) or 10-bit A/D (1.1 Msps), 16 ch.	UART (2), SPI (2), I²C (1,2), CAN (0,1,2)	Codec Interface, 8 ch. DMA
dsPIC30F Motor Control and Power Conversion Family – 20, 30 MIPS, Digital Signal Controllers								
28-80	12-144	0.5-8	3, 5	4-8	2-8	6/9/16 ch. 10-bit A/D (1000 ksps)	UART (2), SPI (1,2), I²C, CAN (0,1,2)	Motor control PWMs & Quad Encoder Modules
dsPIC33F Motor Control and Power Conversion Family – 20, 30 MIPS, Digital Signal Controllers								
64-100	64-256	8-30	9	8	8	User selectable 12-bit A/D (500 ksps) or 10-bit A/D (1.1 Msps), 16 ch.	UART (2), SPI (2), I²C (2), CAN (1,2)	Motor control PWMs (8) & Quad Encoder Modules, 8 ch. DMA

II – Caractéristiques, résumé

Famille	Mémoire programme (instruction)	RAM (Octets)	EEPROM (Octets)	Horloge (MHz)
PIC12	512-4096	25-128	0-256	4
PIC16	512-8192	25-368	0-256	4-20
PIC17	2048-16384	454-902	0	33
PIC18	8192-16383	256-1536	0-256	25-40

- ➔ Reduce Instruction Set Computer (RISC): PIC16 ⇔35 instructions
- ➔ Temps constant d'exécution des instructions (1 ou 2 cycles instruction)
- ➔ 1 type de mémoire programme / PIC : ROM, EEPROM, **FLASH**…
- ➔ 1 type de mémoire données / PIC : **RAM**
  et 1 type de mémoire données optionnel: **EEPROM**

III – Architecture PIC16

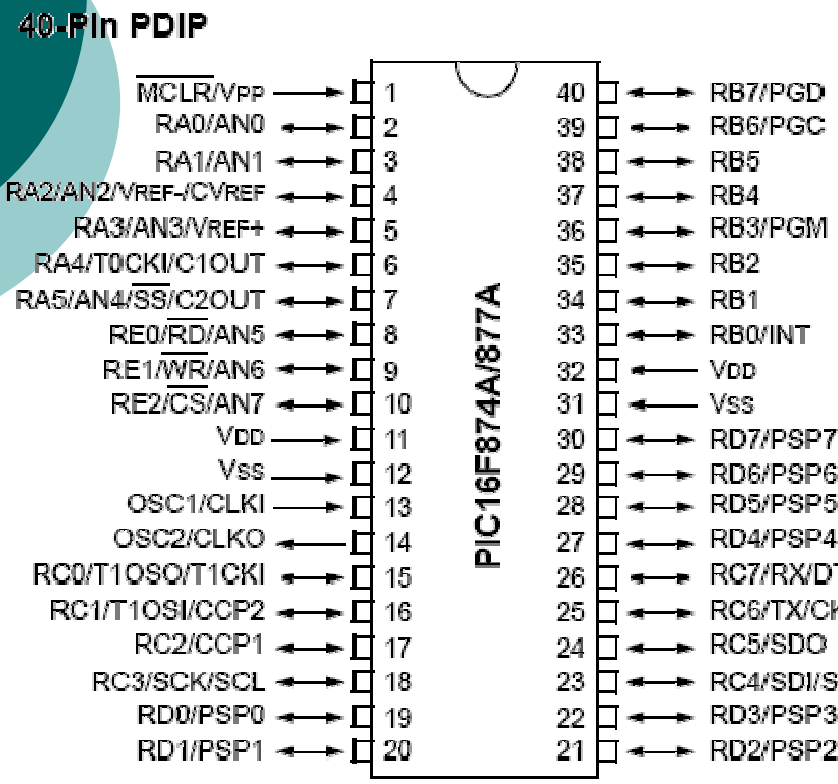
- Famille choisie : PIC16

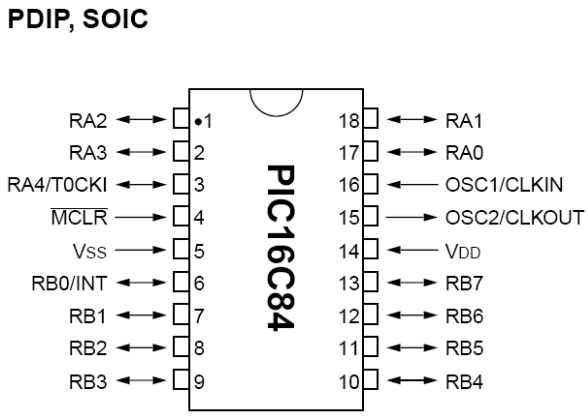
  → Microcontrôleurs utilisés

  PIC16 F 84 (doc TD)

  PIC16 F 877 (cours et TP)

III – Architecture: Boitier PIC 16F877

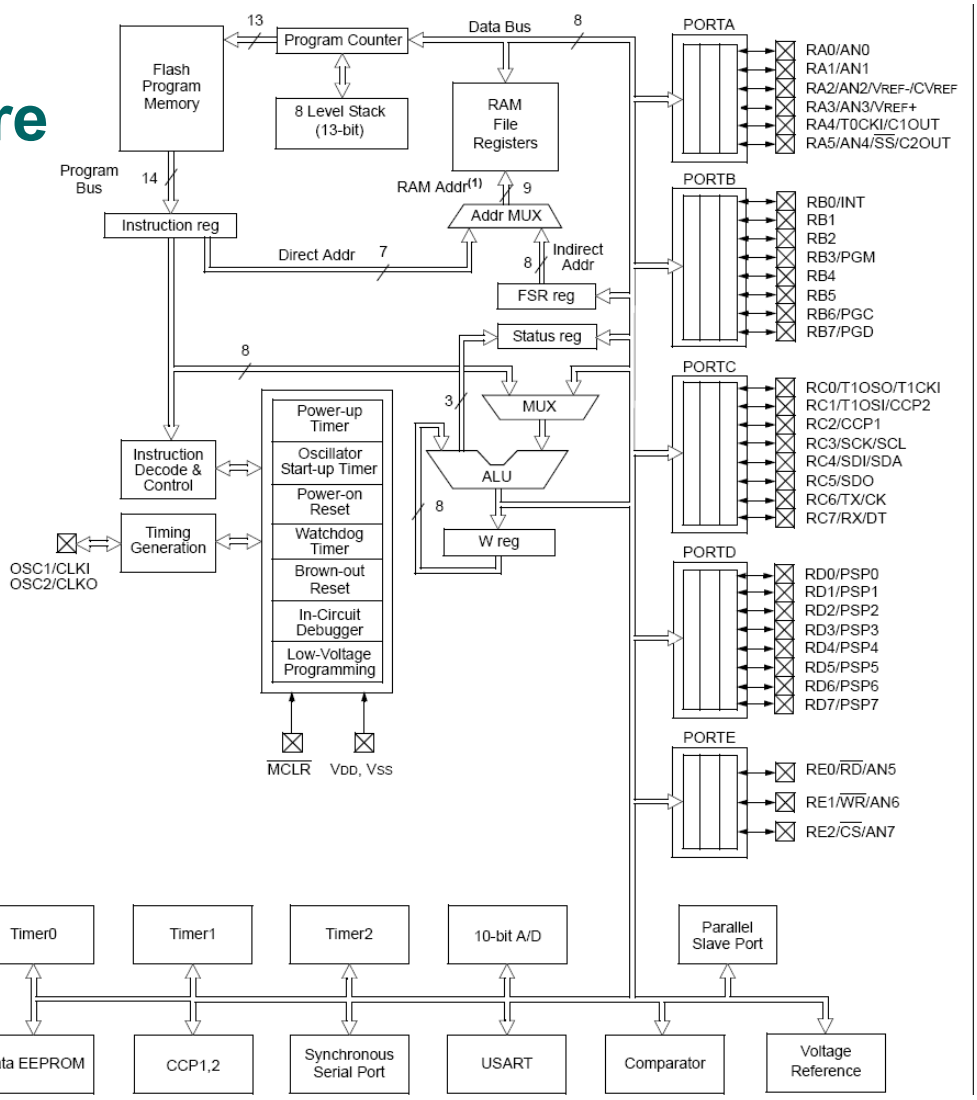




III – Architecture: Caractéristiques PIC 16F87x

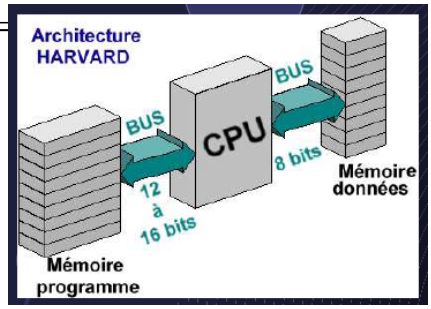
Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions

III – Architecture PIC16



Pic 16 F 877

III – Architecture (Harvard) des PIC16



- Tailles des bus
  - Bus Mémoire Données
    - Bus adresses: 9 bits (512 octets)
    - Bus données: 8 bits (taille des données)
  - Bus Mémoire Programme
    - Bus adresses : 13 bits (8k instructions)
    - Bus données : 14 bits (taille d'une instruction)

III – Architecture des PIC16

- Registres
  - Accumulateur: **W** (*8 bits*)
  - Pointeur programme: **PC** (*13 bits*)
  - Registre d'instruction: Instruction Reg (14 bits)
  - Drapeaux d'état UAL : 3 (Z, DC, C)
  - Adressage indirect :
    adresse: **FSR** (*8bits*), valeur: **INDF** (*8bits*)
  - Zone SFR: *Special Function Registers*
    - Paramétrer et accéder aux périphériques et fonctionnalités intégrés du PIC
    - **En RAM !**

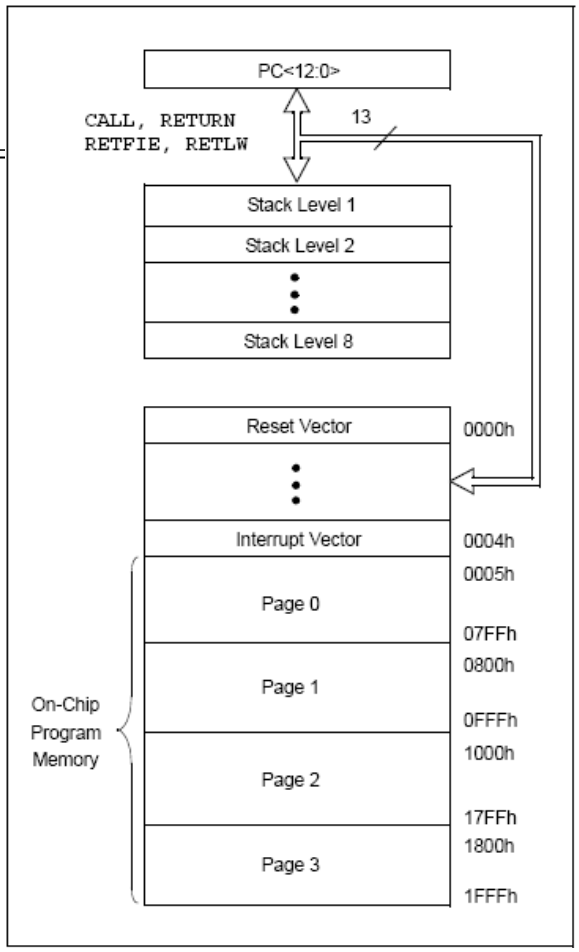
III – Architecture Mémoire

- Memoire programme ROM (*PIC 16F877*)
  - Flash 8k mots (1 mot = 14bits)
- Mémoiree Données RAM (*PIC 16F877*)
  - 368 octets à usage général (utilisateur)
- EEPROM (*PIC 16F877*)
  - *256 octets à usage général*

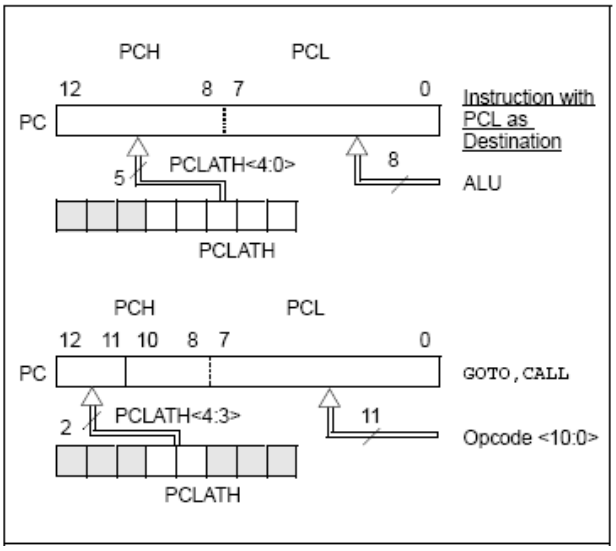
(utilisation détaillée dans une autre partie)

III – Architecture Mémoire ROM

- 4 pages de programme…
- Après un Reset
  Reset ➔ PC = 0000h
- Interruption
  - IRQ ➔ PC = 0004h
  - 1 seul vecteur
  - Gestion de la source d'IRQ par programme! **(TP 4GE)**
- Pile: 8 niveaux
  - Réservée à PC
  - Empilage/dépilage de PC auto. par le microcontrôleur
  - Empilage/dépilage des autres registres par programme !!! **(TP 4GE)**



III – Architecture Mémoire ROM

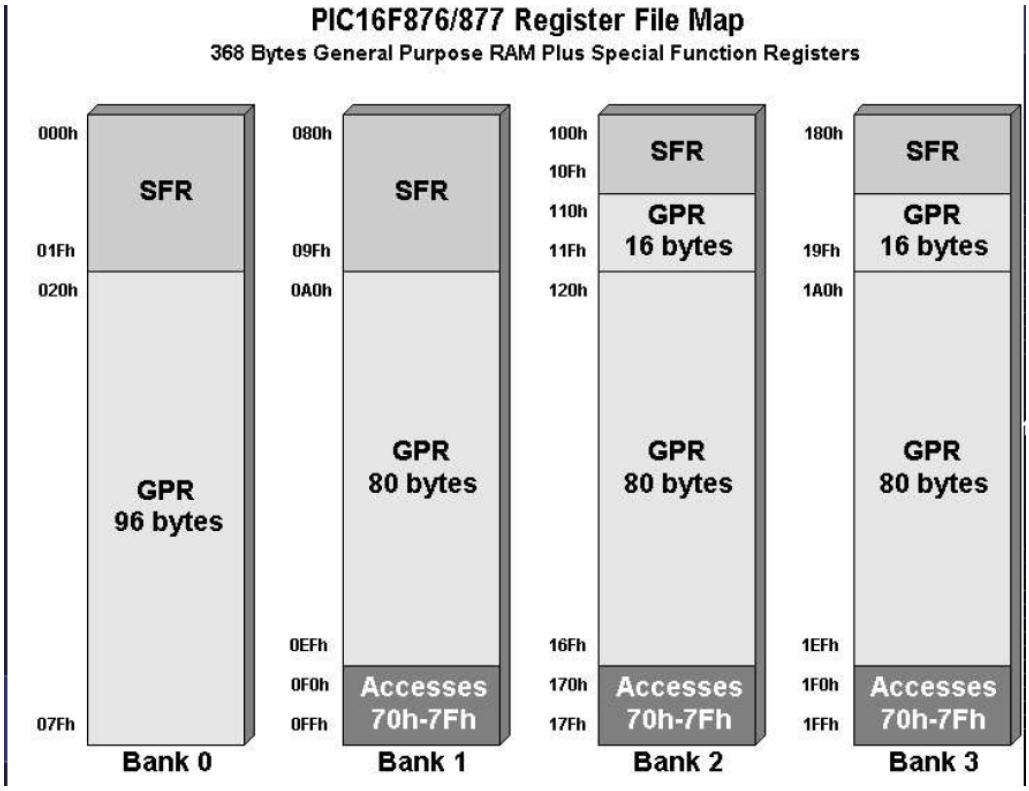



```
        ORG 0x500
        BCF PCLATH,4
        BSF PCLATH,3    ;Select page 1
                        ;(800h-FFFh)
        CALL SUB1_P1    ;Call subroutine in
        :               ;page 1 (800h-FFFh)
        :
        ORG 0x900       ;page 1 (800h-FFFh)
SUB1_P1
        :               ;called subroutine
                        ;page 1 (800h-FFFh)
        :
        RETURN          ;return to
                        ;Call subroutine
                        ;in page 0
                        ;(000h-7FFh)
```

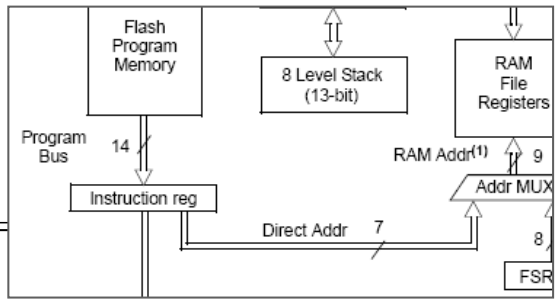
III – Architecture Mémoire RAM

SFR : Special Function Registers
Accès fonctionnalités du microcontrôleur

GPR : General Purpose Registers
Variables utilisateur



III – Architecture Mémoire RAM



- Problème
  - Bus adresse données : 9 bits
  - Nb bits adressage / instruction : 7 bits
  - ➔ Il manque 2 bits !!!
    - ➔ **RP1** et **RP0** bits les plus significatifs d'adressage des données, ajouté aux 7 bits d'adresses de l'instruction
    - ➔ Les valeurs de RP1 et RP0 créés 4 « Bank » en RAM

RP1:RP0	Bank
00	0
01	1
10	2
11	3

III – Architecture SFR

Détails: cf. doc technique

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h		190h	
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h	General Purpose Register 16 Bytes	197h	General Purpose Register 16 Bytes
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch	CMCON	11Ch		19Ch	
1Dh	CCP2CON	9Dh	CVRCON	11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh	accesses 70h-7Fh	16Fh	accesses 70h-7Fh	1EFh	accesses 70h - 7Fh
		F0h		170h		1F0h	
		FFh		17Fh		1FFh	

Unimplemented data memory locations, read as '0'.
* Not a physical register.
1: These registers are not implemented on the PIC16F876A.
2: These registers are reserved; maintain these registers clear.

