S34PH412 : Systèmes Microprogrammés

Architecture des Ordinateurs

Cours 6

Université de la Réunion

Année 2015/2016

Alicalapa F.

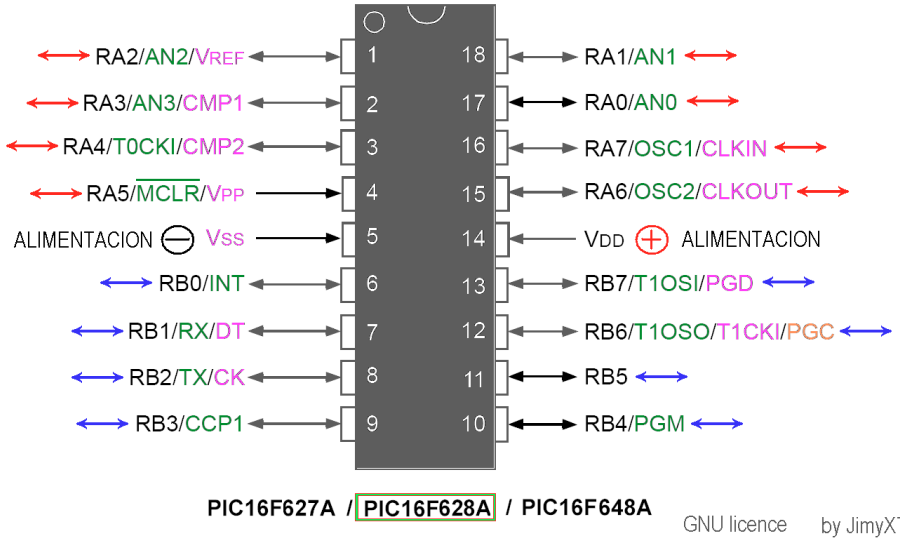
UNIVERSITÉ DE LA RÉUNION
S'ouvrir aux mondes

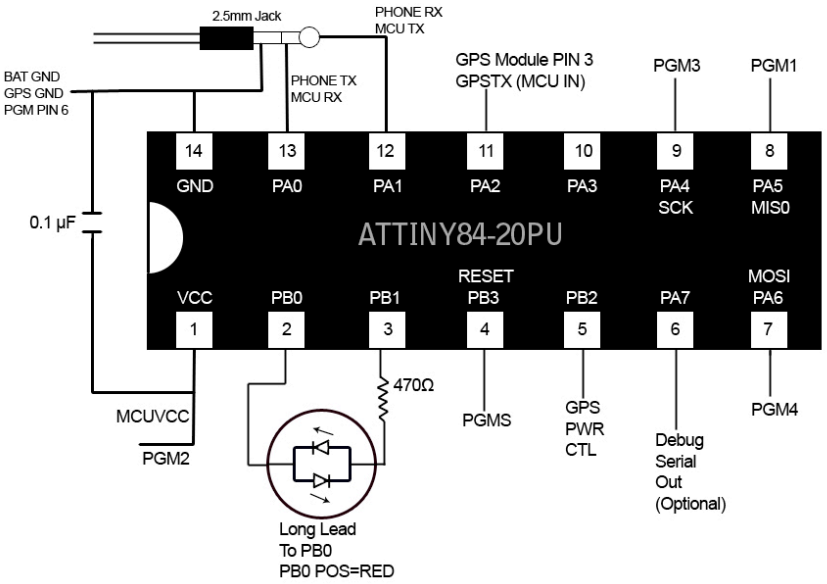
UFR Sciences et Technologies UNIVERSITÉ DE LA RÉUNION

ETUDE DE CAS 1-CM4 :

1. Zéro ou un en sortie ?

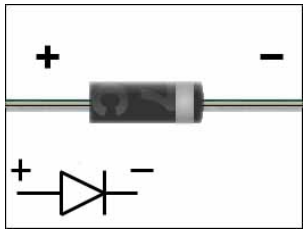
Quand placer un 1 ou un 0 en sortie (sur une patte de sortie) d'un µC ?

