S34PH412 : Systèmes Microprogrammés

Architecture des Ordinateurs

Cours 5

Université de la Réunion

Année 2015/2016

Alicalapa F.

UNIVERSITÉ DE LA RÉUNION
S'ouvrir aux mondes

UFR Sciences et Technologies UNIVERSITÉ DE LA RÉUNION

- **Exemple d'utilisation et de vocabulaire à maitriser : PIC 10F200**

4.1 Program Memory Organization for the PIC10F200/204

The PIC10F200/204 devices have a 9-bit Program Counter (PC) capable of addressing a 512 x 12 program memory space.

Only the first 256 x 12 (0000h-00FFh) for the PIC10F200/204 are physically implemented (see Figure 4-1). Accessing a location above these boundaries will cause a wraparound within the first 256 x 12 space (PIC10F200/204). The effective Reset vector is at 0000h (see Figure 4-1). Location 00FFh (PIC10F200/204) contains the internal clock oscillator calibration value. This value should never be overwritten.

- Quelques **registres généraux et spéciaux** du PIC
- RAM

RAM Memory Banks

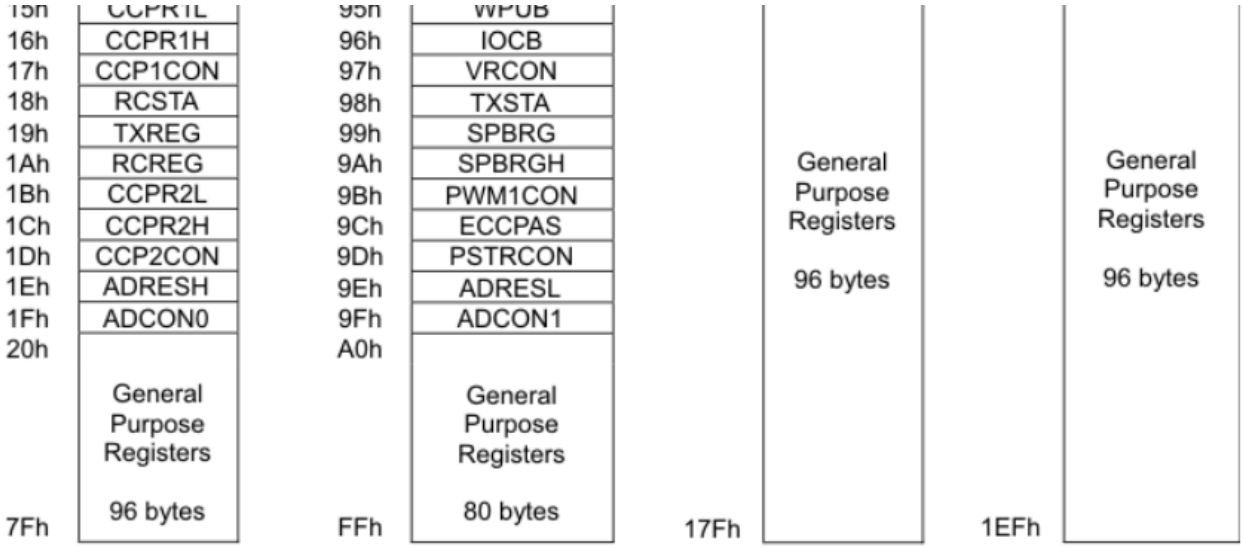
The data memory is partitioned into four banks. Prior to accessing some register during program writing (in order to read or change its contents), it is necessary to select the bank which contains that register. Two bits of the STATUS register are used for bank selecting, which will be discussed later. In order to facilitate operation, the most commonly used SFRs have the same address in all banks which enables them to be easily accessed.

Addr.	Name
00h	INDF
01h	TMR0
02h	PCL
03h	STATUS
04h	FSR

Addr.	Name
80h	INDF
81h	OPTION_REG
82h	PCL
83h	STATUS
84h	FSR

Addr.	Name
100h	INDF
101h	TMR0
102h	PCL
103h	STATUS
104h	FSR

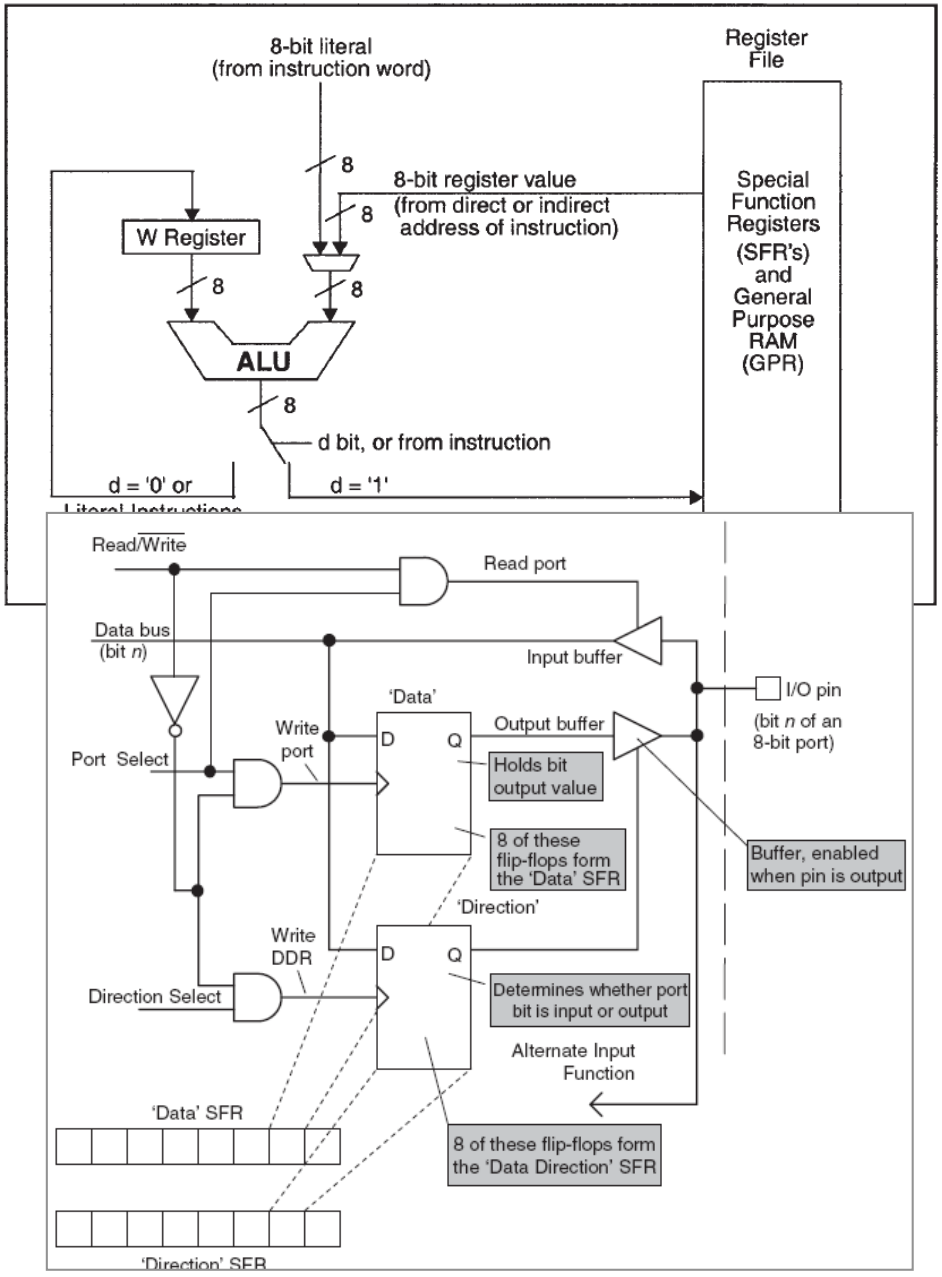
Addr.	Name
180h	INDF
181h	OPTION_REG
182h	PCL
183h	STATUS
184h	FSR