III – Architecture SFR : bank

- Exemple d'accès aux Bank

Pour accéder à un registre SFR d'une bank

1) Paramétrer RP1 et RP0
2) Accéder au registre

Exemple:

```
BSF     STATUS,RP1      ;
BCF     STATUS,RP0      ; Bank 2
MOVP    DATA_EE_ADDR,W  ; Data Memory
MOVWF   EEADR           ; Address to read
BSF     STATUS,RP0      ; Bank 3
BCF     EECON1,EEPGD    ; Point to Data
                        ; memory
BSF     EECON1,RD       ; EE Read
BCF     STATUS,RP0      ; Bank 2
MOVP    EEDATA,W        ; W = EEDATA
```

III – Architecture: qq registres SFR

STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

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

OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Plan

II. Les microcontrôleurs PIC16

III. Jeu d'instructions des PIC16

I. Les types d'instruction

II. Adressage

III. Cycle instruction

IV. Les instructions dans les détails

V. Accès aux différentes mémoires

IV. Ports d'entrées/sorties des PIC16

V. Fonctionnalités standards des PIC16

VI. Considérations techniques

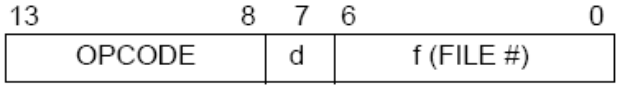
I – Instructions des PIC16

- PIC16: 35 instructions
- 3 catégories d'instructions
  - Manipulation de valeurs littérales (constantes)
  - Manipulation de variables en RAM (SFR ou GPR)
  - Manipulation de bit (mise à 1 ou à 0 d'un bit)

➢ *Support : Listes des instructions et détails de fonctionnement (issu du document constructeur PIC16 User Guide)*

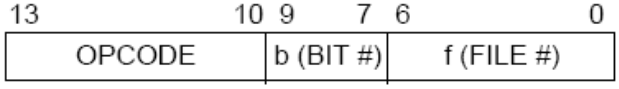
I - Format des instructions

Byte-oriented file register operations



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

Bit-oriented file register operations

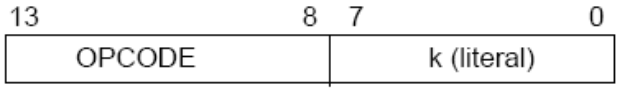


b = 3-bit bit address
f = 7-bit file register address

f : adresse en mémoire données (RAM : SFR et GPR) de 7 bits
000h → 07Fh

Literal and control operations

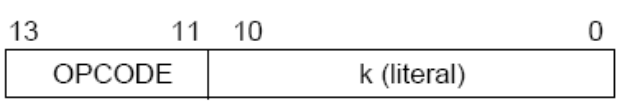
General



k = 8-bit literal (immediate) value

b : 3 bits pour déterminer une position dans un octet
000 → 111
0 → 7

CALL and GOTO instructions only



k = 11-bit literal (immediate) value

k : constante de 8 bits (0 → 255d)
ou de 11 bits (0h → 07FFh)

I – Instructions des PIC16 manipulation de constante

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb LSb	Status Affected	Notes
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

k: valeur constante ou littérale de 8 ou 11 bits

Ex: ***addlw*** 2 ; W + 2 → W

I – Instructions des PIC16 manipulation de *bit*

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb LSb	Status Affected	Notes
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3

f: adresse 7 bits d'une données en RAM (SFR ou GPR)
b: numéro du bit à manipuler (de 0 à 7)

Ex: ***btfsc*** 0x03, 0 ; test si le bit 0 de la valeur à l'adresse 0x03 = 0

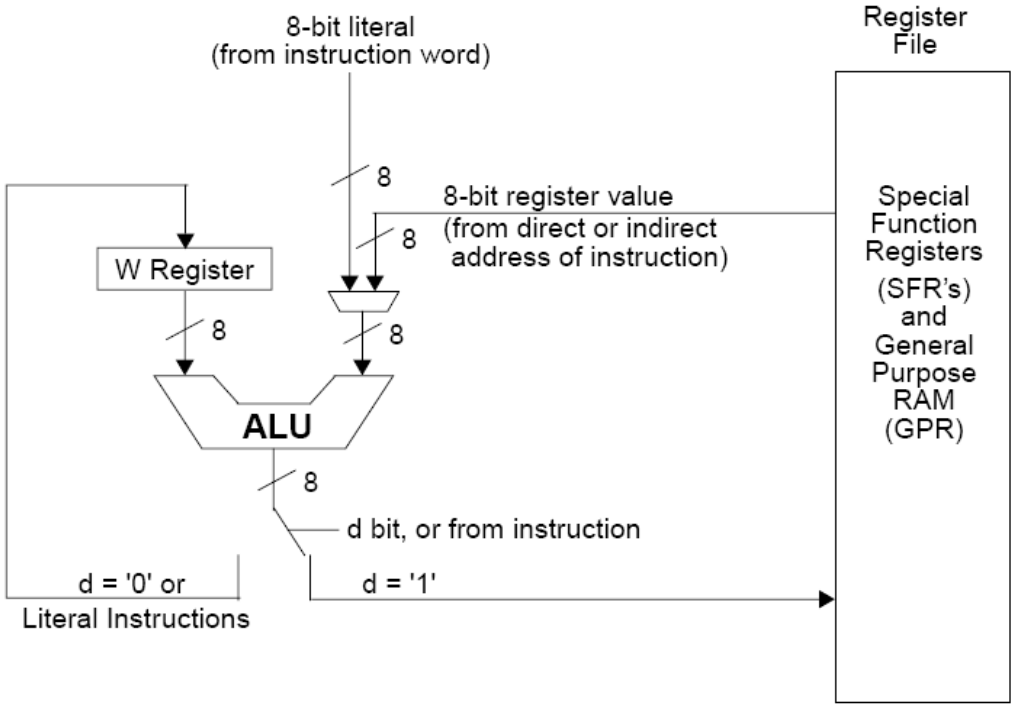
bcf 0x20, 4 ; cette instruction n'est pas exécutée si le test précédant est vrai

I – Instructions des PIC16
manipulation de variables RAM

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2

f: adresse 7 bits d'une données en RAM (SFR ou GPR)
d: destination du résultat '0' ou 'W' → W
'1' ou 'f' → f (même adresse en RAM)

I – Instructions des PIC16
manipulation de variables RAM



I – Instructions des PIC16, exemples simples

1- Mettre 0 dans W

2- Charger la valeur 20h dans W

3- Charger la valeur de W à l'adresse 020h

4- Charger la valeur à l'adresse 020h dans W

5- Mettre à 1 le bit 5 de la valeur à l'adresse 003h

6- décrémenter la valeur à l'adresse 020h,
et stoker le résultat à la **même adresse**

7- décrémenter la valeur à l'adresse 020h,
et stoker le résultat dans W

8- décrémenter la valeur à l'adresse 020h,
et stoker le résultat à l'adresse 021h

II – Adressage

- PIC16: 3 types adressages des valeurs
  - Immédiat (*literal*)
    - Valeur (constante) passée à l'instruction
  - Direct
    - Adresse (constante) contenant la valeur
  - Indirect
    - Adresse d'une adresse contenant la valeur

II – Adressage immédiat

- Valeur (littérale) passée à l'instruction

➔ Bases des valeurs (reconnues par *mpasm*)

Type	Syntax	Example
Decimal	D'<digits>'	D'100' .100 d'100'
Hexadecimal	H'<hex_digits>' 0x<hex_digits>	H'9f' h'9f' 0x9f
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001'
ASCII	'<character>' A'<character>'	'C' A'C'

2b- Charger la valeur **20** dans W ?

II – Adressage direct

- Adresse contenant une valeur

➔ Utilisation d'équivalences

adresse ⇔ nom du registre

bit ⇔ nom du bit

- Les adresses de tous les registres SFR sont connues *implicitement:*

  movwf **STATUS** ⇔ movwf 0x003
- Les bits des registres SFR sont connus *implicitement*:

  bcf STATUS, **RP0** ⇔ bcf STATUS, 5
- Equivalence d'une adresse GPR possible:

  toto equ 0x020

  → movwf **toto** ⇔ movwf 0x020

II – Adressage indirect

- Adresse d'une adresse contenant une valeur
  - Utiliser pour les tableaux
  - Pointeur
  - *Accès rapide à une autre banque*

➔ PIC16 utilisation de 2 registres

FSR : adresse de la donnée

File Select Register

```
char a = 5;
char *FSR =&a;
```

INDF : valeur stockée à l'adresse contenu dans FSR

INDirect File

```
*FSR = 6;
```

INDF

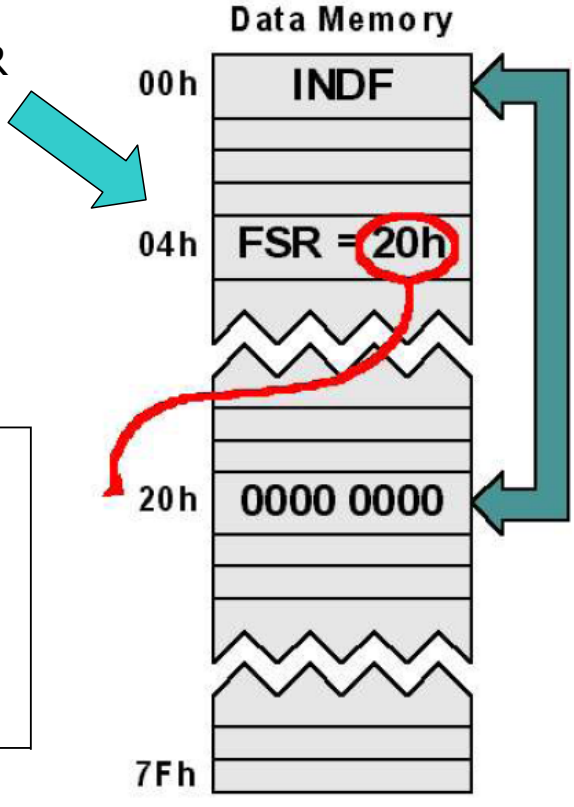
II – Adressage indirect, exemple

Exo: Déterminer INDF si 020h → FSR
INDF = ??

Exo: Faire la somme dans W des valeurs aux adresses 020h à 023h

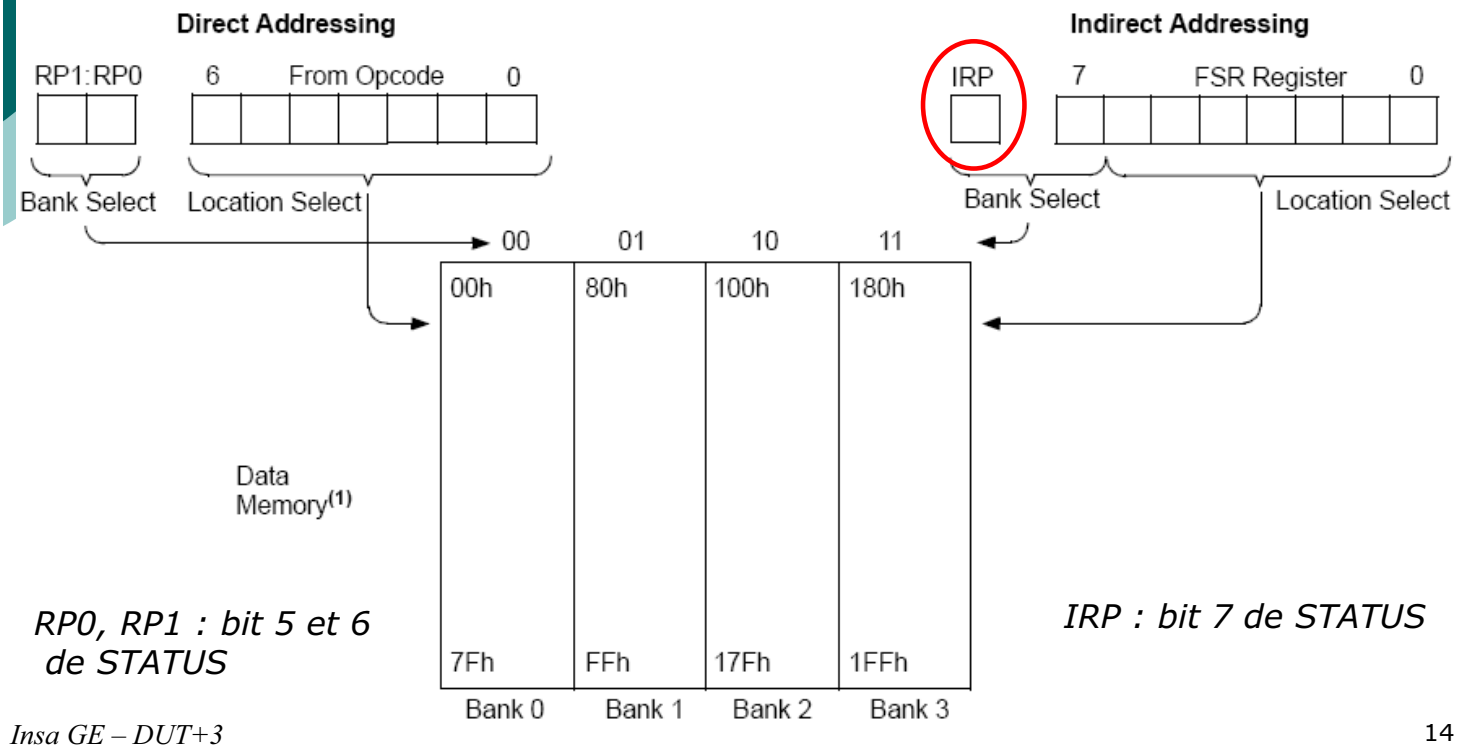
Exo: Que fait le programme suivant:

```
        MOVLW  0x20    ;initialize pointer
        MOVWF  FSR     ;to RAM
NEXT    CLRF   INDF    ;clear INDF register
        INCF   FSR,F   ;inc pointer
        BTFSS  FSR,4   ;all done?
        GOTO   NEXT    ;no clear next
CONTINUE
        :              ;yes continue
```



II – Adressage indirect, et les bank?

➔ FSR : 8 bits, bus d'adresse RAM : 9 bits



RP0, RP1 : bit 5 et 6 de STATUS

IRP : bit 7 de STATUS

III - Cycle instruction

- 1 cycle instruction (T_{cy}) est décomposé en 4 étapes Q1 – Q4
  - Q1: Décodage d'instruction (ou nop)
  - Q2: Lecture (ou nop)
  - Q3: Calcul
  - Q4: Ecriture (ou nop)
- période Q = période oscillateur (T_{osc})

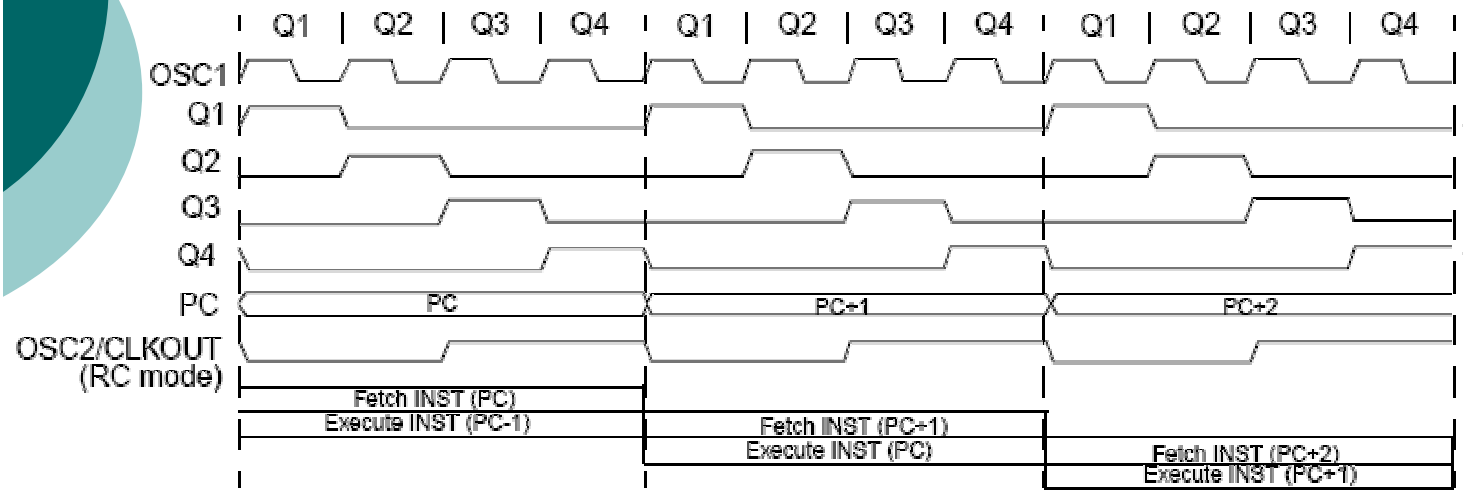
  ➔ $f_{cy} = f_{osc}/4$!!!

III - Cycle instruction

- Une instruction s'exécute entièrement en 2 cycles instructions
  - 1^{ier} T_{cy} : ***fetch*** (gestion de PC, chargement instruction dans Instruction Reg)
  - 2^{ième} T_{cy} : ***décodage et exécution***

- « **Pipelining** » : traitement en parallèle du *fetch* et du « *décodage-exécution* »!

III - Cycle instruction




```
1. MOVLW 55h
2. MOVWF PORTB
3. CALL  SUB_1
4. BSF   PORTA, BIT3
```

Fetch 1	Execute 1				
	Fetch 2	Execute 2			
		Fetch 3	Execute 3		
			Fetch 4	Flush	
				Fetch SUB_1	Execute SUB_1

IV – Les Instructions en détails

- Cf. document annexe

V – Accès aux mémoires

- RAM en lecture / écriture
  → Pas de problèmes
- EEPROM en lecture / écriture
- ROM en lecture / écriture ?

V – Accès à l'EEPROM et à la ROM

- Accès via des SFR
- EEPROM et ROM : même méthode
- 6 registres utilisés
  - EEDATAH:EEDATA → donnée
  - EEADRH:EEADR → adresse à lire/écrire
  - EECON1 → registre de contrôle et de paramétrage des accès
  - EECON2 → registre de contrôle en écriture

    (séquences de valeur pour l'écriture)

V – EECON1 *(adr : 18Ch)*

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit
1 = Accesses program memory
0 = Accesses data memory
Reads '0' after a POR; this bit cannot be changed while a write operation is in progress.