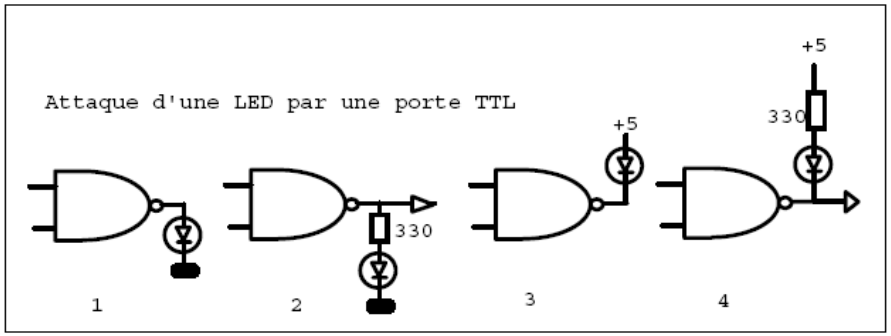


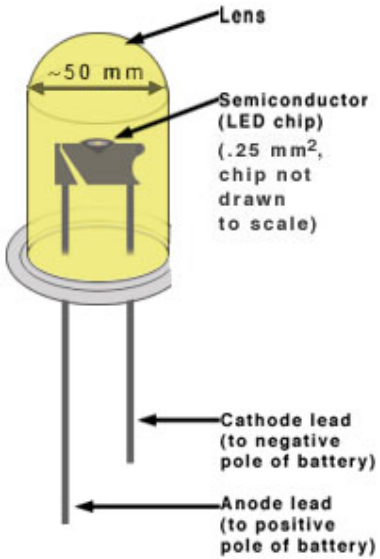


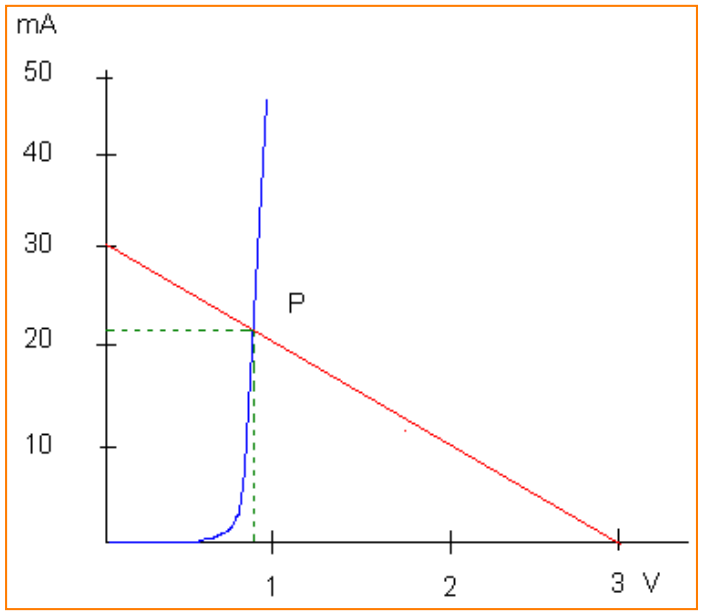
Quand placer un 1 ou un 0 en sortie d'un µC ?

- **Si la sortie est une LED : Quand** devons nous mettre un 0 ou un 1 pour allumer une LED ?
  - Cela dépend du câblage de l'élément à allumer (LED ou autre) ! Donc ici du câblage de la LED.