- Quelques **registres généraux et spéciaux** du PIC

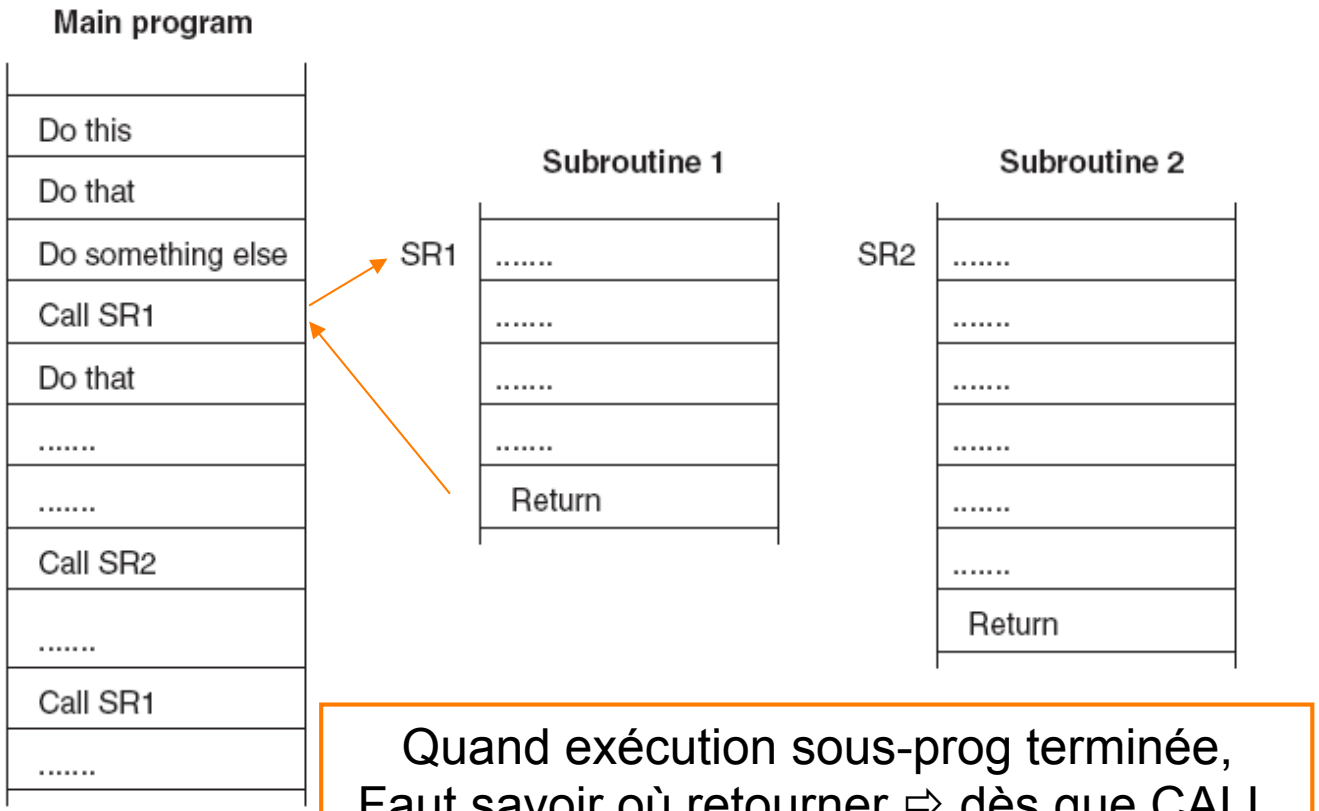
FIGURE 2-2: PIC16F688 SPECIAL FUNCTION REGISTERS

	File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h
TMR0	01h	OPTION_REG	81h	TMR0	101h
PCL	02h	PCL	82h	PCL	102h
STATUS	03h	STATUS	83h	STATUS	103h
FSR	04h	FSR	84h	FSR	104h
PORTA	05h	TRISA	85h	PORTA	105h
	06h		86h		106h
PORTC	07h	TRISC	87h	PORTC	107h
	08h		88h		108h
	09h		89h		109h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh
PIR1	0Ch	PIE1	8Ch		10Ch
	0Dh		8Dh		10Dh
TMR1L	0Eh	PCON	8Eh		10Eh
TMR1H	0Fh	OSCCON	8Fh		10Fh
T1CON	10h	OSCTUNE	90h		110h



- **Reg. Instructions**
  - RI
  - Reg. données de l'UC, contient le **code** de **l'instruction en cours** d'exécution
- **Reg. « compteur de programme »**
  - Program Counter (PC), Instruction Pointer (IP), compteur ordinal (CO)
  - reg. d'adresse de l'UC, contient **l'adresse** de l'instruction **n+1**
- **Reg. « Accumulateurs ou registre de travail»**
  - ACC, Wreg ou W (working), reg. données, stocke résultats de l'UAL
- **Reg. d'état, « Program Status Word - PSW» ou « Condition Code Register - CCR », Status Register**
  - appartient à l'UAL
  - contient des drapeaux « Flag » = indicateurs qui renseignent sur le résultat de l'UAL (nul, positif, etc…)
  - Bit Z = 1 ↔ résultat opération nul, Bit O ou V ↔ OverFlow (?), Bit C ↔ Carry (?), Bit S ↔résultat positif, Bit I = 1 ↔ interruption possible ou non, Bit N = 1 ↔ résultat négatif

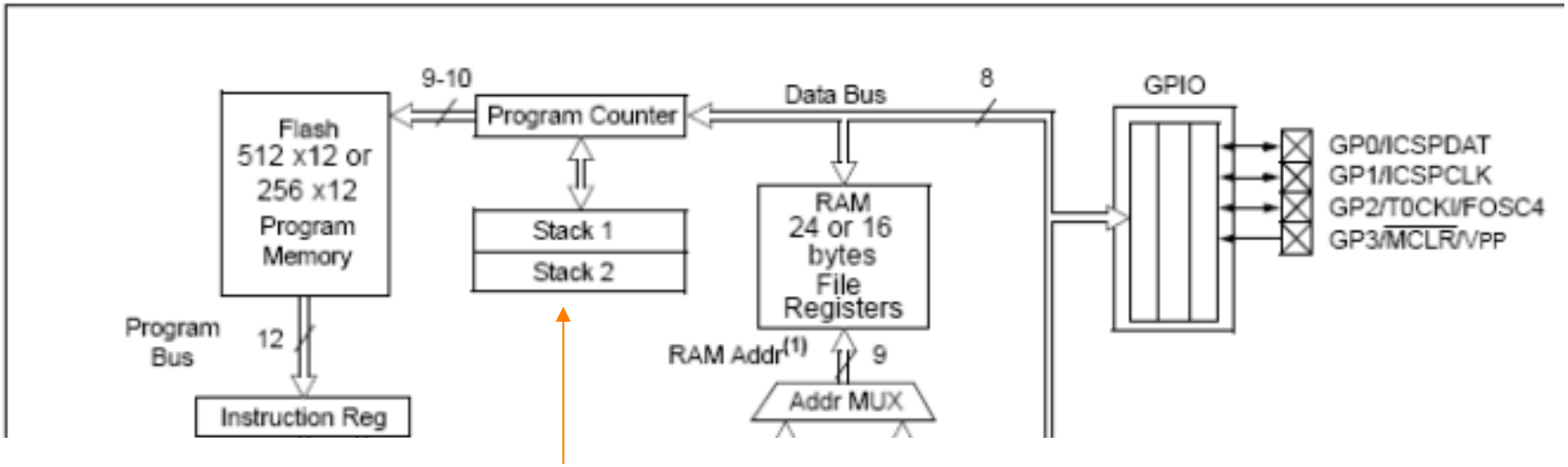
- **Reg. Pointeur de pile, STACK pointer**
  - permet de gérer une pile LIFO ou FIFO dans la mémoire centrale
  - Sert à stocker l'adresse de départ quand **utilise les sous-programmes**.



Quand exécution sous-prog terminée,
Faut savoir où retourner ⇨ dès que CALL a été utilisée, adresse départ a été stockée dans le STACK

- **Reg. Pointeur de pile, STACK pointer**
  - Localisation du registre STACK :

FIGURE 3-1: PIC10F200/202 BLOCK DIAGRAM



  - Combien de sous programmes imbriqués puis-je appeler avec ce µC ???