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **WRERR:** EEPROM Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{\text{MCLR}}$ or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read; RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

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

V – Lire en EEPROM

1. Ecrire dans EEADR l'adresse à lire
2. Choisir un accès à l'EEPROM
   0 → EECON1<EEPGD>
3. Démarrer la lecture
   1 → EECON1<RD>
4. Lire la valeur EEDATA

Code associé :

```
BSF     STATUS,RP1
BCF     STATUS,RP0
MOVF    DATA_EE_ADDR,W
MOVWF   EEADR
BSF     STATUS,RP0
BCF     EECON1,EEPGD

BSF     EECON1,RD
BCF     STATUS,RP0
MOVF    EEDATA,W
```


V – Ecrire en EEPROM *(sans IRQ)*

1. Vérifier qu'une écriture ne soit pas en cours
2. Choisir l'adresse EEPROM et la valeur à écrire
3. Choisir un accès à l'EEPROM
4. Autoriser l'écriture: 1→ EECON1<WREN>
5. Executer la séquence :
   55h → EECON2
   AAh → EECON2
   1 → EECON1<WR>
6. Interdire l'écriture: 0→ EECON1<WREN>

V – Ecrire en EEPROM, code

```
BSF    STATUS,RP1      ;
BSF    STATUS,RP0
BTFSC  EECON1,WR       ;Wait for write
GOTO   $-1             ;to complete
BCF    STATUS,  RP0    ;Bank 2
MOVF   DATA_EE_ADDR,W  ;Data Memory
MOVWF  EEADR           ;Address to write
MOVF   DATA_EE_DATA,W  ;Data Memory Value
MOVWF  EEDATA          ;to write
BSF    STATUS,RP0      ;Bank 3
BCF    EECON1,EEPGD    ;Point to DATA
                       ;memory
BSF    EECON1,WREN     ;Enable writes

BCF    INTCON,GIE      ;Disable INTs.
MOVLW  55h             ;
MOVWF  EECON2          ;Write 55h
MOVLW  AAh             ;
MOVWF  EECON2          ;Write AAh
BSF    EECON1,WR       ;Set WR bit to
                       ;begin write
BSF    INTCON,GIE      ;Enable INTs.
BCF    EECON1,WREN     ;Disable writes
```

Required Sequence

V – Lire et Ecrire en mémoire programme

- Possibilité de lire en ROM, 2 méthodes
- Possibilité d'écrire en ROM !

Cf. doc constructeurs

Plan

I. Généralités
II. Les microcontrôleurs PIC16
III. Jeu d'instructions des PIC16
IV. Ports d'entrées/sorties des PIC16
  I. Port d'E/S standards
  II. Port série
  III. Autres entrées/sorties
V. Fonctionnalités standards des PIC16
VI. Considérations techniques

Introduction

- Plusieurs types d' E/S
  - **Port E/S standards**
    - Lire / écrire un état logique sur une broche
    - Port parallèle
    - Port Parallèle esclave (PSP)
      Communication // commandé par une autre unité
  - Port E/S série
    - SSP et USART
  - Entrée pour détection d'événements
    - Interruptions
    - Compteurs
    - Capture/Compare/PWM
  - Entrées analogiques
    - Convertisseur A/N
    - Comparateurs de tension
  - Sorties
    - Capture/Compare/PWM
    - Générateur de tension

Introduction

- Remarque:

  Grande diversité des fonctions d'E/S

  \+ Beaucoup de possibilités d'interface

  mais Nombre de broches limité

➔Multiplexage des broches:

  - ➔ Chaque broche peut réaliser plusieurs fonctions
  - ➔ Jamais simultanément !!!
  - ➔ Besoin de configurer les interfaces et donc l'utilisation des broches
  - ➔ Attention aux compatibilités électriques

    (éviter double utilisation d'une même broche)

I – Port E/S *standards* (I/O Ports)

- Configuration par défaut du PIC
- Lire ou écrire un état logique sur une broche
- PIC 16F877 5 ports

  Nom du Registre (SFR)

  - Ports A: 6 bits RA0 à RA5 ➔ PORTA
  - Ports B: 8 bits RB0 à RB7 ➔ PORTB
  - Ports C: 8 bits RC0 à RC7 ➔ PORTC
  - Ports D: 8 bits RD0 à RD7 ➔ PORTD

    (non présent sur les boitiers 28 broches)
  - Ports E: 3 bits RE0 à RE2 ➔ PORTE

    (non présent sur les boitiers 28 broches)

I – Port E/S *standards* (I/O Ports)

- Bit de chaque port en lecture ou écriture
  - 3 états électriques possibles de la broche
    0L (0V) ou 1L (Vcc) → écriture
    Haute Impédance → lecture

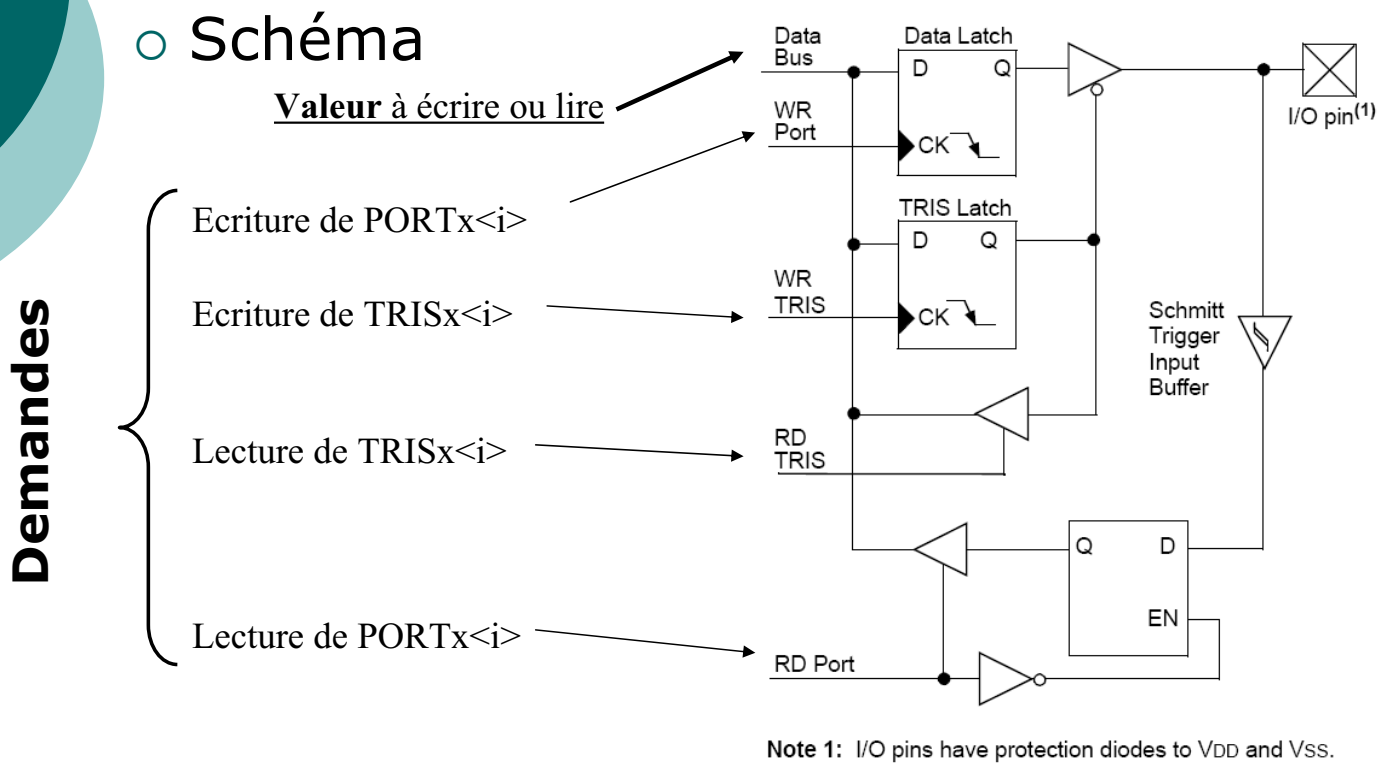
➔ **Choix de la direction de communication !!**

  - Rôle des registres SFR: TRISA, TRISB, …
  - L'état du bit i de TRISx (TRISx<i>) contrôle la direction du bit i de PORTx (PORTx<i>):
    TRISx<i> = 0 ➔ PORTx<i> en sortie (écriture)
    TRISx<i> = 1 ➔ PORTx<i> en entrée (lecture)

Exemple : TRISB = 0xFF ➔ les 8 bits du PORTB en entrées
TRISC = 0x0F ➔ RC7 à RC4 en sorties, RC3 à RC0 en entrées

I – Port E/S *standards* (I/O Ports)

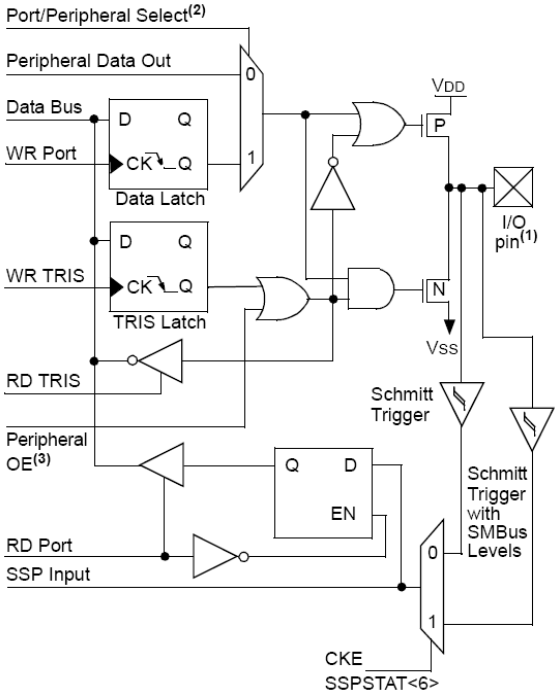
- Schéma



Note 1: I/O pins have protection diodes to VDD and VSS.

I – Port E/S *standards* (I/O Ports)

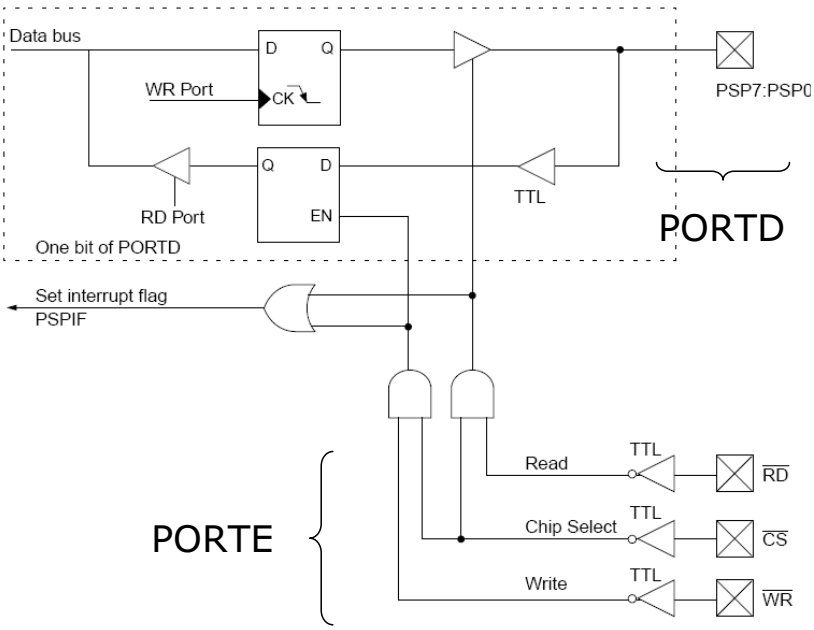
- Les broches des ports d'E/S sont multiplexées afin de réaliser plusieurs fonction



PORTC<4:3>
3 fonctions…

I – Port E/S *standards*
Parallel Slave Port

- Port parallèle *esclave*
  - ➔ Port (communication) piloté par un autre composant



Le port D est complètement piloté par le port E:
- accès au registre (R/W)
- configuration de la direction (entrée (HZ) / sortie (O-1))

Registre principal de configuration PSP: TRISE

II – Port série SSP

- SSP: *Synchronous Serial Port*
  - SSP = BSSP (B pour Basic)
  - ~ MSSP, M pour Master
- Communication avec périphériques série

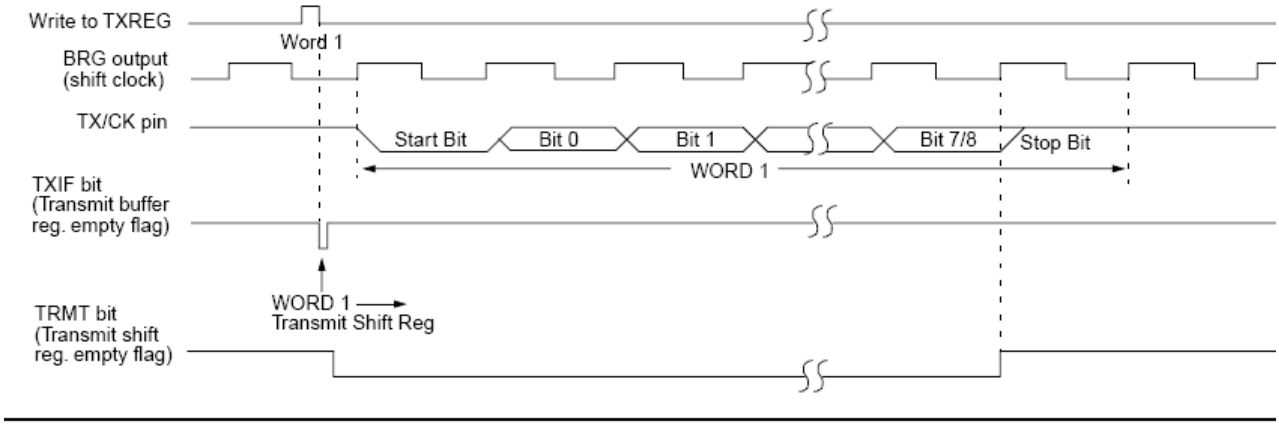
  EEPROM, registres à décalage, convertisseur A/D, afficheur, …
- SSP opère dans les modes
  - Serial Peripheral Interface (SPI™)
  - Inter-Integrated Circuit (I^2C™)
    - Mode esclave
    - Mode maitre ou multi maitre
- Registres de paramétrages principaux
  - SSPSTAT : status SSP
  - SSPCON : contrôle SSP

II – Port série USART

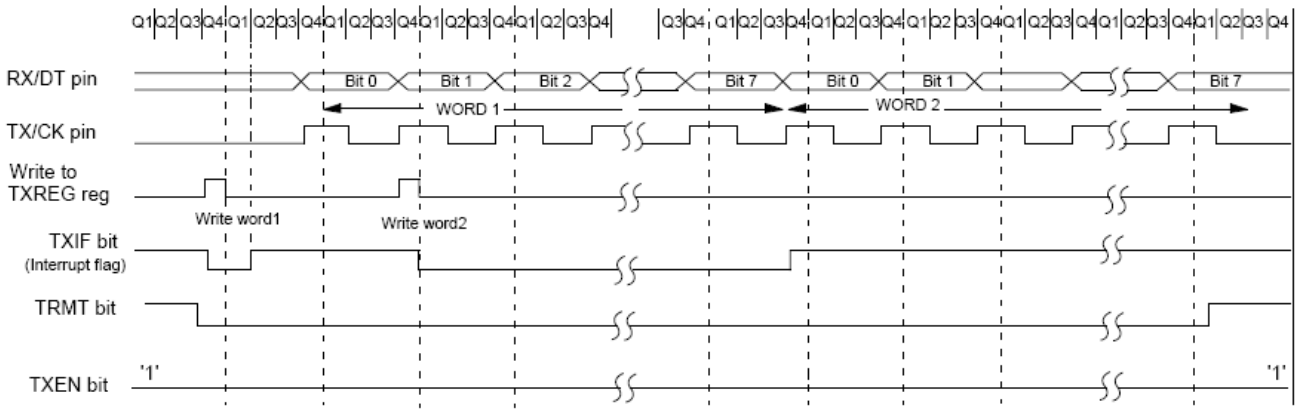
- USART: *Universal Synchronous Asynchronous Receiver Transmitter*
  - Autre nom: Serial Communication Interface (*SCI*)
- communication avec périphériques
  - Mode duplex : terminal, ordinateur personnel…
  - Mode half duplex: circuit A/D ou D/A, EEPROM série…
- 3 modes de fonctionnement
  - Asynchrone Full Duplex
  - Maitre Synchrone (Half duplex)
  - Esclave synchrone (Half duplex)
- Registres de paramétrages principaux
  - TXSTA : statu et contrôle transmission
  - RCSTA : statu et contrôle réception
  - SPBGR: configuration du *Baud rate*

II – Port série, USART

Transmission Asynchrone (Master)



Transmission Synchrone (Master)



III – Autres ports

- Entrée pour détection d'événements
  - Interruptions
  - Compteurs
  - Capture/Compare/PWM
- Entrées analogiques
  - Convertisseurs A/N
  - Comparateur de tension
- Sorties
  - Capture/Compare/PWM
  - Générateur de tension (convertisseur N/A)

III – Autres ports: entrées d'interruption ?

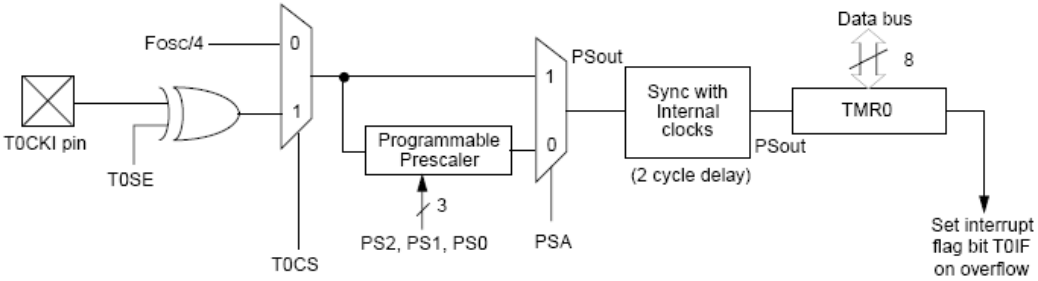
- **INT** : *même broche que RB0*
  - entrée d'interruption externe
- **RB7:RB4**
  - interruption si changement d'état d'un des 4 bits
- Comparator Change IRQ
  - IRQ si une tension dépasse un seuil
- CCP, SSP, USART…

III – Autres ports: compteur

- Permet de compter des impulsions arrivant sur des broches
- 2 compteurs d'impulsion
  - TIMER0 (8bits)
  - TIMER1 (16bits)
- Registres
  - OPTION_REG : configuration TIMER0
  - TMR0 : valeur du comptage du TIMER0
  - T1CON: configuration TIMER1
  - TMR1H:TMR1L : valeur du comptage du TIMER1

III – Autres ports: compteur

TIMER0 :



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).

OPTION_REG

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$ (1)	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$** (1): Weak Pull-up Enable bit
1 = Weak pull-ups are disabled
0 = Weak pull-ups are enabled by individual port latch values
➔ 0

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of INT pin
0 = Interrupt on falling edge of INT pin
choisir

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)
➔ 1

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin
choisir

III – Autres ports: capture/compare/PWM

- Deux modules CCP
  - CCP1 et CCP2
- Registres
  - CCPxCON : registre de contrôle et statu
  - CCPRxH:CCPRxL : valeurs 16 bits de comparaison
  - CCPx : broche utilisée
  - Les registres liés au TIMER1 et TIMER2…

III – Autres ports: capture/compare/PWM

- Capture (16 bits)
  - mémorisation de la valeur du TIMER1 (cf. prochain cours) à l'apparition d'un front sur une entrée CCPx
- Compare (16 bits)
  - une valeur constante est comparée avec celle de TIMER1 (qui évolue)
  - quand une égalité apparaît une sortie CCPx peut passer à 0L ou 1L, ou rester inchangée
- PWM (max. 10 bits)
  - création d'une impulsion de rapport cyclique variable (fréquence constante)
  - il s'agit d'une sortie PWM
  - La PWM est liée au TIMER2

III – Autres ports: convertisseur A/N

- CAN: charge d'une capacité + conversion A/N
  - Temps d'acquisition (T_{ACQ}) + temps conversion !!

    ~12.10^{-6} s composants du circuit + $9,5*T_{AD}$

    T_{AD}: temps conversion/bit

    $T_{ADmin}=1,6.10^{-6}$s

Par programme de conversion:

➔ Configurer la conversion A/N

➔ Attendre T_{ACQ}

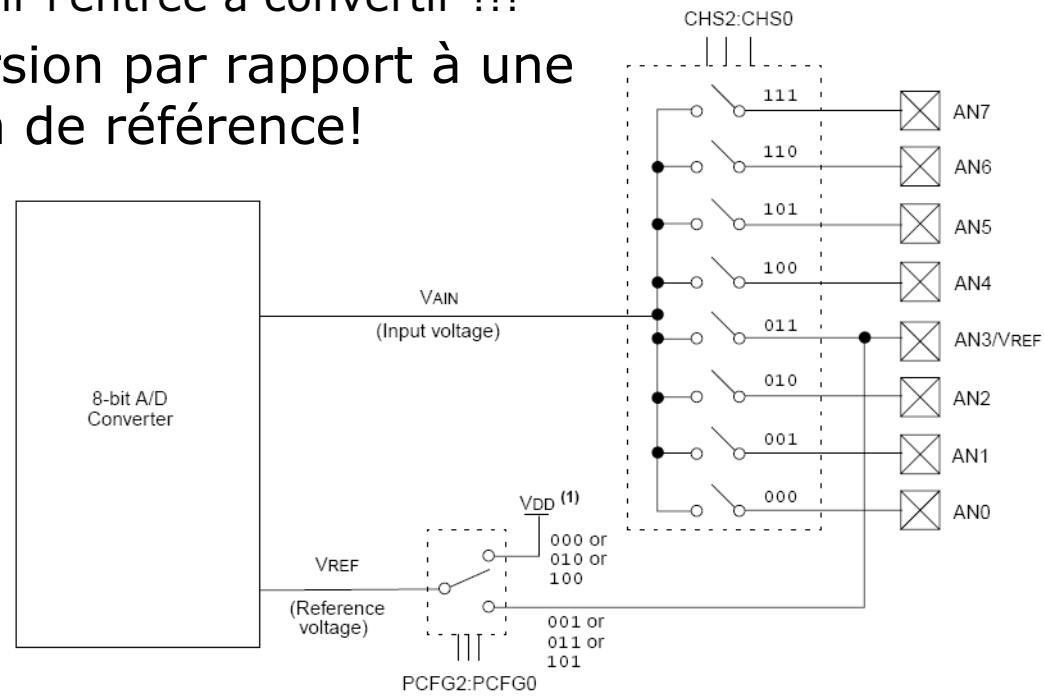
➔ Lancer la conversion

➔ Attendre conversion finie

➔ Lire la valeur convertie

III – Autres ports: convertisseur A/N

- 8 Entrées analogiques, 1 module de conversion:
  - Choisir l'entrée à convertir !!!
- Conversion par rapport à une tension de référence!



III – Autres ports: convertisseur A/N

- Registres
  - ADRES : résultat de la conversion A/N (8bits)
  - ADCON0, ADCON1 : registres de configuration de la conversion

ADCON0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	Resv	ADON
bit 7							bit 0

ADCON1

U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	PCFG2	PCFG1	PCFG0
bit 7							bit 0

III – Autres ports: comparateur de tension

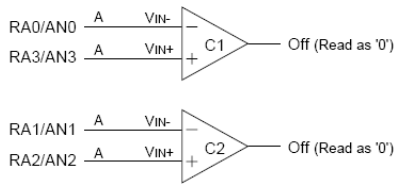
- 2 comparateurs analogiques disponibles
  - 2*2 entrées analogiques
- Les 2 comparateurs sont indépendants
- Possibilité de générer une tension de référence par le pic
- 8 possibilités d'utilisation des 2 comparateurs
  - Désactivé, indépendants, référence commune, multiplexage des entrées…
- Registre
  - CMCON : configuration et états des comparateurs

III – Autres ports: comparateur de tension

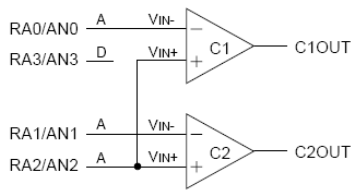
CMCON

R-0	R-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	—	—	CIS	CM2	CM1	CM0
bit 7							bit 0

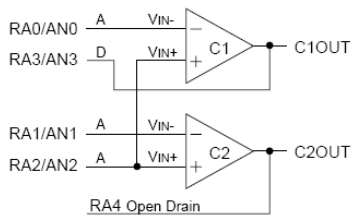
CM2:CM0 = 000
Comparators Reset (POR Default Value)



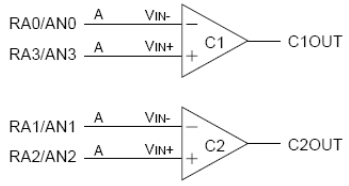
CM2:CM0 = 011
Two Common Reference Comparators



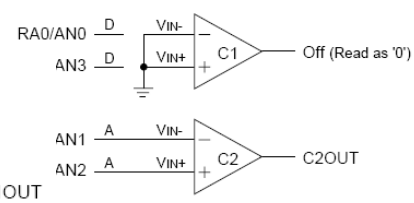
CM2:CM0 = 110
Two Common Reference Comparators with Outputs



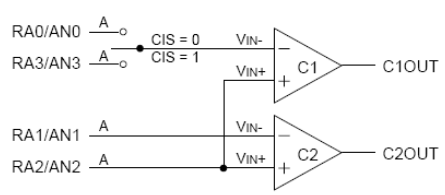
CM2:CM0 = 100
Two Independent Comparators



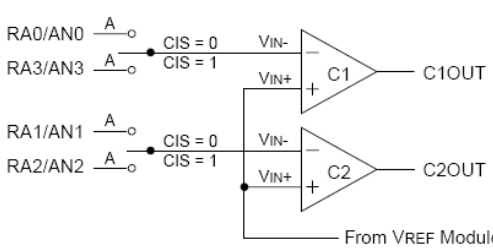
CM2:CM0 = 101
One Independent Comparator



CM2:CM0 = 001
Three Inputs Multiplexed to Two Comparators



CM2:CM0 = 010
Four Inputs Multiplexed to Two Comparators



III – Autres ports: générateur de tension

- équivalent à un convertisseur N/A 4 bits

Register 19-1: VRCON Register

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
VREN	VROE	VRR	—	VR3	VR2	VR1	VR0
bit 7							bit 0

bit 7 **VREN:** VREF Enable
1 = VREF circuit powered on
0 = VREF circuit powered down

bit 6 **VROE:** VREF Output Enable
1 = VREF is internally connected to Comparator module's VREF. This voltage level is also output on the VREF pin
0 = VREF is not connected to the comparator module. This voltage is disconnected from the VREF pin

bit 5 **VRR:** VREF Range selection
1 = 0V to 0.75 VDD, with VDD/24 step size
0 = 0.25 VDD to 0.75 VDD, with VDD/32 step size

bit 4 **Unimplemented:** Read as '0'

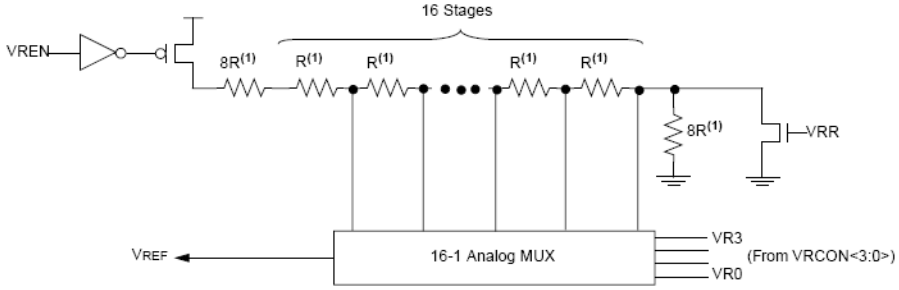
bit 3:0 **VR3:VR0:** VREF value selection 0 ≤ VR3:VR0 ≤ 15

When VRR = 1:

$V_{REF} = (VR<3:0>/ 24) \bullet V_{DD}$

When VRR = 0:

$V_{REF} = 1/4 * V_{DD} + (VR3:VR0/ 32) \bullet V_{DD}$



Plan

I. Généralités
II. Les microcontrôleurs PIC16
III. Jeu d'instructions des PIC16
IV. Ports d'entrées/sorties des PIC16
V. Fonctionnalités standards des PIC16
   I. Compteur/Timer
   II. IRQ
   III. (WatchDog)…
VI. Considérations techniques

I – Compteur / Timer

- Rôle d'un timer : compter !
  - Compter un nombre d'impulsions
    - externes
    - internes (basées sur f_{osc})
  - Intérêt principal
    - l'incrémentation est automatique
  - Utilité
    - Comptage automatique
    - Temporisation (très précise)

I – Compteur / Timer

- 3 TIMERs disponibles sur PIC16F877
  - TIMER0 : 8 bits
  - TIMER1 : 16 bits
  - TIMER2 : 8 bits (uniquement sur f_{osc})
- Paramétrage des TIMERs
  - **TIMER0** : OPTION_REG, INTCON<2,5>, *TMR0*
  - **TIMER1** : T1CON, PIR<TMR1IF>, PIE<TMR1IE>, *TMR1H:TMR1L*
  - **TIMER2** : T2CON, PIR<TMR2IF>, PIE<TMR2IE>, *TMR2, PR2 (équivalent à 16bits)*

I – Compteur / Timer

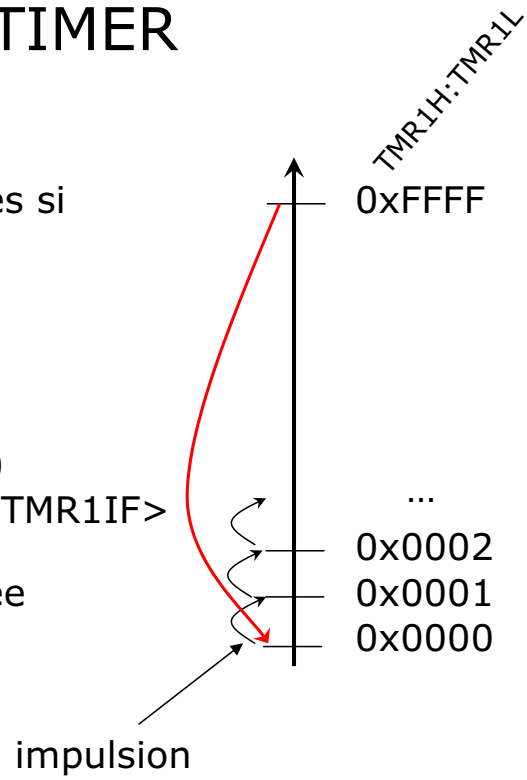
- Fonctionnement d'un TIMER
  exemple du TIMER1

→ Les impulsions sont comptées si T1CON<TMR1ON> = 1

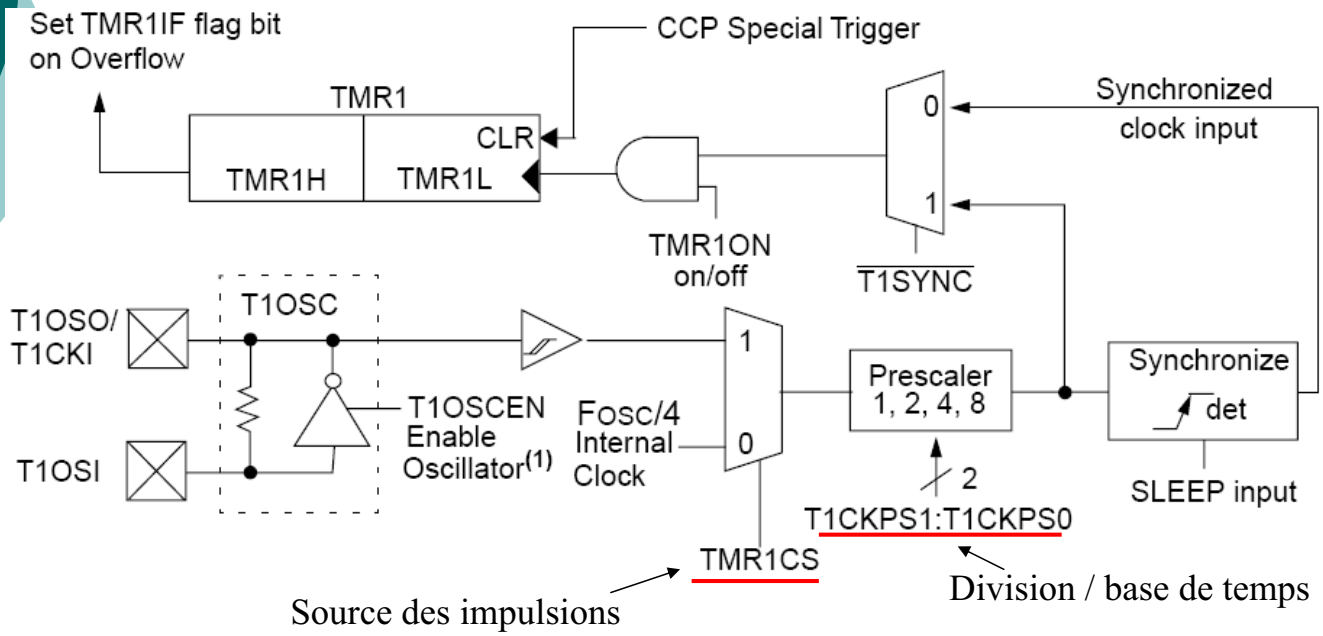
→ Le TIMER1 est bloqué si T1CON<TMR1ON> = 0

→ Au retour à 0x0000
mise à 1 du bit PIR1<TMR1IF>

→La valeur du comptage est stockée dans TMR1H:TMR1L
(accessible en lecture et écriture)



I – Compteur / Timer
Paramétrage, ex TIMER1



Note 1: When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

I – Compteur / Timer, reg T1CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 5:4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable bit

1 = Oscillator is enabled
0 = Oscillator is shut off. The oscillator inverter and feedback resistor are turned off to eliminate power drain

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Select bit

When TMR1CS = 1:

1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR1CS = 0:

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit

1 = External clock from pin T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON**: Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

I – Compteur / Timer
exemple de paramétrage

- Réaliser une temporisation d'une milliseconde avec le TIMER1
  (basée sur Fosc=20MHz)

→ Utilisation de Fosc : 0 → TMR1CS

Fosc / 4 = 5MHz

→ Choix d'un prescale : 2 ➔ T1CKPS1:T1CKPS0 = 01

(Fosc/4) / **2** = 2500kHz

Donc 2500 incrémentations = 1 ms

→ Valeur de départ de TMR1? :65536 - 2500 = 63036

63036 ➔ 0xF63C

0xF6 → TMR1H 0x3C → TMR1L

I – Compteur / Timer exemple temporisation 1ms

```
; INIT compteur
        bsf     STATUS, RP0 ; Bank1
        clrf    PIE1 ; Disable peripheral interrupts
        bcf     STATUS, RP0 ; Bank0
        clrf    PIR1 ; Clear peripheral interrupts Flags
        movlw   0x10 ; Internal Clock source with 1:2 prescaler,
        movwf   T1CON ;   Timer1 is stopped and T1 osc is disabled
        movlw   0xF6
        movwf   TMR1H   ;
        movlw   0x3C
        movwf   TMR1L   ;
        bsf     T1CON, TMR1ON ; Timer1 starts to increment

Tempo_Wms_OVFL_WAIT:
        btfss   PIR1, TMR1IF
        goto    Tempo_Wms_OVFL_WAIT

; Timer has overflowed
        bcf     PIR1, TMR1IF
        bcf     T1CON, TMR1ON ; Timer1 stops to increment
```


II – Interruption (*IRQ*), définition

rapide

- Interruption
  - événement produisant l'interruption du programme en cours d'exécution pour exécuter une routine d'interruption
  - Une interruption provient d'une source d'interruption et peut se produire n'importe quand!
  - Une source d'interruption peut être ignorée (par configuration)
- Une routine d'interruption doit
  1) sauvegarder le contexte (valeur des registres)
  2) prendre en charge la demande d'interruption
  3) la traiter (le + rapidement possible)
  4) Restaurer le contexte
  5) Retour au programme
     (instruction RETFIE pour les PIC, RETI en général)

II – Interruption (*IRQ*), PIC?