I. Notion de numération binaire (le monde du numérique et de l'analogique)

II. Rapide Historique de l'évolution des ordinateurs

III. Le modèle Von Neuman et représentation hiérarchique

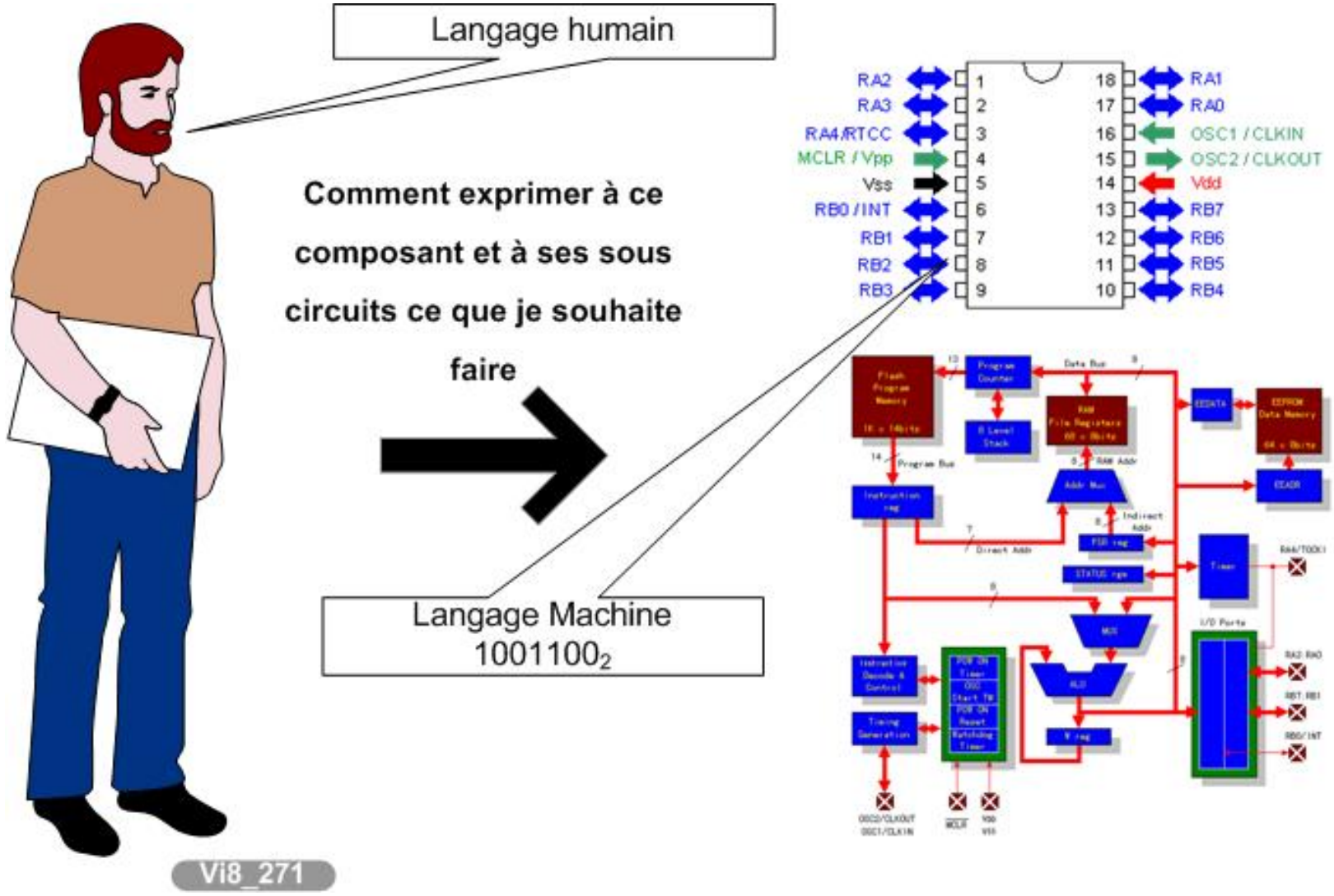
IV. Les différents composants du système et leur caractéristiques

a. Les mémoires

b. Le processeur :

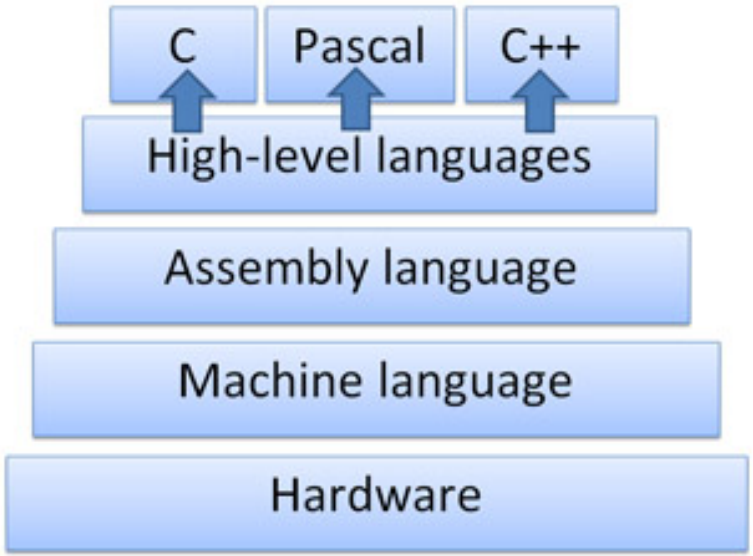
1. modélisation primaire
2. L'horloge
3. **Jeu d'instruction**

- La situation de communication :



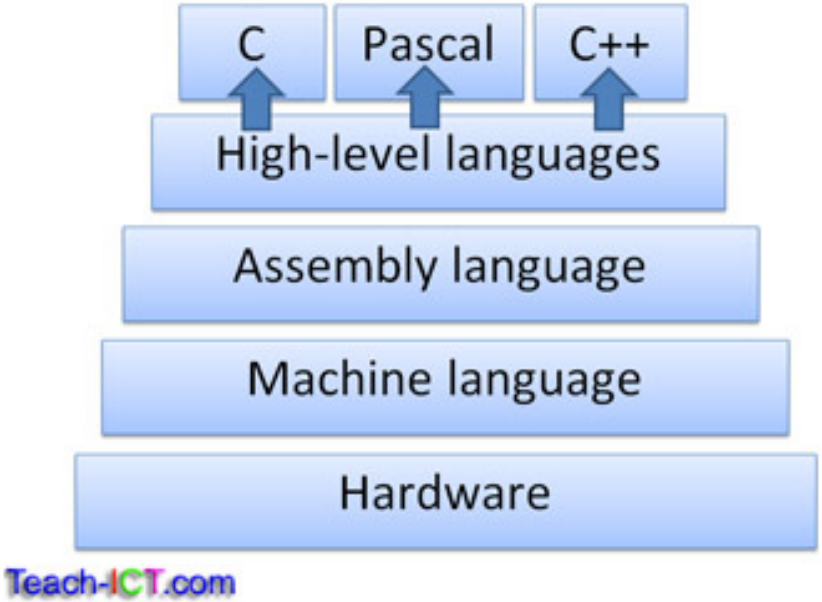
- Choix possible (L'assembleur ??? Et le langage machine ???) :
- Choix 1 : Le **processeur « s'adapte » à l'homme** ⇨ utilisation de langage de programmation de **haut niveau C et de compilateur C (ou autre langage : C++, .Net, Java, etc …)**

Vi9_271



Teach-ICT.com

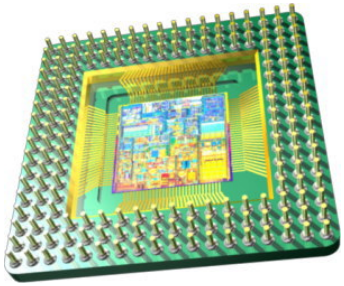
- Le concepteur est « loin » de la manipulation des ressources matérielles du µC ⇨ les programmes ne sont pas forcément optimisés en terme d'utilisation des ressources matérielles et donc en vitesse d'exécution. L'utilisateur n'apprend rien sur l'architecture du µC pour optimiser son fonctionnement.

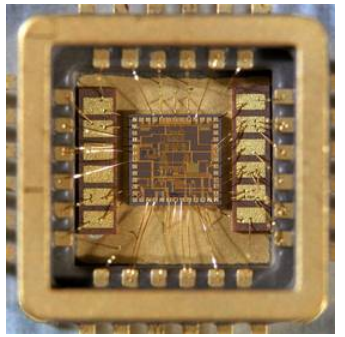


- Choix possible (L'assembleur ??? Et le langage machine ???) :
- **Choix 2 : L'humain « s'adapte » au processeur**
  - Hors de question de « parler » le langage du processeur : 100111_2, incompréhensible !!!
  - ⇨ utilisation d'un **langage proche** de celui du processeur (⇨ plus optimisé en utilisation de ressources, acquisition de connaissances sur le µC et ses registres) = langage **ASSEMBLEUR** = utilisation de **mnémoniques.**
  - Exemple de mnémonique :

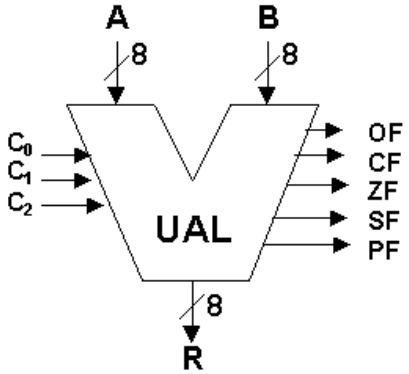
chargement de l'accumulateur
LDA #<valeur> 2 octets code=10
LDA <adr 16 bits> 3 octets code=12
LDA <adr 8 bits> 2 octets code=11

Le processeur (P)





- Les instructions que comprend le µP sont très simples = « son **jeu d'instructions** » : lire des cases mémoire (les registres), additionner, multiplier 2 nombres binaires, décalage, opération binaire...