- PIC16 :
  - 1 vecteur reset 0x0000→ lancement programme
  - **1** seul vecteur d'interruption 0x0004
    → une seule routine d'interruption
- PIC 16 :
  - ~15 sources d'interruption

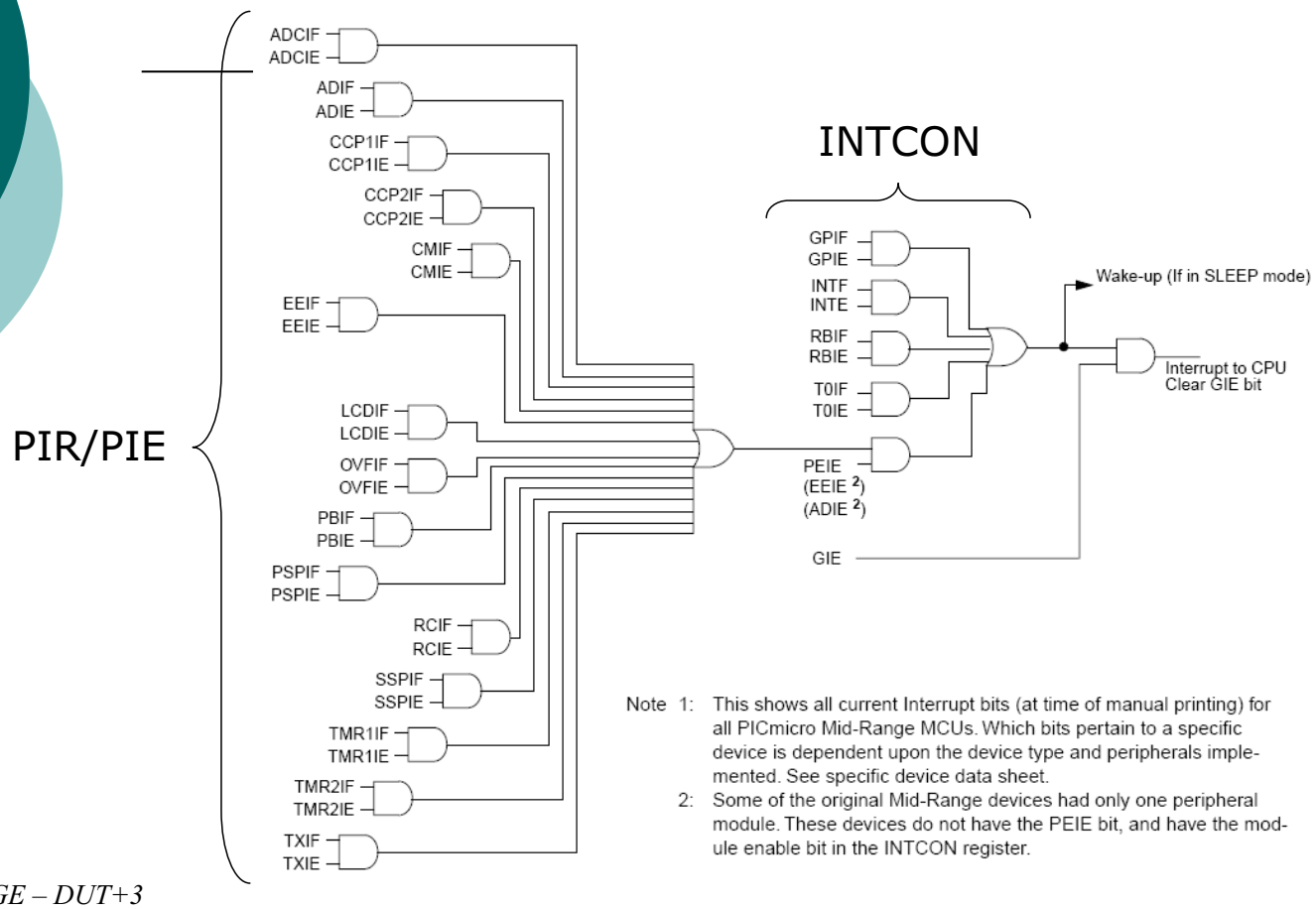
➔ LA routine d'interruption doit trouver quelle est la sources (tester les registres et trouver la source)

- Registres
  - INTCON: registre principal
  - PIR1, PIR2 : registres des IRQ des sources

(1bit ou *flag* par source)

  - PIE1, PIE2 : registres d'activation des IRQ pour les sources

II – Interruption (*IRQ*)



Note 1: This shows all current Interrupt bits (at time of manual printing) for all PICmicro Mid-Range MCUs. Which bits pertain to a specific device is dependent upon the device type and peripherals implemented. See specific device data sheet.

2: Some of the original Mid-Range devices had only one peripheral module. These devices do not have the PEIE bit, and have the module enable bit in the INTCON register.

II – Interruption (*IRQ*)

Détails des registres associés:

INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x
PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0

PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0

II – Interruption (*IRQ*)

INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

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

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

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

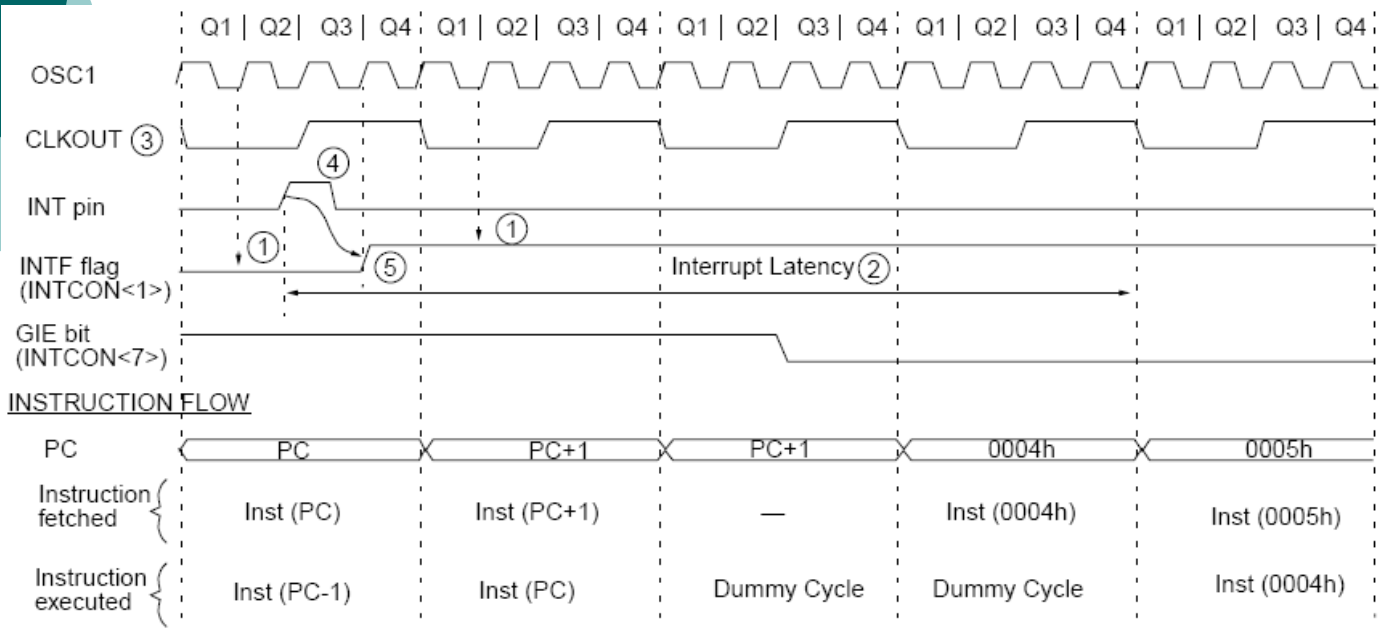
bit 0 **RBIF** (1): RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

II – Interruption (*IRQ*)

Sauvegarde et restauration du contexte dans l'ISR:

```
MOVWF     W_TEMP            ;Copy W to TEMP register
SWAPF     STATUS,W          ;Swap status to be saved into W
CLRF      STATUS            ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
MOVWF     STATUS_TEMP       ;Save status to bank zero STATUS_TEMP register
MOVF      PCLATH, W         ;Only required if using pages 1, 2 and/or 3
MOVWF     PCLATH_TEMP       ;Save PCLATH into W
CLRF      PCLATH            ;Page zero, regardless of current page
:
:(ISR)                      ;(Insert user code here)
:
MOVF      PCLATH_TEMP, W    ;Restore PCLATH
MOVWF     PCLATH            ;Move W into PCLATH
SWAPF     STATUS_TEMP,W     ;Swap STATUS_TEMP register into W
                            ;(sets bank to original state)
MOVWF     STATUS            ;Move W into STATUS register
SWAPF     W_TEMP,F          ;Swap W_TEMP
SWAPF     W_TEMP,W          ;Swap W_TEMP into W
```


II – Interruption (*IRQ*) temps de latence



Plan

I. Généralités
II. Les microcontrôleurs PIC16
III. Jeu d'instructions des PIC16
IV. Ports d'entrées/sorties des PIC16
V. Fonctionnalités standards des PIC16
VI. Considérations techniques
   I. Alimentation et protection électrique
   II. Oscillateur
   III. Cycle instruction et pipeline
   IV. Solutions de développement

I – Alimentation et protection électrique

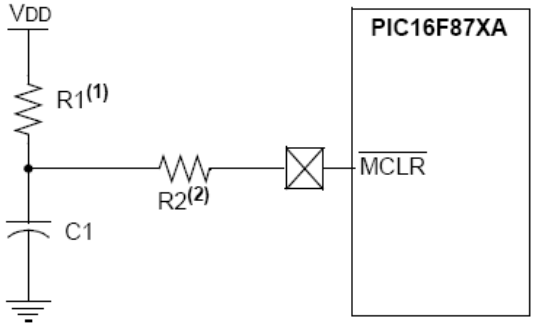
Voltage on any pin with respect to Vss (except VDD, $\overline{\text{MCLR}}$, and RA4)	-0.3V to (VDD + 0.3V)
Voltage on VDD with respect to Vss	-0.3 to +7.5V
Voltage on $\overline{\text{MCLR}}$ with respect to Vss **(Note 2)**	0 to +14V
Voltage on RA4 with respect to Vss	0 to +8.5V
Total power dissipation **(Note 1)**	1.0W
Maximum current out of Vss pin	300 mA
Maximum current into VDD pin	250 mA
Input clamp current, IIK (VI < 0 or VI > VDD)	± 20 mA
Output clamp current, IOK (VO < 0 or VO > VDD)	± 20 mA
Maximum output current sunk by any I/O pin	25 mA
Maximum output current sourced by any I/O pin	25 mA
Maximum current sunk by PORTA, PORTB and PORTE (combined) **(Note 3)**	200 mA
Maximum current sourced by PORTA, PORTB and PORTE (combined) **(Note 3)**	200 mA
Maximum current sunk by PORTC and PORTD (combined) **(Note 3)**	200 mA
Maximum current sourced by PORTC and PORTD (combined) **(Note 3)**	200 mA

Note 1: Power dissipation is calculated as follows: Pdis = VDD x {IDD - ∑ IOH} + ∑ {(VDD - VOH) x IOH} + ∑(VOl x IOL)

2: Voltage spikes below Vss at the $\overline{\text{MCLR}}$ pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{\text{MCLR}}$ pin rather than pulling this pin directly to Vss.

I – Alimentation et protection électrique

- Circuit RESET recommandé



Note 1: R1 < 40 kΩ is recommended to make sure that the voltage drop across R does not violate the device's electrical specification.

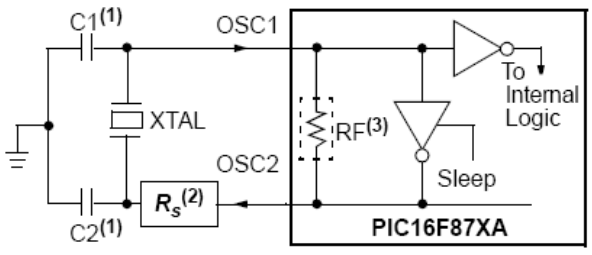
2: R2 > than 1K will limit any current flowing into MCLR from the external capacitor C, in the event of MCLR/VPP breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

Temps nécessaire au PIC pour démarrer après un reset:
moins de 132ms
Temps minimal d'un reset:
2ms

II – Oscillateur

- 4 modes d'oscillateurs disponibles:
  - *RC : Resistance/capacité*
  - LP : Low Power Crystal
  - XT : Crystal/resonator
  - HS : High Speed Crystal/resonator

Mode : LP, XT, HS



Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

II – Oscillateur

- Il est nécessaire de configurer le PIC en fonction du mode d'oscillateur utilisé!
  - Adresse 0x2007 en mémoire programme

(zone « bits de configuration »)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	$\overline{\text{PWRTEN}}$	WDTEN	Fosc1	Fosc0
bit 13													bit0

bit 1-0 **Fosc1:Fosc0**: Oscillator Selection bits

11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

III - Cycle instruction et pipeline

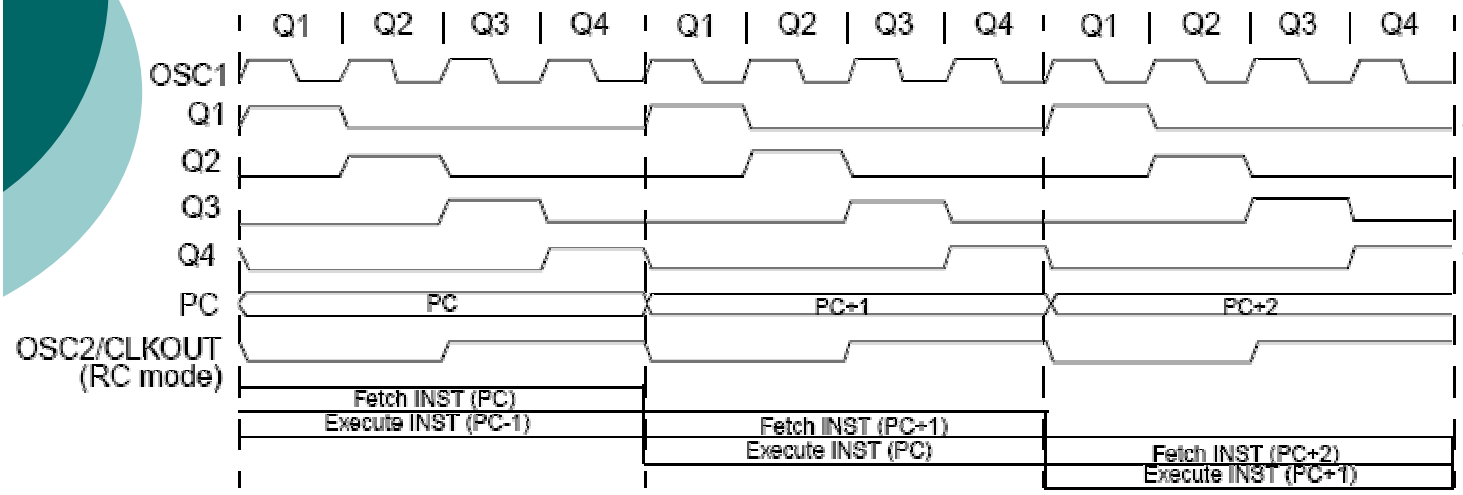
- 1 cycle instruction (T_{cy}) est décomposé en 4 étapes Q1 – Q4
  - Q1: Décodage d'instruction (ou nop)
  - Q2: Lecture (ou nop)
  - Q3: Calcul
  - Q4: Ecriture (ou nop)
- période Q = période oscillateur (T_{osc})

➔ $f_{cy} = f_{osc}/4$!!!

III - Cycle instruction et pipeline

- Une instruction s'exécute entièrement en 2 cycles instructions
  - 1^{ier} T_{cy} : ***fetch*** (gestion de PC, chargement instruction dans Instruction Reg)
  - 2^{ième} T_{cy} : ***décodage et exécution***
- « **Pipelining** » : traitement en parallèle du *fetch* et du « *décodage-exécution* »!

III - Cycle instruction et pipeline




```
1. MOVLW 55h
2. MOVWF PORTB
3. CALL  SUB_1
4. BSF   PORTA, BIT3
```

Fetch 1	Execute 1				
	Fetch 2	Execute 2			
		Fetch 3	Execute 3		
			Fetch 4	Flush	
				Fetch SUB_1	Execute SUB_1

III - Cycle instruction supplémentaire

- Annulation du cycle *fetch*
  - GOTO, CALL, RET, RETLW et RETFIE
  - Exécution d'IRQ
- Ajout de cycles internes
  - Modification des valeurs TIMER (TMR0, TMR1H:TMR1L,…)
    → 2 cycles non comptés par le TIMER

IV – Solutions de développement

- Compilateurs
  - Assembleur: MPASM
  - Langage C :MPLAB-C
  - Éditeur de lien: MPLINK
- Outils
  - Simulateur : MPLAB-SIM
  - Emulateur : ICEPIC, PICMASTER,…
- → Outils développement intégré
  - → MPLAB,…
- Programmateur /débogueur
  - ICD, PICSTART, PRO MATE II, …

MICROCONTROLEUR, FAMILLE PIC16

Documentation technique

Thomas Grenier

Sommaire :

- **Organisation mémoire** **3**
- **Détails des registres** **4**
- **Registre STATUS** **8**

- **Adressage indirect** **9**

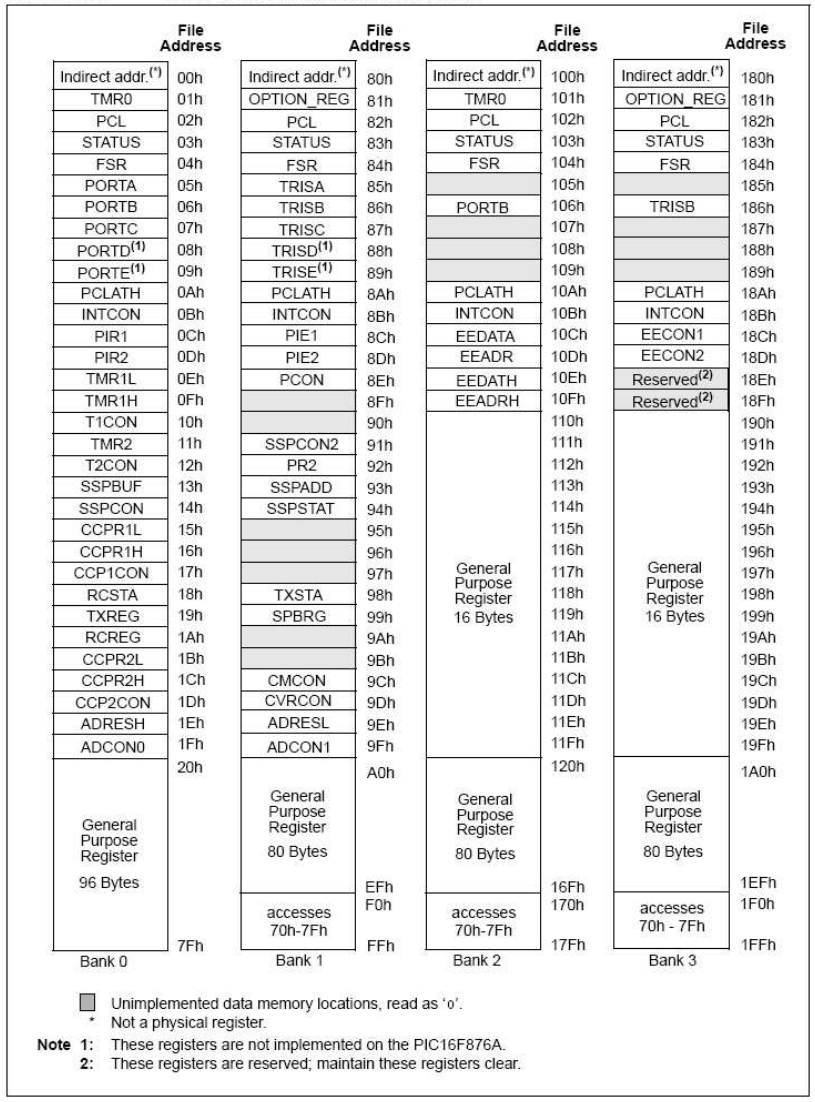
- **Description du PORTA** **10**
- **Description du PORTB** **11**
- **Description du PORTC** **13**
- **Description du PORTD** **14**

- **Description TIMER1** **15**

- **Interruptions : principes de base** **17**
- **Interruptions : détails de OPTION_REG** **19**
- **Interruptions : détails de INTCON** **20**
- **Interruptions : détails de PIE1** **21**
- **Interruptions : détails de PIR1** **21**

- **Jeu d'instructions** **21**

FIGURE 2-3: PIC16F876A/877A REGISTER FILE MAP



Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876A.
2: These registers are reserved; maintain these registers clear.

Figure 1 : Organisation mémoire

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
01h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	43, 150
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	47, 150
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	48, 150
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	49, 150
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
0Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	26, 150
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	28, 150
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	57, 150
11h	TMR2	Timer2 Module Register								0000 0000	62, 150
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	61, 150
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	79, 150
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	82, 82, 150
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	63, 150
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	63, 150
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	64, 150
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	112, 150
19h	TXREG	USART Transmit Data Register								0000 0000	118, 150
1Ah	RCREG	USART Receive Data Register								0000 0000	118, 150
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	63, 150
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	63, 150
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	64, 150
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	133, 150
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	127, 150

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.
5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

Figure 2 : Description des registres

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
101h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	30, 150
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
10Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	39, 151
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	39, 151
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						--xx xxxx	39, 151
10Fh	EEADRH	—	—	—	—(5)	EEPROM Address Register High Byte				---- xxxx	39, 151
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
18Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	34, 151
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	39, 151
18Eh	—	Reserved; maintain clear								0000 0000	—
18Fh	—	Reserved; maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

2.2.2.1 Status Register

The Status register contains the arithmetic status of the ALU, the Reset status and the bank select bits for data memory.

The Status register can be the destination for any instruction, as with any other register. If the Status register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable, therefore, the result of an instruction with the Status register as destination may be different than intended.

For example, `CLRF STATUS`, will clear the upper three bits and set the Z bit. This leaves the Status register as `000u uluu` (where `u` = unchanged).

It is recommended, therefore, that only `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions are used to alter the Status register because these instructions do not affect the Z, C or DC bits from the Status register. For other instructions not affecting any status bits, see **Section 15.0 "Instruction Set Summary"**.

Note:	The C and DC bits operate as a $\overline{\text{borrow}}$ and digit $\overline{\text{borrow}}$ bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

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

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

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

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

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

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

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

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `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

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

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

Figure 3 : Description du registre STATUS

2.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

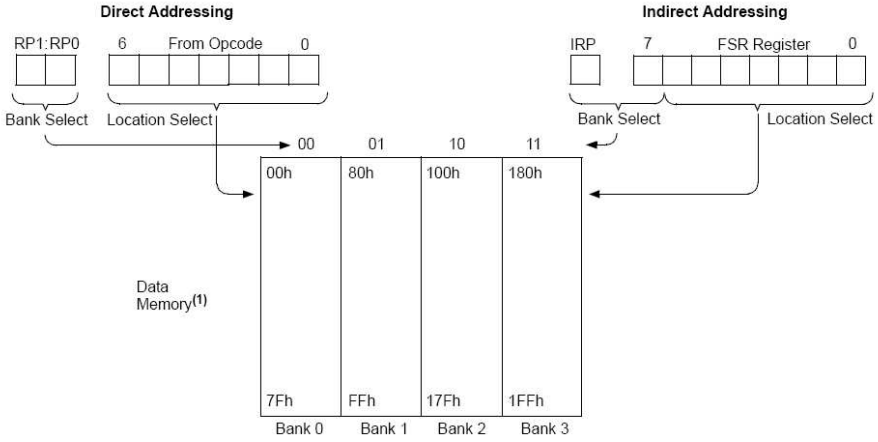
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = 0) will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (Status<7>) as shown in Figure 2-6.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: INDIRECT ADDRESSING

```
        MOVLW  0x20    ;initialize pointer
        MOVWF  FSR     ;to RAM
NEXT    CLRF   INDF    ;clear INDF register
        INCF   FSR,F   ;inc pointer
        BTFSS  FSR,4   ;all done?
        GOTO   NEXT    ;no clear next
CONTINUE
        :              ;yes continue
```

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING



Note 1: For register file map detail, see Figure 2-3.

Figure 4 : Adressage indirect

4.0 I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

Additional information on I/O ports may be found in the PICmicro™ Mid-Range Reference Manual (DS33023).

4.1 PORTA and the TRISA Register

PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, the value is modified and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open-drain output. All other PORTA pins have TTL input levels and full CMOS output drivers.

Other PORTA pins are multiplexed with analog inputs and the analog VREF input for both the A/D converters and the comparators. The operation of each pin is selected by clearing/setting the appropriate control bits in the ADCON1 and/or CMCON registers.

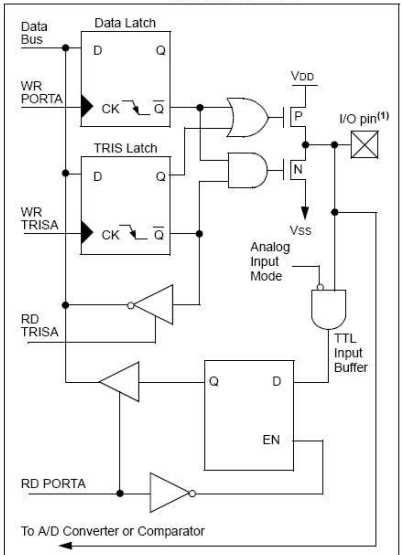
Note:	On a Power-on Reset, these pins are configured as analog inputs and read as '0'. The comparators are in the off (digital) state.

The TRISA register controls the direction of the port pins even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set when using them as analog inputs.

EXAMPLE 4-1: INITIALIZING PORTA

```
BCF     STATUS, RP0   ;
BCF     STATUS, RP1   ; Bank0
CLRF    PORTA         ; Initialize PORTA by
                      ; clearing output
                      ; data latches
BSF     STATUS, RP0   ; Select Bank 1
MOVLW   0x06          ; Configure all pins
MOVWF   ADCON1        ; as digital inputs
MOVLW   0xCF          ; Value used to
                      ; initialize data
                      ; direction
MOVWF   TRISA         ; Set RA<3:0> as inputs
                      ; RA<5:4> as outputs
                      ; TRISA<7:6>are always
                      ; read as '0'.
```

FIGURE 4-1: BLOCK DIAGRAM OF RA3:RA0 PINS



Note 1: I/O pins have protection diodes to VDD and VSS.

Figure 5 : Description du PORTA

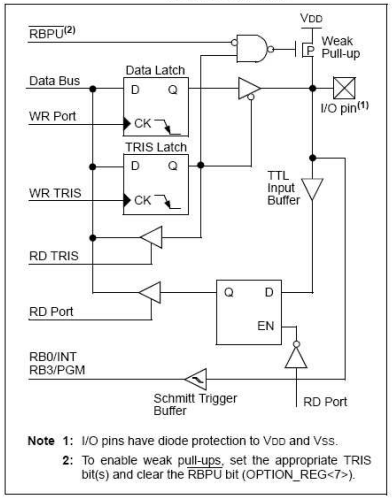
4.2 PORTB and the TRISB Register

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

Three pins of PORTB are multiplexed with the In-Circuit Debugger and Low-Voltage Programming function: RB3/PGM, RB6/PGC and RB7/PGD. The alternate functions of these pins are described in **Section 14.0 "Special Features of the CPU"**.

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (OPTION_REG<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

FIGURE 4-4: BLOCK DIAGRAM OF RB3:RB0 PINS



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Four of the PORTB pins, RB7:RB4, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are OR'ed together to generate the RB port change interrupt with flag bit RBIF (INTCON<0>).

This interrupt can wake the device from Sleep. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.

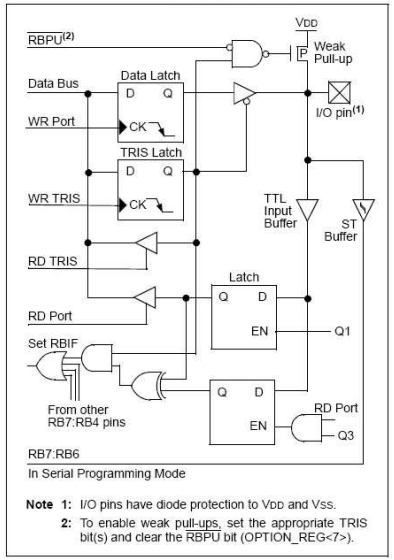
The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

This interrupt-on-mismatch feature, together with software configurable pull-ups on these four pins, allow easy interface to a keypad and make it possible for wake-up on key depression. Refer to the application note, *AN552*, *"Implementing Wake-up on Key Stroke"* (DS00552).

RB0/INT is an external interrupt input pin and is configured using the INTEDG bit (OPTION_REG<6>).

RB0/INT is discussed in detail in **Section 14.11.1 "INT Interrupt"**.

FIGURE 4-5: BLOCK DIAGRAM OF RB7:RB4 PINS



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Figure 6 : Description PORTB

TABLE 4-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT	bit 0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit 1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit 2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3/PGM(3)	bit 3	TTL	Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit 4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit 5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6/PGC	bit 6	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit 7	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode or in-circuit debugger.
3: Low-Voltage ICSP Programming (LVP) is enabled by default which disables the RB3 I/O function. LVP must be disabled to enable RB3 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

TABLE 4-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

4.3 PORTC and the TRISC Register

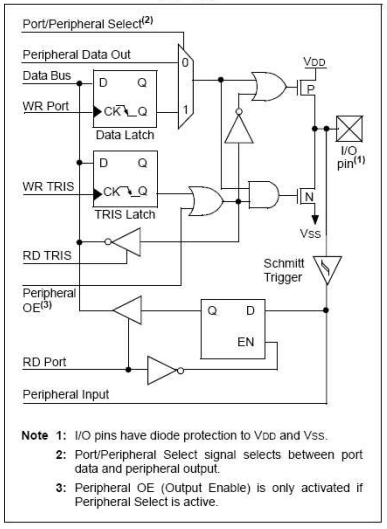
PORTC is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

PORTC is multiplexed with several peripheral functions (Table 4-5). PORTC pins have Schmitt Trigger input buffers.

When the I^2C module is enabled, the PORTC<4:3> pins can be configured with normal I^2C levels, or with SMBus levels, by using the CKE bit (SSPSTAT<6>).

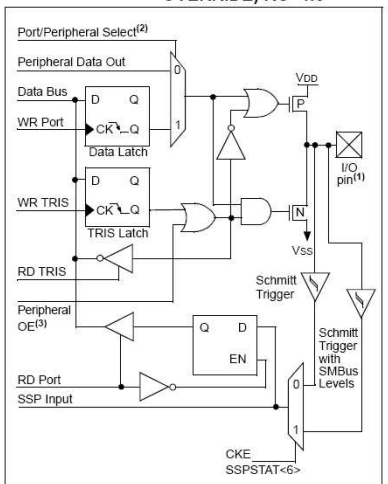
When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. Since the TRIS bit override is in effect while the peripheral is enabled, read-modify-write instructions (BSF, BCF, XORWF) with TRISC as the destination, should be avoided. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

FIGURE 4-6: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC<2:0>, RC<7:5>



Note 1: I/O pins have diode protection to VDD and VSS.
2: Port/Peripheral Select signal selects between port data and peripheral output.
3: Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

FIGURE 4-7: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC<4:3>



Note 1: I/O pins have diode protection to VDD and VSS.
2: Port/Peripheral Select signal selects between port data and peripheral output.
3: Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

Figure 7 : Description PORTC

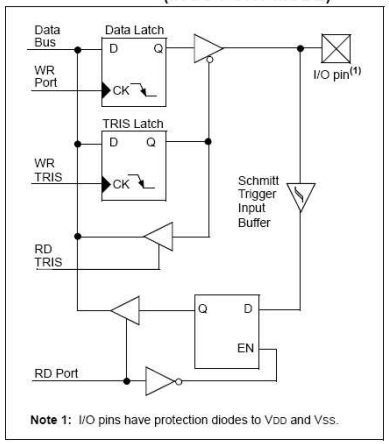
4.4 PORTD and TRISD Registers

Note: PORTD and TRISD are not implemented on the 28-pin devices.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

PORTD can be configured as an 8-bit wide microprocessor port (Parallel Slave Port) by setting control bit, PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

FIGURE 4-8: PORTD BLOCK DIAGRAM (IN I/O PORT MODE)



Note 1: I/O pins have protection diodes to VDD and VSS.

TABLE 4-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit 0	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 0.
RD1/PSP1	bit 1	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 1.
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 2.
RD3/PSP3	bit 3	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 3.
RD4/PSP4	bit 4	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 4.
RD5/PSP5	bit 5	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 5.
RD6/PSP6	bit 6	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 6.
RD7/PSP7	bit 7	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 7.

Legend: ST = Schmitt Trigger input, TTL = TTL input
Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 4-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

Figure 8 : Description PORTD

6.0 TIMER1 MODULE

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 interrupt, if enabled, is generated on overflow which is latched in interrupt flag bit, TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a Timer
- As a Counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit, TMR1ON (T1CON<0>).

Timer1 also has an internal "Reset input". This Reset can be generated by either of the two CCP modules (**Section 8.0 "Capture/Compare/PWM Modules"**). Register 6-1 shows the Timer1 Control register.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored and these pins read as '0'.

Additional information on timer modules is available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
11 = 1:8 prescale value
10 = 1:4 prescale value
01 = 1:2 prescale value
00 = 1:1 prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

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

Figure 9 : Description TIMER1

6.1 Timer1 Operation in Timer Mode

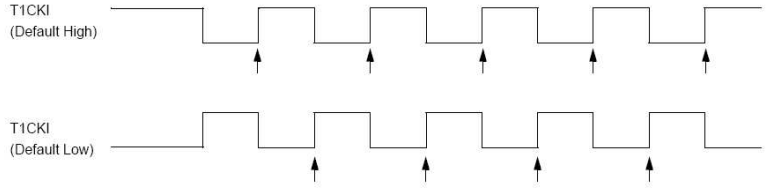
Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is Fosc/4. The synchronize control bit, $\overline{\text{T1SYNC}}$ (T1CON<2>), has no effect since the internal clock is always in sync.

6.2 Timer1 Counter Operation

Timer1 may operate in either a Synchronous, or an Asynchronous mode, depending on the setting of the TMR1CS bit.

When Timer1 is being incremented via an external source, increments occur on a rising edge. After Timer1 is enabled in Counter mode, the module must first have a falling edge before the counter begins to increment.

FIGURE 6-1: TIMER1 INCREMENTING EDGE



Note: Arrows indicate counter increments.

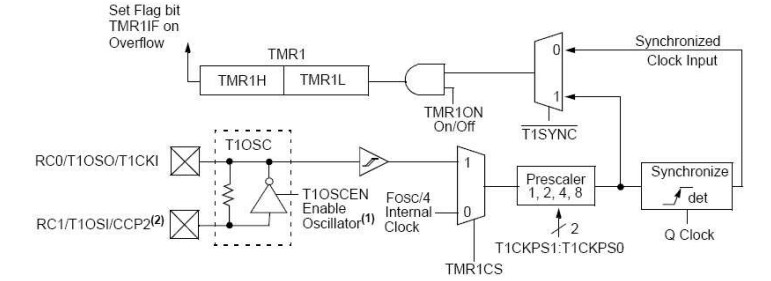
6.3 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR1CS. In this mode, the timer increments on every rising edge of clock input on pin RC1/T1OSI/CCP2 when bit T1OSCEN is set, or on pin RC0/T1OSO/T1CKI when bit T1OSCEN is cleared.

If $\overline{\text{T1SYNC}}$ is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler stage is an asynchronous ripple counter.

In this configuration, during Sleep mode, Timer1 will not increment even if the external clock is present since the synchronization circuit is shut-off. The prescaler, however, will continue to increment.

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

14.11 Interrupts

The PIC16F87XA family has up to 15 sources of interrupt. The Interrupt Control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits.

Note:	Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

A global interrupt enable bit, GIE (INTCON<7>), enables (if set) all unmasked interrupts or disables (if cleared) all interrupts. When bit GIE is enabled and an interrupt's flag bit and mask bit are set, the interrupt will vector immediately. Individual interrupts can be disabled through their corresponding enable bits in various registers. Individual interrupt bits are set regardless of the status of the GIE bit. The GIE bit is cleared on Reset.

The "return from interrupt" instruction, RETFIE, exits the interrupt routine, as well as sets the GIE bit, which re-enables interrupts.

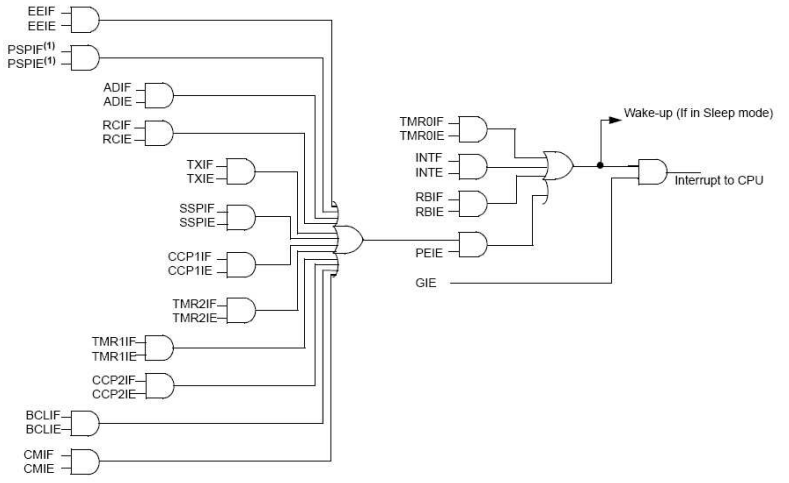
The RB0/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

The peripheral interrupt flags are contained in the Special Function Registers, PIR1 and PIR2. The corresponding interrupt enable bits are contained in Special Function Registers, PIE1 and PIE2, and the peripheral interrupt enable bit is contained in Special Function Register, INTCON.