Le jeu d'instructions

- langage de programmation d'un µP/µC = **langage machine composé que de 1 et de 0** (signification **non évidente** pour l'homme).
- Ex: additionner deux nombres : 10001010(instruction) 01011000(1er opérande) 11010010(2ème opérande).

- Pour une meilleure **lisibilité**, on a donné des noms aux opérations (**mnémoniques**), exemple : ADD, DIV, OR etc. .
- Pour **convertir mnémoniques ⇨ langage machine**, utilise un programme que l'on nomme **assembleur**

PIC10F200/202/204/206

TABLE 10-2: INSTRUCTION SET SUMMARY

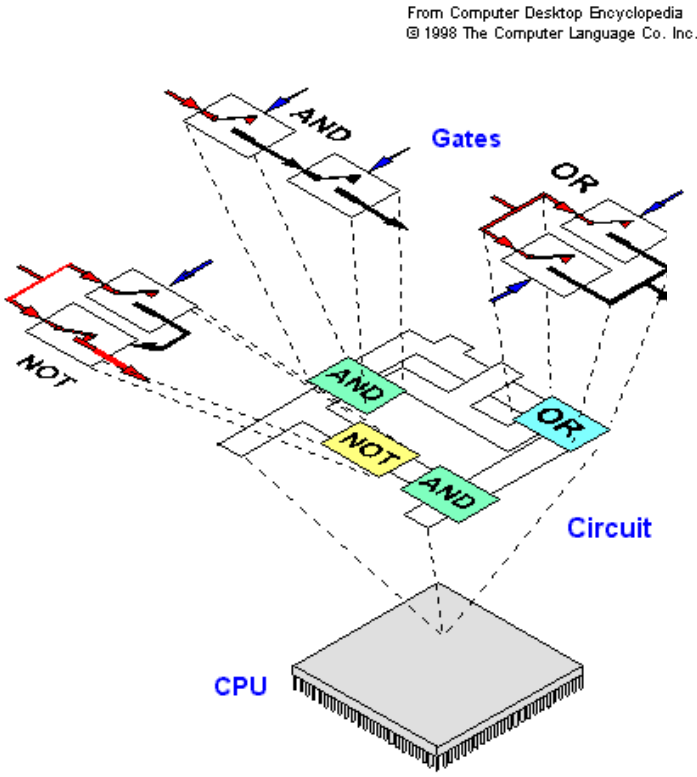
Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected	Notes
ADDWF	f, d	Add W and f	1	0001	11df	ffff	C, DC, Z	1, 2, 4
ANDWF	f, d	AND W with f	1	0001	01df	ffff	Z	2, 4
CLRF	f	Clear f	1	0000	011f	ffff	Z	4
CLRW	—	Clear W	1	0000	0100	0000	Z	
COMF	f, d	Complement f	1	0010	01df	ffff	Z	
DECF	f, d	Decrement f	1	0000	11df	ffff	Z	2, 4
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	0010	11df	ffff	None	2, 4
INCF	f, d	Increment f	1	0010	10df	ffff	Z	2, 4

- Au lieu du terme assembleur, **CROSS-ASSEMBLEUR** car votre « ordinateur = un microprocesseur AMD, INTEL ou SPARC» réalise une traduction en langage machine pour un autre processeur PIC par exemple.

- L'ensemble des **mnémoniques** constituent le jeu d'instruction du µC

Le jeu d'instructions

- c'est l'ensemble des **instructions de base** « **presque** câblées en réel» sur la puce, donc que le microprocesseur peut exécuter **sans « traduction » = rapidité**
- Ces **instructions** simples qui sont **codées en binaire = le langage machine**



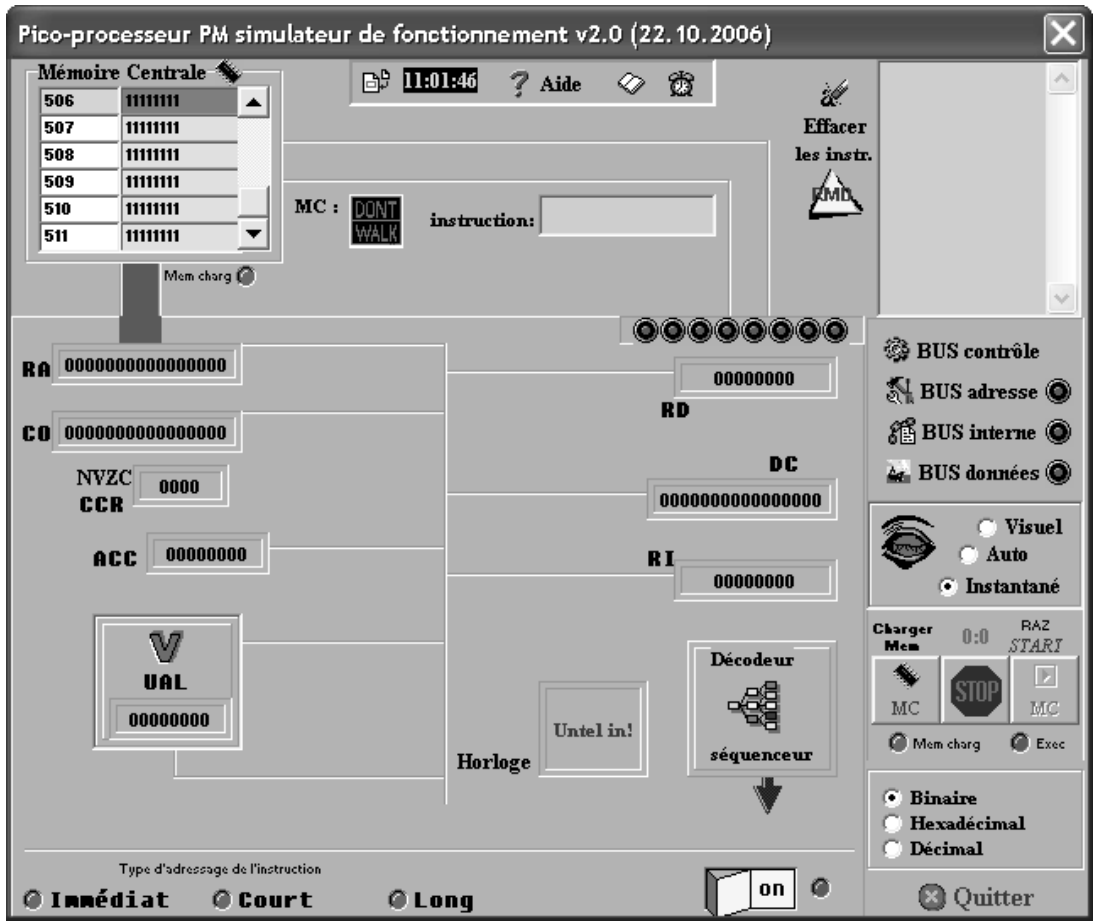
PIC10F200/202/204/206

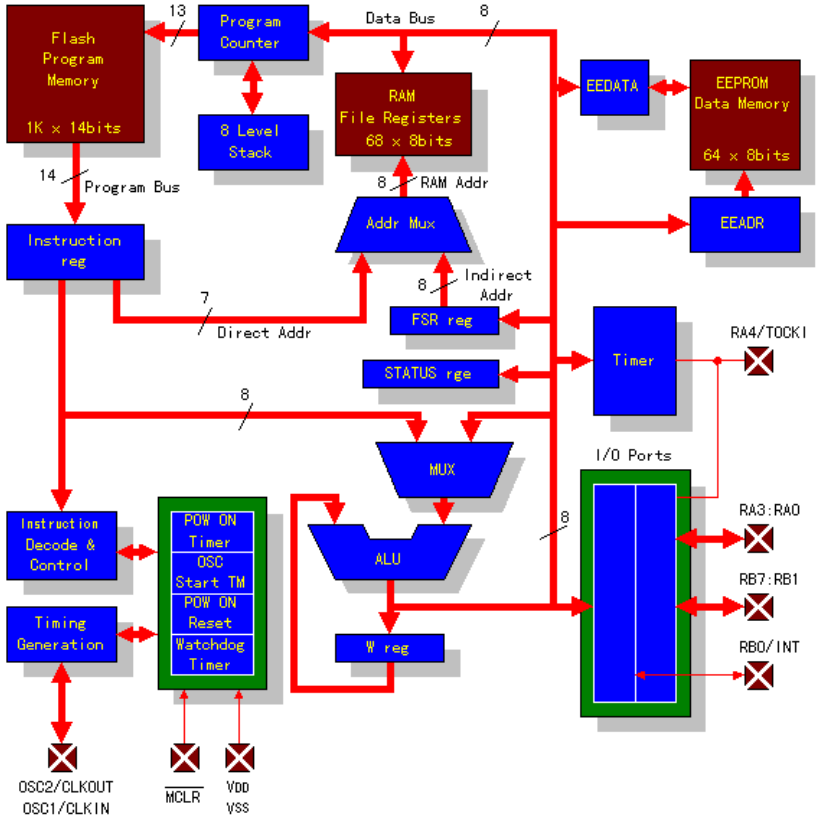
TABLE 10-2: INSTRUCTION SET SUMMARY

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected	Notes
ADDWF	f, d	Add W and f	1	0001	11df	ffff	C, DC, Z	1, 2, 4
ANDWF	f, d	AND W with f	1	0001	01df	ffff	Z	2, 4
CLRF	f	Clear f	1	0000	011f	ffff	Z	4
CLRW	—	Clear W	1	0000	0100	0000	Z	
COMF	f, d	Complement f	1	0010	01df	ffff	Z	
DECF	f, d	Decrement f	1	0000	11df	ffff	Z	2, 4
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	0010	11df	ffff	None	2, 4
INCF	f, d	Increment f	1	0010	10df	ffff	Z	2, 4

Exemple de programme en assembleur :

```
addlw  120   ; Add the constant 120 (#120) decimal to W
clrw         ; Clear the Working register
clrf 20h     ; Clear the byte at File store address 20h
```





Exemple de programme en assembleur :

```
;  Delay = W x 1 sec
;
delay1s_R
        banksel dc4             ;
sec
        movwf   dc4
dly3    movlw   .100            ;
        movwf   dc3             ;
dly2    movlw   .13             ;
        movwf   dc2             ;
        clrf    dc1             ;
dly1    decfsz  dc1,f
        goto    dly1
```

Exemple de programme en C (langage haut niveau) :

```
for (p_cnt = 0; p_cnt < 1000/POLL_MS; p_cnt++)
{
    __delay_ms(POLL_MS);    // polling interval

    // check for button press
    if (!BUTTON)            // if button pressed
    {
        time_cnt = 0;           // reset timeout coun

        // light next LED in sequence
        LEDS = 0b111111;        // turn off all LEDs

        switch (state)          // next LED depends o
        {
            case GREEN:             // if green:
```

Exemple de jeu d'instruction d'un PIC :

Instruction	Mnemonic	Operation	Flags	
			Z	C
Arithmetic				
Add W and F	addwf f,d	(d) <- W + (f)	√	√
Add Literal and W	addlw L	W <- #L + W	√	√
Clear F	clrf f	(f) <- 00	√	•
Clear W	clrw	(f) <- 00	√	•
Increment F	incf f,d	(d) <- (f) + 1	√	•
Decrement F	decf f	(d) <- (f) - 1	√	•
Subtract W from F	subwf f,d	(d) <- (f) - W	√	√
Subtract W from L	sublw L	W <- #L - W	√	√
Movement				
Move F	movf f,d	(d) <- (f)	√	•
Move W to F	movwf f	W <- (f)	•	•
Move Literal to W	movlw L	W <- #L	•	•
Logic				
AND W and F	andwf f,d	(d) <- W · (f)	√	•
AND Literal and W	andlw L	W <- #L · W	√	•
Complement F	comf f	(f) <- $\overline{(f)}$	√	•
Inclusive OR W and F	iorwf f,d	(d) <- W + (f)	√	•
Inclusive OR Literal and W	iorlw L	W <- #L + W	√	•
Rotate left F	rlf f,d	(d) <- C ← f	•	√
Rotate right F	rlr f,d	(d) <- f → C	•	√
eXclusive OR W and F	xorwf f,d	(d) <- W ⊕ (f)	√	•
eXclusive OR Literal and W	xorlw L	W <- #L ⊕ W	√	•
Skip and Jump				
Bit Test F, Skip if Clear	btfsc f,b	b == 0 ? PC++ : PC	•	•
Bit Test F, Skip if Set	btfss f,b	b == 1 ? PC++ : PC	•	•
Decrement F, Skip if Zero	decfsz f	--(f) == 0 ? PC++ : PC	√	•
Go to address	goto <ea>	PC <- <ea>	•	•
Increment F, Skip if Zero	incfsz f	++(f) == 0 ? PC++ : PC	√	•

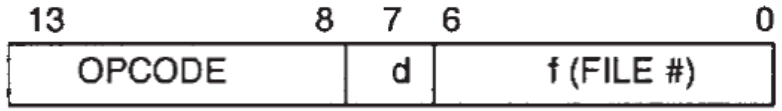
Il faut à partir de ces simples instructions pouvoir faire des programmes qui réalisent une détection d'alarme de température et un affichage sur des afficheurs 7 segments d'un message par exemple

Le jeu d'instructions

- Les **instructions** sont en fait composées : **Opcode** (code opération ex: 00011 = addition) + des **informations sur les opérandes** de l'opération (valeur avec laquelle il faut faire une addition…)

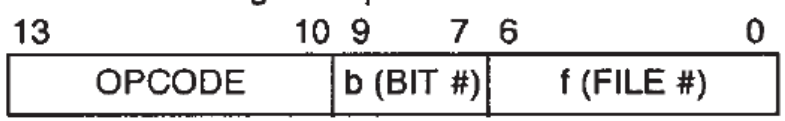
Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1, 2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	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

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

- Exemple de jeu d'instruction du PIC (🡻)

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

Cycle exécution

Mnémoniques

Langage Machine
Rem : les opérandes sont dans le code machine « dff… »

Digit du Registre d'état positionné

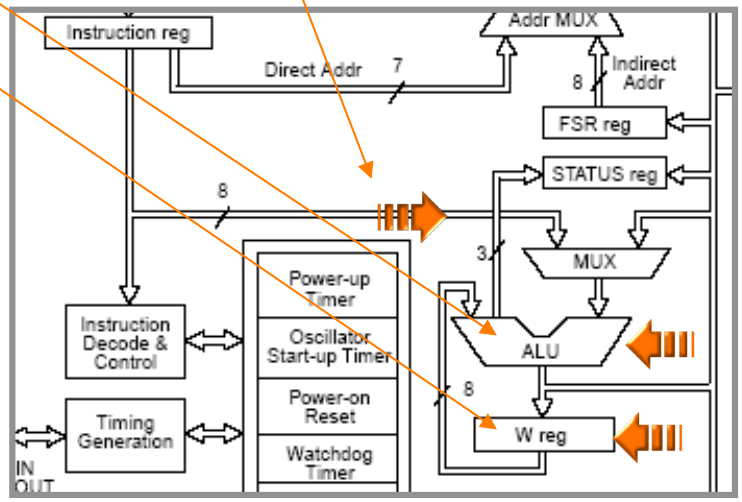
- Exemple de l'instruction ADDLW (🡻)

- Exemple de l'instruction ADDLW (🡻)

LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11	111x kkkk	kkkk C, DC, Z

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

- Sommation d'un nombre contenu dans l'instruction avec le registre de travail W

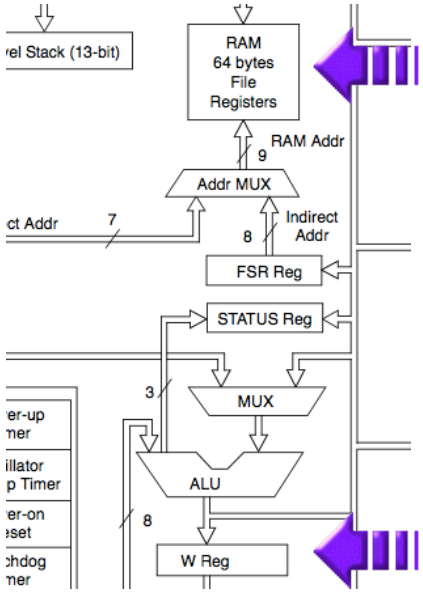


- **Autre instruction** ADDWF, « d » = le bit de destination du résultat de l'addition, permet d'envoyer le résultat vers W ou « vers » l'adresse f d'origine

BYTE-ORIENTED FILE REGISTER OPERATIONS

ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C, DC, Z

- Le résultat est stocké soit dans le registre W ou en mémoire f



ADDWF	Add W and f
Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) + (f) → (destination)
Status Affected:	C, DC, Z
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'.