When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

For external interrupt events, such as the INT pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set regardless of the status of their corresponding mask bit, PEIE bit or GIE bit.

FIGURE 14-10: INTERRUPT LOGIC



Note 1: PSP interrupt is implemented only on PIC16F874A/877A devices.

Figure 10 : Interruptions

14.11.1 INT INTERRUPT

External interrupt on the RB0/INT pin is edge triggered, either rising if bit INTEDG (OPTION_REG<6>) is set or falling if the INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, flag bit, INTF (INTCON<1>), is set. This interrupt can be disabled by clearing enable bit, INTE (INTCON<4>). Flag bit INTF must be cleared in software in the Interrupt Service Routine before re-enabling this interrupt. The INT interrupt can wake-up the processor from Sleep if bit INTE was set prior to going into Sleep. The status of global interrupt enable bit, GIE, decides whether or not the processor branches to the interrupt vector following wake-up. See **Section 14.14 "Power-down Mode (Sleep)"** for details on Sleep mode.

14.11.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in the TMR0 register will set flag bit, TMR0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit, TMR0IE (INTCON<5>). See **Section 5.0 "Timer0 Module"**.

14.11.3 PORTB INTCON CHANGE

An input change on PORTB<7:4> sets flag bit, RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit, RBIE (INTCON<4>). See **Section 4.2 "PORTB and the TRISB Register"**.

14.12 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt (i.e., W register and Status register). This will have to be implemented in software.

For the PIC16F873A/874A devices, the register W_TEMP must be defined in both Banks 0 and 1 and must be defined at the same offset from the bank base address (i.e., If W_TEMP is defined at 0x20 in Bank 0, it must also be defined at 0xA0 in Bank 1). The registers, PCLATH_TEMP and STATUS_TEMP, are only defined in Bank 0.

Since the upper 16 bytes of each bank are common in the PIC16F876A/877A devices, temporary holding registers, W_TEMP, STATUS_TEMP and PCLATH_TEMP, should be placed in here. These 16 locations don't require banking and therefore, make it easier for context save and restore. The same code shown in Example 14-1 can be used.

EXAMPLE 14-1: SAVING STATUS, W AND PCLATH REGISTERS IN RAM

```
    MOVWF   W_TEMP          ;Copy W to TEMP register
    SWAPF   STATUS,W        ;Swap status to be saved into W
    CLRF    STATUS          ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
    MOVWF   STATUS_TEMP     ;Save status to bank zero STATUS_TEMP register
    MOVF    PCLATH, W       ;Only required if using pages 1, 2 and/or 3
    MOVWF   PCLATH_TEMP     ;Save PCLATH into W
    CLRF    PCLATH          ;Page zero, regardless of current page
    :
    :(ISR)                  ;(Insert user code here)
    :
    MOVF    PCLATH_TEMP, W  ;Restore PCLATH
    MOVWF   PCLATH          ;Move W into PCLATH
    SWAPF   STATUS_TEMP,W   ;Swap STATUS_TEMP register into W
                            ;(sets bank to original state)
    MOVWF   STATUS          ;Move W into STATUS register
    SWAPF   W_TEMP,F        ;Swap W_TEMP
    SWAPF   W_TEMP,W        ;Swap W_TEMP into W
```

2.2.2.2 OPTION_REG Register

The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler/WDT postscaler (single assignable register known also as the prescaler), the external INT interrupt, TMR0 and the weak pull-ups on PORTB.

Note: To achieve a 1:1 prescaler assignment for the TMR0 register, assign the prescaler to the Watchdog Timer.

REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

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

Note: When using Low-Voltage ICSP Programming (LVP) and the pull-ups on PORTB are enabled, bit 3 in the TRISB register must be cleared to disable the pull-up on RB3 and ensure the proper operation of the device

Figure 11 :Détails registre OPTION_REG

2.2.2.3 INTCON Register

The INTCON register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow, RB port change and external RB0/INT pin interrupts.

Note: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

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

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT 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 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT 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; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

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

Figure 12 : Détails du registre INTCON

2.2.2.4 PIE1 Register

The PIE1 register contains the individual enable bits for the peripheral interrupts.

Note: Bit PEIE (INTCON<6>) must be set to enable any peripheral interrupt.

REGISTER 2-4: PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit(1)
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt
Note 1: PSPIE is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

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

Figure 13 : Détails du registre PIE1

2.2.2.5 PIR1 Register

The PIR1 register contains the individual flag bits for the peripheral interrupts.

Note: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt bits are clear prior to enabling an interrupt.

REGISTER 2-5: PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred
Note 1: PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag bit
1 = The SSP interrupt condition has occurred and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI – A transmission/reception has taken place.
- I^2C Slave – A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated Start condition was completed by the SSP module.
  - The initiated Stop condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A Start condition occurred while the SSP module was Idle (multi-master system).
  - A Stop condition occurred while the SSP module was Idle (multi-master system).

0 = No SSP interrupt condition has occurred

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

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

Figure 14: Détails du registre PIR1

15.0 INSTRUCTION SET SUMMARY

The PIC16 instruction set is highly orthogonal and is comprised of three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

Each PIC16 instruction is a 14-bit word divided into an **opcode** which specifies the instruction type and one or more **operands** which further specify the operation of the instruction. The formats for each of the categories is presented in Figure 15-1, while the various opcode fields are summarized in Table 15-1.

Table 15-2 lists the instructions recognized by the MPASM™ Assembler. A complete description of each instruction is also available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven-bit constant or literal value

One instruction cycle consists of four oscillator periods; for an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1 µs. All instructions are executed within a single instruction cycle, unless a conditional test is true, or the program counter is changed as a result of an instruction. When this occurs, the execution takes two instruction cycles with the second cycle executed as a NOP.

Note:	To maintain upward compatibility with future PIC16F87XA products, do not use the OPTION and TRIS instructions.

All instruction examples use the format '0xhh' to represent a hexadecimal number, where 'h' signifies a hexadecimal digit.

15.1 READ-MODIFY-WRITE OPERATIONS

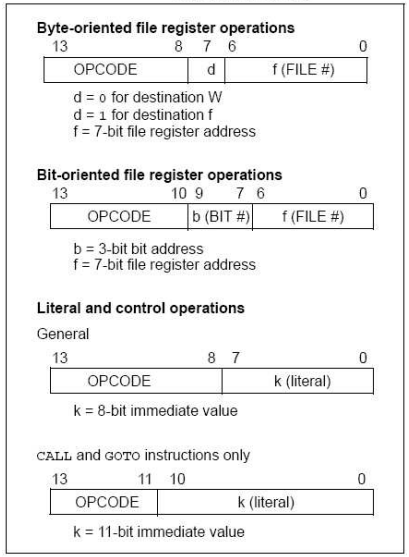
Any instruction that specifies a file register as part of the instruction performs a Read-Modify-Write (R-M-W) operation. The register is read, the data is modified, and the result is stored according to either the instruction or the destination designator 'd'. A read operation is performed on a register even if the instruction writes to that register.

For example, a "CLRF PORTB" instruction will read PORTB, clear all the data bits, then write the result back to PORTB. This example would have the unintended result that the condition that sets the RBIF flag would be cleared.

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1.
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS



Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF PORTB, 1), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figure 15: Jeu d'instructions des PIC 16

Section 29. Instruction Set

HIGHLIGHTS

This section of the manual contains the following major topics:

29.1 Introduction 29-2
29.2 Instruction Formats 29-4
29.3 Special Function Registers as Source/Destination 29-6
29.4 Q Cycle Activity 29-7
29.5 Instruction Descriptions 29-8
29.6 Design Tips 29-45
29.7 Related Application Notes 29-47
29.8 Revision History 29-48

29

Instruction Set

29.1 Introduction

Each midrange instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The midrange Instruction Set Summary in Table 29-1 lists the instructions recognized by the MPASM assembler. The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

Table 29-2 gives the opcode field descriptions.

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value.

All instructions are executed in one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In these cases, the execution takes two instruction cycles with the second cycle executed as an NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 29-1: Midrange Instruction Set

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

29.2 Instruction Formats

Figure 29-1 shows the three general formats that the instructions can have. As can be seen from the general format of the instructions, the opcode portion of the instruction word varies from 3-bits to 6-bits of information. This is what allows the midrange instruction set to have 35 instructions.

Note 1: Any unused opcode is Reserved. Use of any reserved opcode may cause unexpected operation.

Note 2: To maintain upward compatibility with future midrange products, do not use the `OPTION` and `TRIS` instructions.

All instruction examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

To represent a binary number:

`00000100b`

where b is a binary string identifier.

Figure 29-1: General Format for Instructions

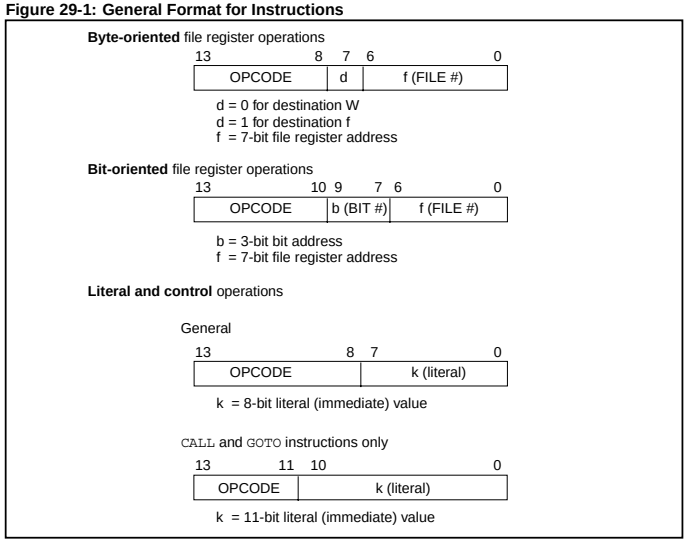


Table 29-2: Instruction Description Conventions

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register (0 to 7)
k	Literal field, constant data or label (may be either an 8-bit or an 11-bit value)
x	Don't care (0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f.
dest	Destination either the W register or the specified register file location
label	Label name
TOS	Top of Stack
PC	Program Counter
PCLATH	Program Counter High Latch
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer
$\overline{TO}$	Time-out bit
$\overline{PD}$	Power-down bit
[ ]	Optional
( )	Contents
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

29

Instruction Set

29.3 Special Function Registers as Source/Destination

The Section 29. Instruction Set's orthogonal instruction set allows read and write of all file registers, including special function registers. Some special situations the user should be aware of are explained in the following subsections:

29.3.1 STATUS Register as Destination

If an instruction writes to the STATUS register, the Z, C, DC and OV bits may be set or cleared as a result of the instruction and overwrite the original data bits written. For example, executing `CLRF STATUS` will clear register STATUS, and then set the Z bit leaving `0000 0100b` in the register.

29.3.2 PCL as Source or Destination

Read, write or read-modify-write on PCL may have the following results:

Read PC: PCL → dest; PCLATH does not change;

Write PCL: PCLATH → PCH;
8-bit destination value → PCL

Read-Modify-Write: PCL→ ALU operand
PCLATH → PCH;
8-bit result → PCL

Where PCH = program counter high byte (not an addressable register), PCLATH = Program counter high holding latch, dest = destination, W register or register file f.

29.3.3 Bit Manipulation

All bit manipulation instructions will first read the entire register, operate on the selected bit and then write the result back (read-modify-write (R-M-W)) the specified register. The user should keep this in mind when operating on some special function registers, such as ports.

Note: Status bits that are manipulated by the device (including the interrupt flag bits) are set or cleared in the Q1 cycle. So there is no issue with executing R-M-W instructions on registers which contain these bits.

29.4 Q Cycle Activity

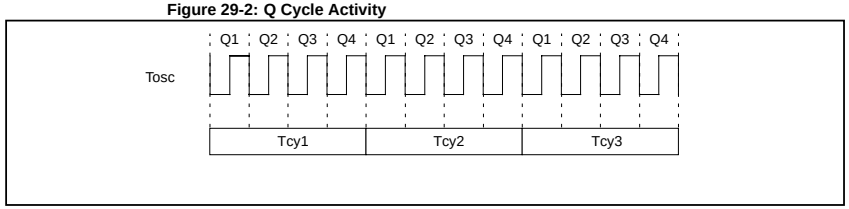
Each instruction cycle (Tcy) is comprised of four Q cycles (Q1-Q4). The Q cycle is the same as the device oscillator cycle (TOSC). The Q cycles provide the timing/designation for the Decode, Read, Process Data, Write etc., of each instruction cycle. The following diagram shows the relationship of the Q cycles to the instruction cycle.

The four Q cycles that make up an instruction cycle (Tcy) can be generalized as:

Q1: Instruction Decode Cycle or forced No Operation
Q2: Instruction Read Cycle or No Operation
Q3: Process the Data
Q4: Instruction Write Cycle or No Operation

Each instruction will show the detailed Q cycle operation for the instruction.

Figure 29-2: Q Cycle Activity



29 Instruction Set

29.5 Instruction Descriptions

ADDLW — Add Literal and W

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: $(W) + k \rightarrow W$

Status Affected: C, DC, Z

Encoding:

11	111x	kkkk	kkkk

Description: The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example1

```
ADDLW   0x15
```

Before Instruction
W = 0x10
After Instruction
W = 0x25

Example 2

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x47

Example 3

```
ADDLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0xA3

Example 4

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of PCL † = 0x02
† PCL is the symbol for the Program Counter low byte location
After Instruction
W = 0x12

ADDWF Add W and f

Syntax: [*label*] ADDWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (W) + (f) → destination

Status Affected: C, DC, Z

Encoding:

00	0111	dfff	ffff

Description: Add the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
ADDWF   FSR, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0xD9
FSR = 0xC2

Example 2

```
ADDWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37

Example 3

```
ADDWF   PCL
```

Case 1:
Before Instruction
W = 0x10
PCL = 0x37
C = x
After Instruction
PCL = 0x47
C = 0

Case 2:
Before Instruction
W = 0x10
PCL = 0xF7
PCH = 0x08
C = x
After Instruction
PCL = 0x07
PCH = 0x08
C = 1

ANDLW And Literal with W

Syntax: [*label*] ANDLW k

Operands: 0 ≤ k ≤ 255

Operation: (W).AND. (k) → W

Status Affected: Z

Encoding:

11	1001	kkkk	kkkk

Description: The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
ANDLW   0x5F
```

Before Instruction
W = 0xA3
After Instruction
W = 0x03

```
; 0101 1111  (0x5F)
; 1010 0011  (0xA3)
;---------  ------
; 0000 0011  (0x03)
```

Example 2

```
ANDLW   MYREG
```

Before Instruction
W = 0xA3
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x23

Example 3

```
ANDLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0xA3
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x83

ANDWF AND W with f

Syntax: [*label*] ANDWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (W).AND. (f) → destination

Status Affected: Z

Encoding:

00	0101	dfff	ffff

Description: AND the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
ANDWF   FSR, 1
```

Before Instruction
W = 0x17 ; 0001 0111 (0x17)
FSR = 0xC2 ; 1100 0010 (0xC2)
;---------- ------
After Instruction ; 0000 0010 (0x02)
W = 0x17
FSR = 0x02

Example 2

```
ANDWF   FSR, 0
```

Before Instruction ; 0001 0111 (0x17)
W = 0x17 ; 1100 0010 (0xC2)
FSR = 0xC2 ;---------- ------
After Instruction ; 0000 0010 (0x02)
W = 0x02
FSR = 0xC2

Example 3

```
ANDWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x5A
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x15

29

Instruction Set

BCF Bit Clear f

Syntax: [*label*] BCF f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b ≤ 7

Operation: 0 → f<b>

Status Affected: None

Encoding:

01	00bb	bfff	ffff

Description: Bit 'b' in register 'f' is cleared.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
BCF     FLAG_REG, 7
```

Before Instruction
FLAG_REG = 0xC7 ; 1100 0111
After Instruction
FLAG_REG = 0x47 ; 0100 0111

Example 2

```
BCF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x2F
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x27

BSF Bit Set f

Syntax: [*label*] BSF f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b ≤ 7

Operation: 1 → f<b>

Status Affected: None

Encoding:

01	01bb	bfff	ffff

Description: Bit 'b' in register 'f' is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
BSF     FLAG_REG, 7
```

Before Instruction
FLAG_REG =0x0A ; 0000 1010

After Instruction
FLAG_REG =0x8A ; 1000 1010

Example 2

```
BSF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20

After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x28

BTFSC Bit Test, Skip if Clear

Syntax: [*label*] BTFSC f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b ≤ 7

Operation: skip if (f<b>) = 0

Status Affected: None

Encoding:

01	10bb	bfff	ffff

Description: If bit 'b' in register 'f' is '0' then the next instruction is skipped.
If bit 'b' is '0' then the next instruction (fetched during the current instruction execution) is discarded, and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    BTFSC   FLAG, 4
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = address TRUE

Case 2: Before Instruction
PC = address HERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = address FALSE

BTFSS — Bit Test f, Skip if Set

Syntax: [*label*] BTFSS f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b < 7

Operation: skip if (f<b>) = 1

Status Affected: None

Encoding:

01	11bb	bfff	ffff

Description: If bit 'b' in register 'f' is '1' then the next instruction is skipped.
If bit 'b' is '1', then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    BTFSS   FLAG, 4
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = address FALSE

Case 2: Before Instruction
PC = address HERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = address TRUE

29

Instruction Set

CALL — Call Subroutine

Syntax: [*label*] CALL k

Operands: 0 ≤ k ≤ 2047

Operation: (PC)+ 1→ TOS,
k → PC<10:0>,
(PCLATH<4:3>) → PC<12:11>

Status Affected: None

Encoding:

10	0kkk	kkkk	kkkk

Description: Call Subroutine. First, the 13-bit return address (PC+1) is pushed onto the stack. The eleven bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH<4:3>. CALL is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    CALL    THERE
```

Before Instruction
PC = Address HERE
After Instruction
TOS = Address HERE+1
PC = Address THERE

CLRF Clear f

Syntax: [*label*] CLRF f

Operands: 0 ≤ f ≤ 127

Operation: 00h → f
1 → Z

Status Affected: Z

Encoding:

00	0001	1fff	ffff

Description: The contents of register 'f' are cleared and the Z bit is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
CLRF    FLAG_REG
```

Before Instruction
FLAG_REG=0x5A
After Instruction
FLAG_REG=0x00
Z = 1

Example 2

```
CLRF    INDF
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=0x00
Z = 1

CLRW Clear W

Syntax: [*label*] CLRW

Operands: None

Operation: 00h → W
1 → Z

Status Affected: Z

Encoding:

00	0001	0xxx	xxxx

Description: W register is cleared. Zero bit (Z) is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'W'

Example 1