- Dans l'instruction ADDWF, « f » = l'adresse de l'opérande de l'instruction = « sur qui va agir l'instruction »

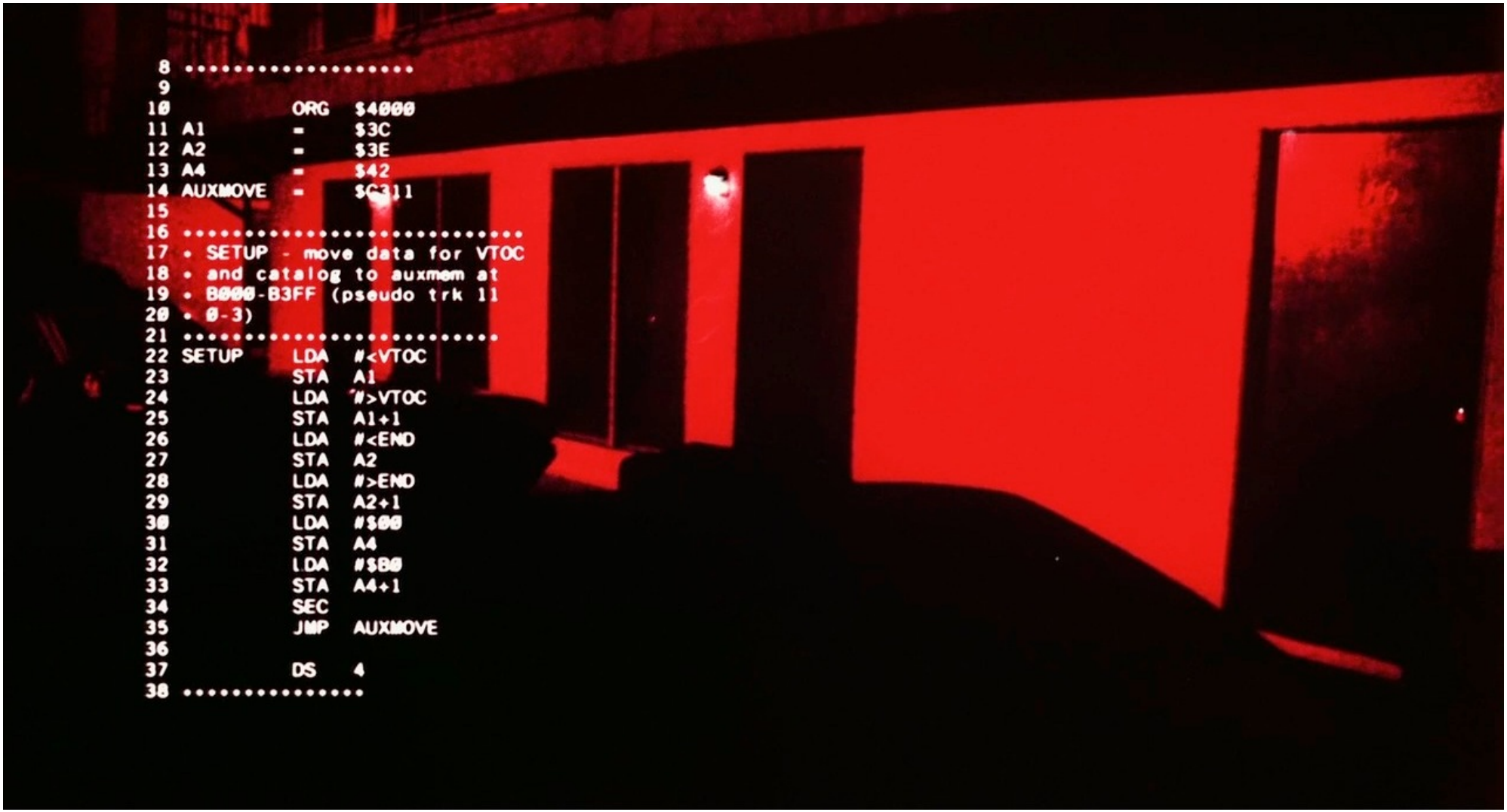
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C, DC, Z	1, 2
ANDWF		AND W with f	1	00 0101 dfff ffff	Z	1, 2

- f : fichier = une zone RAM de 7 bits
- b : bit
- k : littéral = une données sur 8 bits ou une adresse sur 11 bits

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

Hollywood utilise l'assembleur !!!!

Hollywood utilise l'assembleur !!!!

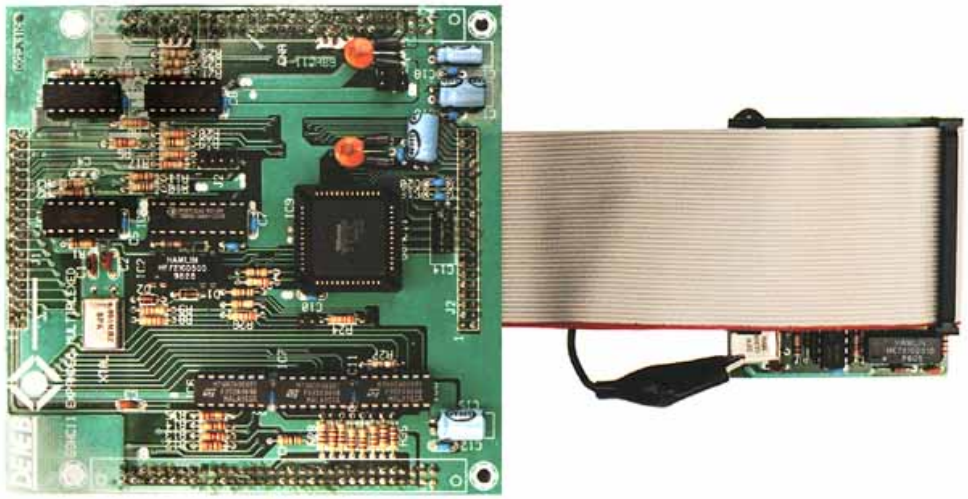


```
 8 ••••••••••••••••••
 9
10           ORG   $4000
11 A1        =     $3C
12 A2        =     $3E
13 A4        =     $42
14 AUXMOVE   =     $C311
15
16 ••••••••••••••••••••••••••••
17 • SETUP - move data for VTOC
18 • and catalog to auxmem at
19 • B000-B3FF (pseudo trk 11
20 • 0-3)
21 ••••••••••••••••••••••••••
22 SETUP     LDA   #<VTOC
23           STA   A1
24           LDA   #>VTOC
25           STA   A1+1
26           LDA   #<END
27           STA   A2
28           LDA   #>END
29           STA   A2+1
30           LDA   #$00
31           STA   A4
32           LDA   #$B0
33           STA   A4+1
34           SEC
35           JMP   AUXMOVE
36
37           DS    4
38 ••••••••••••••
```

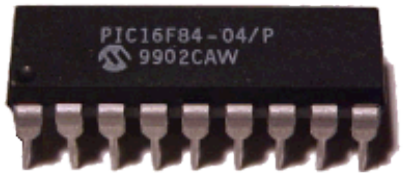
- **Chaque famille** de microprocesseurs possède **son propre jeu** d'instructions. Ce jeu d'instructions est tributaire d'exigences telles que
  - La simplicité du programme réalisé
  - La rapidité d'exécution du programme
  - L'universalité des instructions (conversion, exportation de programmes)
  - L'occupation sur le wafer de silicium.
  - Le « confort » du programmeur en terme de fonctions disponibles.

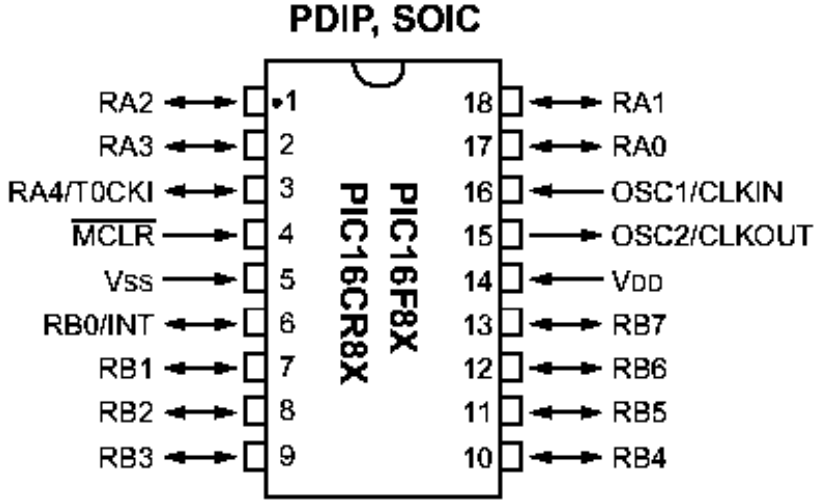
- ⇨ **Bien choisir son µP** (et donc son jeu d'instruction) **en fonction de l'application souhaitée**…ne sert à rien d'avoir « un gros » processeur avec beaucoup d'instructions si on utilise 10% de ces instructions (coût, vitesse, encombrement sur la carte, consommation énergétique)

Le jeu d'instructions



Le 68HC11, **nombre d'instructions > 100**.





le pic 16f84 ne comporte que **35 instructions** (architecture RISC)
Le temps d'apprentissage en est réduit d'autant.

- RISC et CISC ????
- architecture **CISC** (Complex Instruction Set Code) un processeur dont jeu d'instructions possède les propriétés suivantes :
  - Il contient beaucoup de classes d'instructions différentes.
  - Il contient beaucoup de type d'instructions différentes complexes et de taille variable.
  - Il se sert de beaucoup de registres spécialisés et de peu de registres généraux.

- L'architecture **RISC** (Reduced Instruction Set Code), un processeur RISC est un processeur dont le jeu d'instructions possède les propriétés suivantes :
  - Le nombre de classes d'instructions différentes est réduit par rapport à un CISC.
  - Les instructions sont de taille fixe.
  - Il se sert de beaucoup de registres généraux.
  - Il fonctionne avec un pipe-line

- Depuis les décennies 90, les microprocesseur adoptent le meilleur des fonctionnalités de chaque architecture provoquant de fait la **disparition progressive de la différence entre RISC et CISC** et l'inévitables polémiques sur l'efficacité supposée meilleure de l'une ou de l'autre architecture.

Le jeu d'instructions (**voir exemple document imprimé 16F690 - TP**)

- Les instructions peuvent être classées en six catégories.
  1. Transfert de données, entre le P et la mémoire, MOVE, LOAD, STORE,. . .
  2. Opérations arithmétiques, ADD, SUB,. . .
  3. Opérations logiques OR, AND,. . .
  4. Contrôles des séquences, sauts conditionnels ou inconditionnels, . .
  5. Entrées / Sorties, entre le P et les périphériques,. . .
  6. Divers, décalages, incrémentation,. . .

- Ou se connecter au PC : Kit Pédagogique Discala si installer.

Exemple de programme en PM

```
LDA #18 ; {chargement de l'accumulateur avec la valeur 18}
STA 50 ; {rangement de l'accumulateur dans la mémoire n°50}
LDA #5 ; {chargement de l'accumulateur avec la valeur 5}
STA 51 ; {rangement de l'accumulateur dans la mémoire n°51}
ADD 50 ; {addition de l'accumulateur avec la mémoire n°50}
STA 52 ; {rangement de l'accumulateur dans la mémoire n°52}
END
```

I. Notion de numération binaire (le monde du numérique et de l'analogique)

II. Rapide Historique de l'évolution des ordinateurs

III. Le modèle Von Neuman et représentation hiérarchique

IV. Les différents composants du système et leur caractéristiques

a. Les mémoires

b. Le processeur :

1. modélisation primaire
2. L'horloge
3. Jeu d'instruction
4. **Les interruptions**

- Concept de base :
  - **envoi d'un signal électrique au µP alors que celui-ci est en cours d'exécution d'une d'un programme. Ceci de manière ASYNCHRONE (??).**
    - **Le µP possède une ou des entrées IRQ (Interrupt ReQuest) sur la puce, à laquelle est associée « bit d'interruption ».**
    - **Une interruption peut aussi venir des compteurs du uC (compteur = chronomètre)**
    - **Enfin une IRQ peut être généré après l'exécution d'une instruction**
  - **Exécution d'une routine spécifique (= programme court spécifique) de service associée à l'interruption**

- Une routine de service spécifique (= petit programme) est exécutée pour « répondre » au problème signalé par l'IRQ.

- L'interruption est aussi une fonction du µP ⇨ paramétrable par registre (⇩)

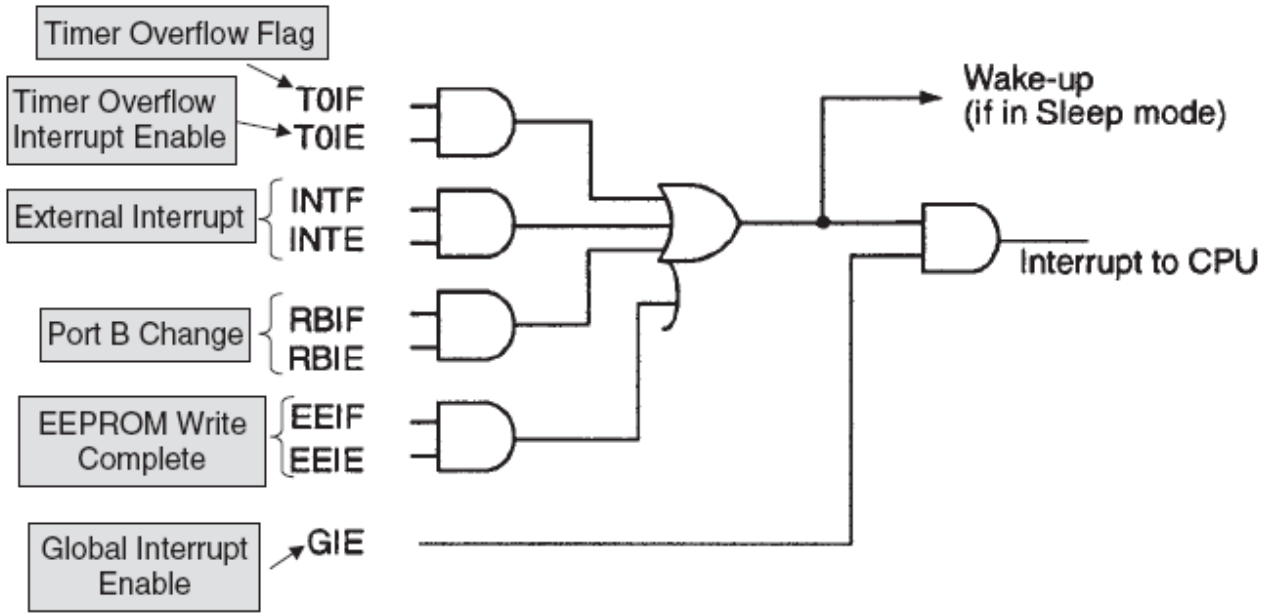
- Registre INTCON du PIC

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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