```
CLRW
```

Before Instruction
W = 0x5A
After Instruction
W = 0x00
Z = 1

CLRWDT — Clear Watchdog Timer

Syntax: [*label*] CLRWDT

Operands: None

Operation: 00h → WDT
0 → WDT prescaler count,
1 → $\overline{TO}$
1 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0100

Description: CLRWDT instruction clears the Watchdog Timer. It also clears the prescaler count of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	Clear WDT Counter

Example 1

```
CLRWDT
```

Before Instruction
WDT counter= x
WDT prescaler =1:128
After Instruction
WDT counter=0x00
WDT prescaler count=0
$\overline{TO}$ = 1
$\overline{PD}$ = 1
WDT prescaler =1:128

Note: The CLRWDT instruction does not affect the assignment of the WDT prescaler.

COMF — Complement f

Syntax: [*label*] COMF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: ($\bar{f}$) → destination

Status Affected: Z

Encoding:

00	1001	dfff	ffff

Description: The contents of register 'f' are 1's complemented. If 'd' is 0 the result is stored in W. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
COMF    REG1, 0
```

Before Instruction
REG1= 0x13
After Instruction
REG1= 0x13
W = 0xEC

Example 2

```
COMF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=0x55

Example 3

```
COMF    REG1, 1
```

Before Instruction
REG1= 0xFF
After Instruction
REG1= 0x00
Z = 1

DECF — Decrement f

Syntax: [*label*] DECF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) - 1 → destination

Status Affected: Z

Encoding:

00	0011	dfff	ffff

Description: Decrement register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
DECF    CNT, 1
```

Before Instruction
CNT = 0x01
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2

```
DECF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x01
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
DECF    CNT, 0
```

Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x0F
Z = 0

DECFSZ — Decrement f, Skip if 0

Syntax: [*label*] DECFSZ f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) - 1 → destination; skip if result = 0

Status Affected: None

Encoding:

00	1011	dfff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
If the result is 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

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

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE      DECFSZ   CNT, 1
          GOTO     LOOP
CONTINUE  •
          •
          •
```

Case 1:
Before Instruction
PC = address HERE
CNT = 0x01
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2:
Before Instruction
PC = address HERE
CNT = 0x02
After Instruction
CNT = 0x01
PC = address HERE + 1

GOTO — Unconditional Branch

Syntax: [*label*] GOTO k

Operands: 0 ≤ k ≤ 2047

Operation: k → PC<10:0>
PCLATH<4:3> → PC<12:11>

Status Affected: None

Encoding:

10	1kkk	kkkk	kkkk

Description: GOTO is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
GOTO THERE
```

After Instruction
PC =Address THERE

INCF — Increment f

Syntax: [*label*] INCF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) + 1 → destination

Status Affected: Z

Encoding:

00	1010	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
INCF CNT, 1
```

Before Instruction
CNT = 0xFF
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2

```
INCF INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0xFF
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
INCF CNT, 0
```

Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x11
Z = 0

INCFSZ — Increment f, Skip if 0

Syntax: [*label*] INCFSZ f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) + 1 → destination, skip if result = 0

Status Affected: None

Encoding:

00	1111	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
If the result is 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

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

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE     INCFSZ   CNT, 1
         GOTO     LOOP
CONTINUE •
         •
         •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2: Before Instruction
PC = address HERE
CNT = 0x00
After Instruction
CNT = 0x01
PC = address HERE + 1

IORLW — Inclusive OR Literal with W

Syntax: [*label*] IORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W).OR. k → W

Status Affected: Z

Encoding:

11	1000	kkkk	kkkk

Description: The contents of the W register is OR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
IORLW   0x35
```

Before Instruction
W = 0x9A
After Instruction
W = 0xBF
Z = 0

Example 2

```
IORLW   MYREG
```

Before Instruction
W = 0x9A
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x9F
Z = 0

Example 3

```
IORLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x9A
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x9B
Z = 0

Example 4

```
IORLW   0x00
```

Before Instruction
W = 0x00
After Instruction
W = 0x00
Z = 1

IORWF — Inclusive OR W with f

Syntax: [*label*] IORWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (W).OR. (f) → destination

Status Affected: Z

Encoding:

00	0100	dfff	ffff

Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
IORWF   RESULT, 0
```

Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x13
W = 0x93
Z = 0

Example 2

```
IORWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x30
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37
Z = 0

Example 3

```
IORWF   RESULT, 1
```

Case 1:
Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x93
W = 0x91
Z = 0

Case 2:
Before Instruction
RESULT=0x00
W = 0x00
After Instruction
RESULT=0x00
W = 0x00
Z = 1

MOVLW — Move Literal to W

Syntax: [*label*] MOVLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W

Status Affected: None

Encoding:

11	00xx	kkkk	kkkk

Description: The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
MOVLW   0x5A
```

After Instruction
W = 0x5A

Example 2

```
MOVLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x37

Example 3

```
MOVLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x93

MOVF Move f

Syntax: [*label*] MOVF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) → destination

Status Affected: Z

Encoding:

00	1000	dfff	ffff

Description: The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If 'd' = 0, destination is W register. If 'd' = 1, the destination is file register 'f' itself. 'd' = 1 is useful to test a file register since status flag Z is affected.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1
```
MOVF    FSR, 0
```
Before Instruction
W = 0x00
FSR = 0xC2
After Instruction
W = 0xC2
Z = 0

Example 2
```
MOVF    INDF, 0
```
Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3
```
MOVF    FSR, 1
```
Case 1: Before Instruction
FSR = 0x43
After Instruction
FSR = 0x43
Z = 0

Case 2: Before Instruction
FSR = 0x00
After Instruction
FSR = 0x00
Z = 1

29

Instruction Set

MOVWF Move W to f

Syntax: [*label*] MOVWF f

Operands: 0 ≤ f ≤ 127

Operation: (W) → f

Status Affected: None

Encoding:

00	0000	lfff	ffff

Description: Move data from W register to register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1
```
MOVWF   OPTION_REG
```
Before Instruction
OPTION_REG=0xFF
W = 0x4F
After Instruction
OPTION_REG=0x4F
W = 0x4F

Example 2
```
MOVWF   INDF
```
Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x17

NOP — No Operation

Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Encoding:	00 | 0000 | 0xx0 | 0000
Description:	No operation.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example :

```
HERE    NOP
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE + 1

29

Instruction Set

OPTION — Load Option Register

Syntax:	[*label*] OPTION
Operands:	None
Operation:	(W) → OPTION
Status Affected:	None
Encoding:	00 | 0000 | 0110 | 0010
Description:	The contents of the W register are loaded in the OPTION register. This instruction is supported for code compatibility with PIC16C5X products. Since OPTION is a readable/writable register, the user can directly address it.
Words:	1
Cycles:	1

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

RETFIE Return from Interrupt

Syntax: [*label*] RETFIE

Operands: None

Operation: TOS → PC,
1 → GIE

Status Affected: None

Encoding:

00	0000	0000	1001

Description: Return from Interrupt. The 13-bit address at the Top of Stack (TOS) is loaded in the PC. The Global Interrupt Enable bit, GIE (INTCON<7>), is automatically set, enabling Interrupts. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
RETFIE
```

After Instruction
PC = TOS
GIE = 1

RETLW Return with Literal in W

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W;
TOS → PC

Status Affected: None

Encoding:

11	01xx	kkkk	kkkk

Description: The W register is loaded with the eight bit literal 'k'. The program counter is loaded 13-bit address at the Top of Stack (the return address). This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

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

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    CALL TABLE   ; W contains table
                     ; offset value
        •            ; W now has table value
        •
        •
TABLE   ADDWF PC     ;W = offset
        RETLW k1     ;Begin table
        RETLW k2     ;
        •
        •
        •
        RETLW kn     ; End of table
```

Before Instruction
W = 0x07
After Instruction
W = value of k8
PC = TOS = Address `Here + 1`

RETURN — Return from Subroutine

Syntax: [*label*] RETURN

Operands: None

Operation: TOS → PC

Status Affected: None

Encoding:

00	0000	0000	1000

Description: Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    RETURN
```

After Instruction

PC = TOS

29

Instruction Set

RLF — Rotate Left f through Carry

Syntax: [*label*] RLF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

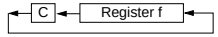
Operation: See description below

Status Affected: C

Encoding:

00	1101	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is stored back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
RLF     REG1,0
```

Before Instruction
REG1= 1110 0110
C = 0
After Instruction
REG1=1110 0110
W =1100 1100
C =1

Example 2

```
RLF     INDF, 1
```

Case 1: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1010
C = 1
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0111 0101
C = 0

Case 2: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 1011 1001
C = 0
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0111 0010
C = 1

RRF

Rotate Right f through Carry

Syntax: [*label*] RRF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

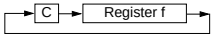
Operation: See description below

Status Affected: C

Encoding:

00	1100	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
RRF     REG1,0
```

Before Instruction
REG1= 1110 0110
W = xxxx xxxx
C = 0
After Instruction
REG1= 1110 0110
W = 0111 0011
C = 0

Example 2

```
RRF     INDF, 1
```

Case 1: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1010
C = 1
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 1001 1101
C = 0

Case 2: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1001
C = 0
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0001 1100
C = 1

SLEEP

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → WDT prescaler count,
1 → $\overline{TO}$,
0 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0011

Description: The power-down status bit, $\overline{PD}$ is cleared. Time-out status bit, $\overline{TO}$ is set. Watchdog Timer and its prescaler count are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Go to sleep

Example:

```
SLEEP
```

Note: The SLEEP instruction does not affect the assignment of the WDT prescaler

SUBLW Subtract W from Literal

Syntax: [*label*] SUBLW k

Operands: 0 ≤ k ≤ 255

Operation: k - (W) → W

Status Affected: C, DC, Z

Encoding:

11	110x	kkkk	kkkk

Description: The W register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBLW   0x02
```
Case 1: Before Instruction
```
W = 0x01
C = x
Z = x
```
After Instruction
```
W = 0x01
C = 1      ; result is positive
Z = 0
```
Case 2: Before Instruction
```
W = 0x02
C = x
Z = x
```
After Instruction
```
W = 0x00
C = 1      ; result is zero
Z = 1
```
Case 3: Before Instruction
```
W = 0x03
C = x
Z = x
```
After Instruction
```
W = 0xFF
C = 0      ; result is negative
Z = 0
```

Example 2
```
SUBLW   MYREG
```
Before Instruction
```
W = 0x10
```
Address of MYREG † = 0x37

† MYREG is a symbol for a data memory location

After Instruction
```
W = 0x27
C = 1      ; result is positive
```

29

Instruction Set

SUBWF Subtract W from f

Syntax: [*label*] SUBWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) - (W) → destination

Status Affected: C, DC, Z

Encoding:

00	0010	dfff	ffff

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWF   REG1,1
```
Case 1: Before Instruction
```
REG1= 3
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 1
W   = 2
C   = 1      ; result is positive
Z   = 0
```
Case 2: Before Instruction
```
REG1= 2
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0
W   = 2
C   = 1      ; result is zero
Z   = 1
```
Case 3: Before Instruction
```
REG1= 1
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0xFF
W   = 2
C   = 0      ; result is negative
Z   = 0
```

SWAPF — Swap Nibbles in f

Syntax: [*label*] SWAPF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f<3:0>) → destination<7:4>,
(f<7:4>) → destination<3:0>

Status Affected: None

Encoding:

00	1110	dfff	ffff

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0 the result is placed in W register. If 'd' is 1 the result is placed in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
SWAPF   REG, 0
```

Before Instruction
REG1= 0xA5

After Instruction
REG1= 0xA5
W = 0x5A

Example 2

```
SWAPF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20

After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x02

Example 3

```
SWAPF   REG, 1
```

Before Instruction
REG1= 0xA5

After Instruction
REG1= 0x5A

TRIS — Load TRIS Register

Syntax: [*label*] TRIS f

Operands: 5 ≤ f ≤ 7

Operation: (W) → TRIS register f;

Status Affected: None

Encoding:

00	0000	0110	0fff

Description: The instruction is supported for code compatibility with the PIC16C5X products. Since TRIS registers are readable and writable, the user can directly address them.

Words: 1

Cycles: 1

Example

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

XORLW Exclusive OR Literal with W

Syntax: [*label*] XORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W).XOR. k → W

Status Affected: Z

Encoding:

11	1010	kkkk	kkkk

Description: The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
XORLW   0xAF                ; 1010 1111   (0xAF)
```
Before Instruction
```
                            ; 1011 0101   (0xB5)
    W  =  0xB5              ; ---------   ------
```
After Instruction
```
                            ; 0001 1010   (0x1A)
    W  =  0x1A
    Z  =  0
```

Example 2

```
XORLW   MYREG
```
Before Instruction
```
    W  =  0xAF
    Address of MYREG † = 0x37
```
† MYREG is a symbol for a data memory location

After Instruction
```
    W  =  0x18
    Z  =  0
```

Example 3

```
XORLW   HIGH (LU_TABLE)
```
Before Instruction
```
    W  =  0xAF
    Address of LU_TABLE † = 0x9375
```
† LU_TABLE is a label for an address in program memory

After Instruction
```
    W  =  0x3C
    Z  =  0
```

29

Instruction Set

XORWF Exclusive OR W with f

Syntax: [*label*] XORWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (W).XOR. (f) → destination

Status Affected: Z

Encoding:

00	0110	dfff	ffff

Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1

```
XORWF   REG, 1              ; 1010 1111   (0xAF)
```
Before Instruction
```
                            ; 1011 0101   (0xB5)
    REG=  0xAF              ; ---------   ------
    W  =  0xB5              ; 0001 1010   (0x1A)
```
After Instruction
```
    REG=  0x1A
    W  =  0xB5
```

Example 2

```
XORWF   REG, 0              ; 1010 1111   (0xAF)
```
Before Instruction
```
                            ; 1011 0101   (0xB5)
    REG=  0xAF              ; ---------   ------
    W  =  0xB5              ; 0001 1010   (0x1A)
```
After Instruction
```
    REG=  0xAF
    W  =  0x1A
```

Example 3

```
XORWF   INDF, 1
```
Before Instruction
```
    W  =  0xB5
    FSR=  0xC2
    Contents of Address (FSR) = 0xAF
```
After Instruction
```
    W  =  0xB5
    FSR=  0xC2
    Contents of Address (FSR) = 0x1A
```

29.6 Design Tips

Question 1: ***How can I modify the value of W directly? I want to decrement W.***

Answer 1:

There are a few possibilities, two are:

1. For the midrange devices, there are several instructions that work with a literal and W. For instance, if it were desired to decrement W, this can be done with an `ADDLW 0xFF` (the 0x prefix denotes hex to the assembler)
2. Notice that all of the instructions can modify a value right where it sits in the file register. This means you can decrement it right where it is. You do not even need to move it to W. If you want to decrement it AND move it somewhere else, then you make W the DESTINATION of the decrement (`DECF` register,W) then put it where you want it. It is the same number of instructions as a straight move, but it gets decremented along the way.

Question 2: ***Is there any danger in using the `TRIS` instruction for the PIC16CXXX since there is a warning in the Data book suggesting it not be used?***

Answer 2:

For code compatibility and upgrades to later parts, the use of the TRIS instruction is not recommended. You should note the `TRIS` instruction is limited to ports A, B and C. Future devices may not support these instructions.

Question 3: ***Do I have to switch to Bank1 of data memory before using the `TRIS` instruction (for parts with TRIS registers in the memory map)?***

Answer 3:

No. The `TRIS` instruction is Bank independent. Again the use of the `TRIS` instruction is not recommended.

Question 4: ***I have seen references to "Read-Modify-Write" instructions in your data sheet, but I do not know what that is. Can you explain what it is and why I need to know this?***

Answer 4:

An easy example of a Read-Modify-Write (R-M-W) instruction is the bit clear instruction `BCF`. You might think that the processor just clears the bit, which on a port output pin would clear the pin. What actually happens is the whole port (or register) is first read, THEN the bit is cleared, then the new modified value is written back to the port (or register). Actually, any instruction that depends on a value currently in the register is going to be a Read-Modify-Write instruction. This includes `ADDWF, SUBWF, BCF, BSF, INCF, XORWF`, etc... Instructions that do not depend on the current register value, like `MOVWF, CLRF`, and so on are not R-M-W instructions.

One situation where you would want to consider the affects of a R-M-W instruction is a port that is continuously changed from input to output and back. For example, say you have TRISB set to all outputs, and write all ones to the PORTB register, all of the PORTB pins will go high. Now, say you turn pin RB3 into an input, which happens to go low. A `BCF PORTB,6` is then executed to drive pin RB6 low. If you then turn RB3 back into an output, it will now drive low, even though the last value you put there was a one. What happened was that the `BCF` of the other pin (RB6) caused the whole port to be read, including the zero on RB3 when it was an input. Then, bit 6 was changed as requested, but since RB3 was read as a zero, zero will also be placed back into that port latch, overwriting the one that was there before. When the pin is turned back into an output, the new value was reflected.

Question 5: ***When I perform a `BCF` other pins get cleared in the port. Why?***

Answer 5:

There are a few possibilities, two are:

1. Another case where a R-M-W instruction may seem to change other pin values unexpectedly can be illustrated as follows: Suppose you make PORTC all outputs and drive the pins low. On each of the port pins is an LED connected to ground, such that a high output lights it. Across each LED is a 100 µF capacitor. Let's also suppose that the processor is running very fast, say 20 MHz. Now if you go down the port setting each pin in order; `BSF PORTC,0` then BSF PORTC,1 then `BSF PORTC,2` and so on, you may see that only the last pin was set, and only the last LED actually turns on. This is because the capacitors take a while to charge. As each pin was set, the pin before it was not charged yet and so was read as a zero. This zero is written back out to the port latch (R-M-W, remember) which clears the bit you just tried to set the instruction before. This is usually only a concern at high speeds and for successive port operations, but it can happen so take it into consideration.
2. If this is on a PIC16C7X device, you may not have configured the I/O pins properly in the ADCON1 register. If a pin is configured for analog input, any read of that pin will read a zero, regardless of the voltage on the pin. This is an exception to the normal rule that the pin state is always read. You can still configure an analog pin as an output in the TRIS register, and drive the pin high or low by writing to it, but you will always read a zero. Therefore if you execute a Read-Modify-Write instruction (see previous question) all analog pins are read as zero, and those not directly modified by the instruction will be written back to the port latch as zero. A pin configured as analog is expected to have values that may be neither high nor low to a digital pin, or floating. Floating inputs on digital pins are a no-no, and can lead to high current draw in the input buffer, so the input buffer is disabled.

29.7 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Mid-Range MCU family (that is they may be written for the Base-Line, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the instruction set are:

Currently No related Application Notes