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

bit 6 **EEIE:** EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

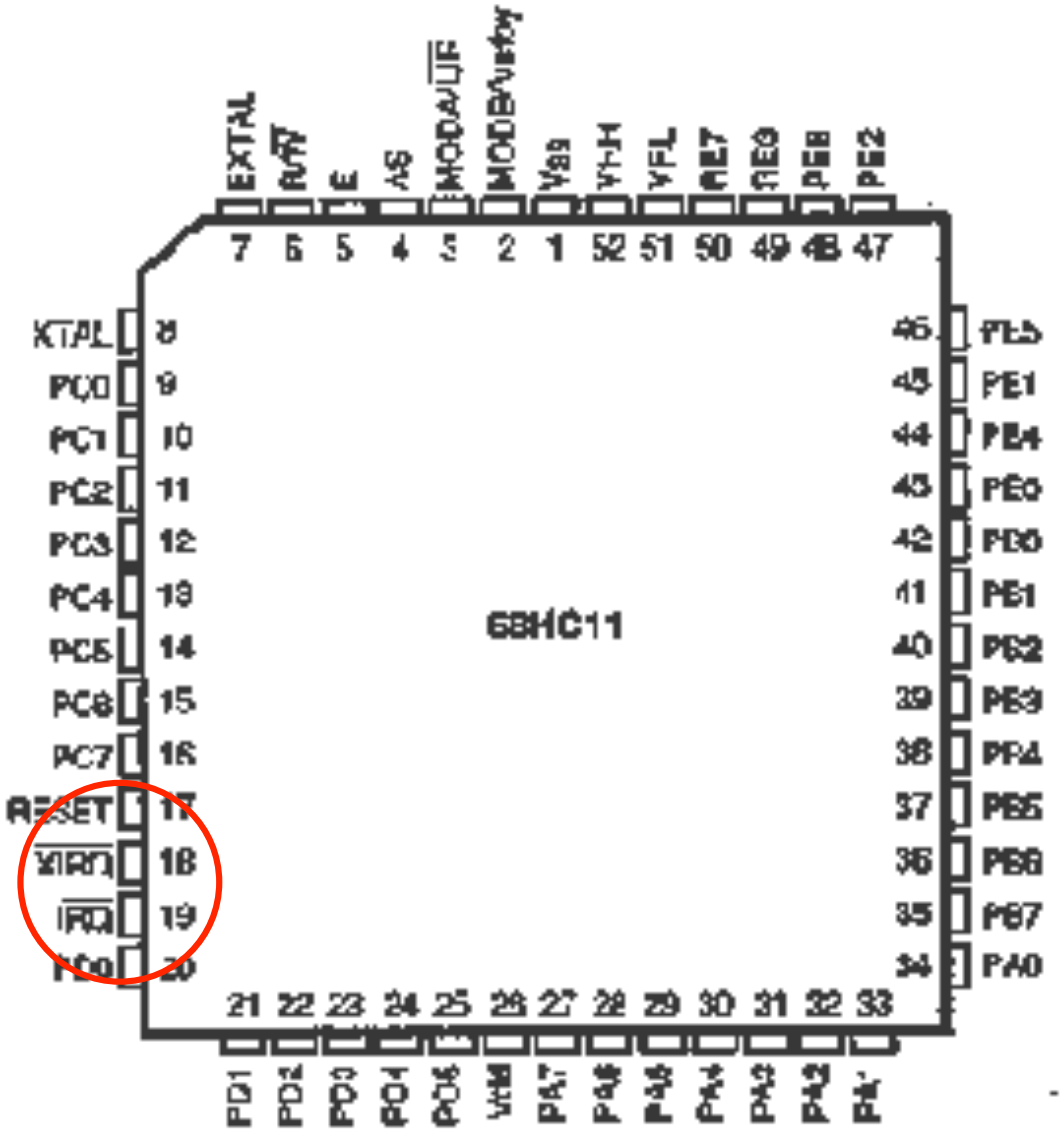
bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt



Comment cela fonctionne ?

- Est une fonction du UC -> paramétrable par le registre INTCON

- Exemple : entrées IRQ du 68HC11 : broche 19,18



- A quoi peut bien servir une IRQ dans nos programmes ?

Suppose, however, that you want to run a procedure in your loop that takes awhile. For example, perhaps you want to slowly ramp up the brightness of an LED or the speed of a motor using a `for()` loop with some `delay()` statements. If you want button presses to adjust the color or speed of such an LED fade, you will miss any presses of the button that occur while the `delay()` is happening.

Knowing the Tradeoffs Between Polling and Interrupting

Hardware interrupts are an alternative to repeatedly polling external inputs in `loop()`. They are not better or worse; instead, there are tradeoffs between using the two. When designing a system, you must consider all your options and choose the appropriate one for your application. This section describes the

- IRQ & Arduino ?

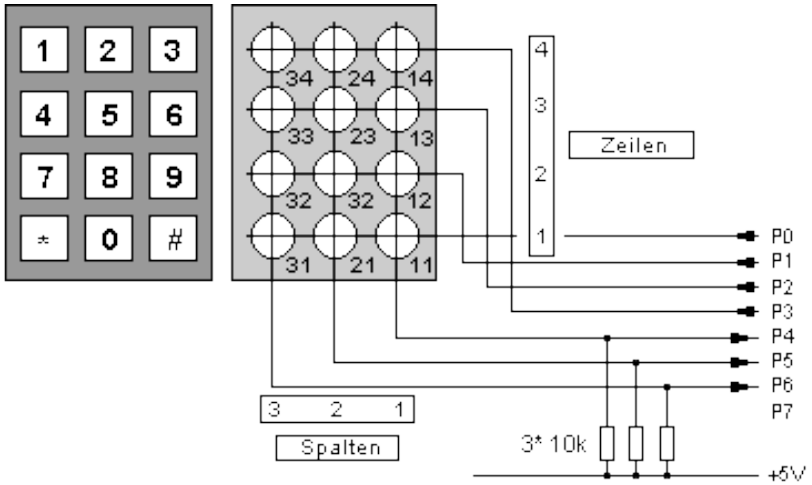
With most Arduino boards, you can use only certain pins as interrupts. Interrupts are referred to by an ID number that corresponds to a particular pin. The exception is the Due, on which all the pins can act as interrupts, and you reference them by pin number. If you are not using the Due, consult Table 12-1 to determine what pins on your Arduino can act as interrupts and what ID number they are.

Table 12-1: Available Hardware Interrupts on Various Arduinos

BOARD	INT 0	INT 1	INT 2	INT 3	INT 4	INT 5
Uno, Ethernet	Pin 2	Pin 3	-	-	-	-
Mega2560	Pin 2	Pin 3	Pin 21	Pin 20	Pin 19	Pin 18
Leonardo	Pin 3	Pin 2	Pin 0	Pin 1	-	-

- **Utilité : permet surtout au « matériel (logiciel aussi, ex:plantage) » de communiquer avec le µP dans cas où évènement extérieur arrive et doit être traité .**

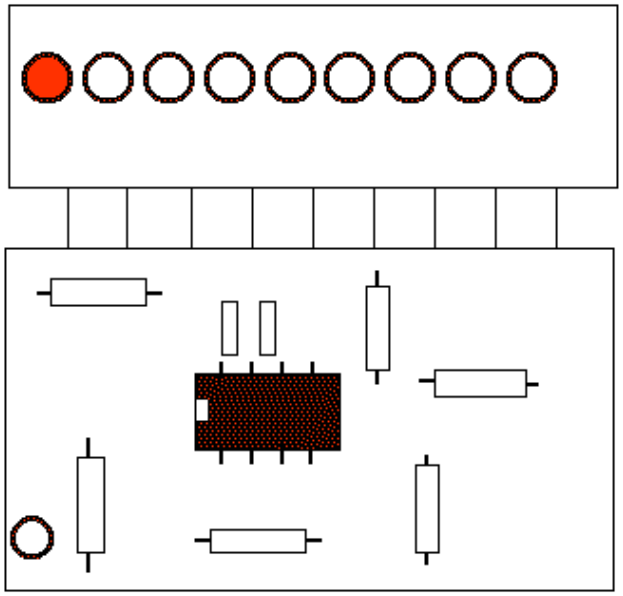
- exemples : CTRL + ALT + SUPPR, frappe au clavier, arrivée d'une donnée sur la connexion réseau, arrivée d'une information d'une carte de mesure, pression sur un bouton poussoir par l'utilisateur



- Exemple concret de la nécessité d'utiliser une interruption chenillard (⇓)

- **Cahier des charges** : nous souhaitons réaliser un chenillard qui change de sens de défilement quand l'utilisateur appuie sur un bouton

- **Évidence** : il faut un interrupteur + prise en compte de la position de l'interrupteur (= 1 interruptions « hardware » du cycle « chenillard » dans l'exemple proposé).



- **Exemple IRQ Uno Arduino :**

```
#define LED1 9
#define LED2 10
#define SW1 2
#define SW2 3

void toggle(byte pinNum) {
  byte pinState = !digitalRead(pinNum);
  digitalWrite(pinNum, pinState);
}

void setup()  {
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  digitalWrite(LED1, 0); // LED off
  digitalWrite(LED2, 0); // LED off
  pinMode(SW1, INPUT);
  pinMode(SW2, INPUT);
  attachInterrupt(0, ISR0, FALLING);
  // interrupt 0 digital pin 2 connected SW0
  attachInterrupt(1, ISR1, RISING);
  // interrupt 1 digital pin 3 connected SW1
}
```

```
void loop() {
  // do nothing
}

// can't use delay(x) in IRQ routine
void ISR0() {
  toggle(LED1);
}

void ISR1() {
  toggle(LED2);
}
```

- Que fait le uC « en temps normal » ?
- Quelles sont les pates de l'ATMEGA328 qui peuvent détecter une IRQ ?
- Comment fonctionne toogle ?
- Comment fonctionne le programme ?