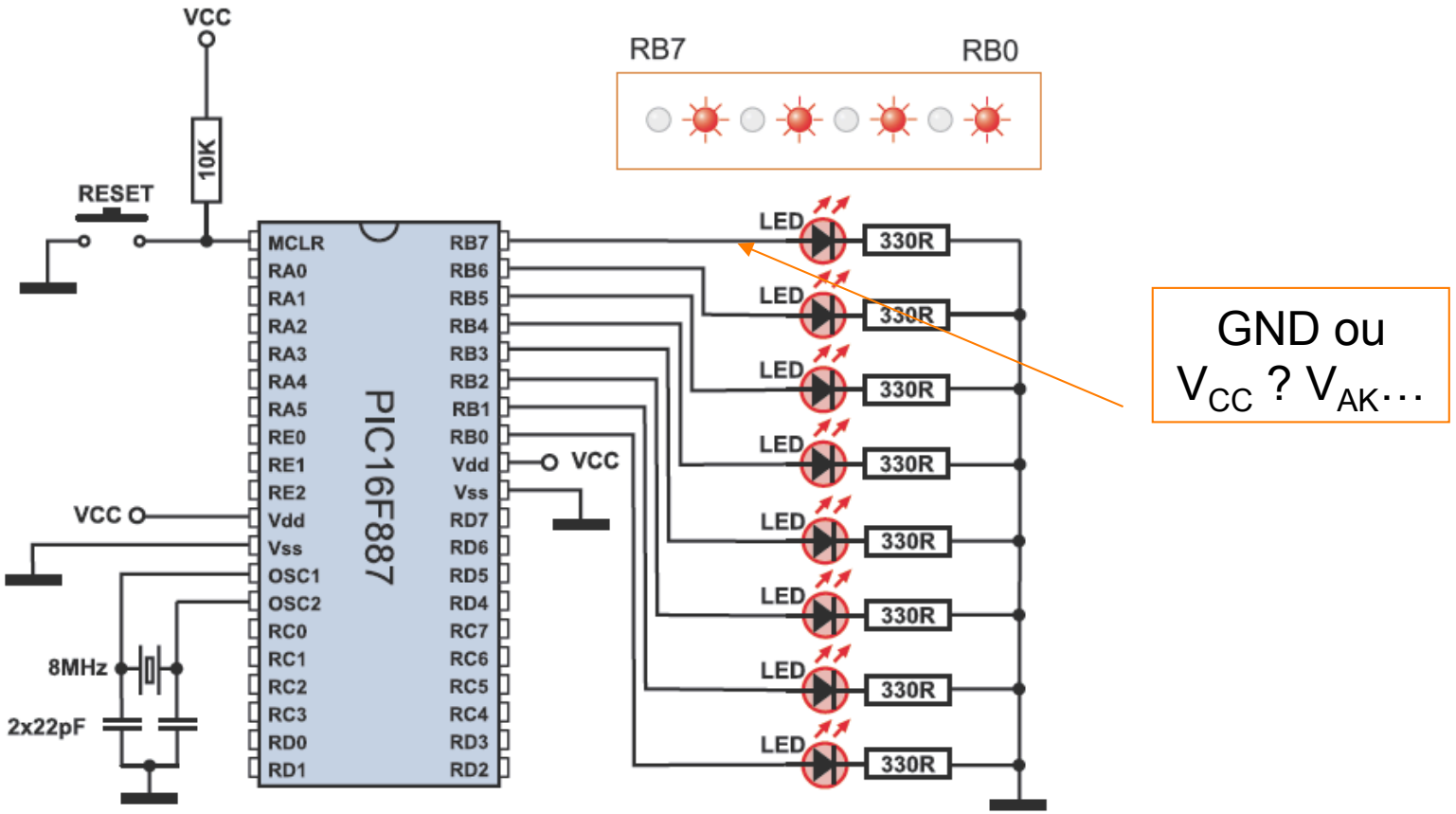




Une LED s'allume ou une « diode signal » laisse passer le courant si :
V_{AK}>0… V_{AK}>Vth

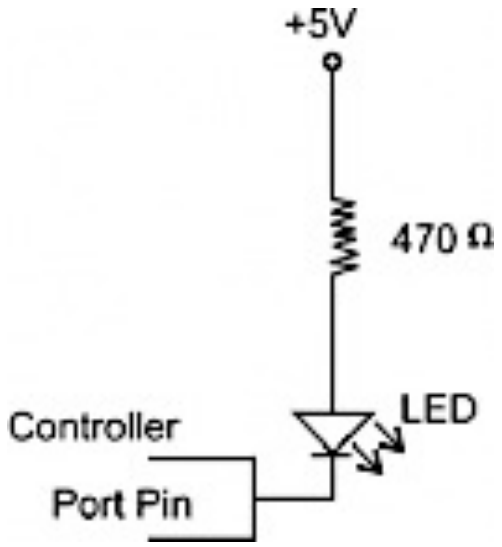
Quand placer un 1 ou un 0 en sortie d'un µC ?

- Dois je placer un 0 ou un 1 sur I/O pin pour allumer une LED ?



Quand placer un 1 ou un 0 en sortie d'un µC ?

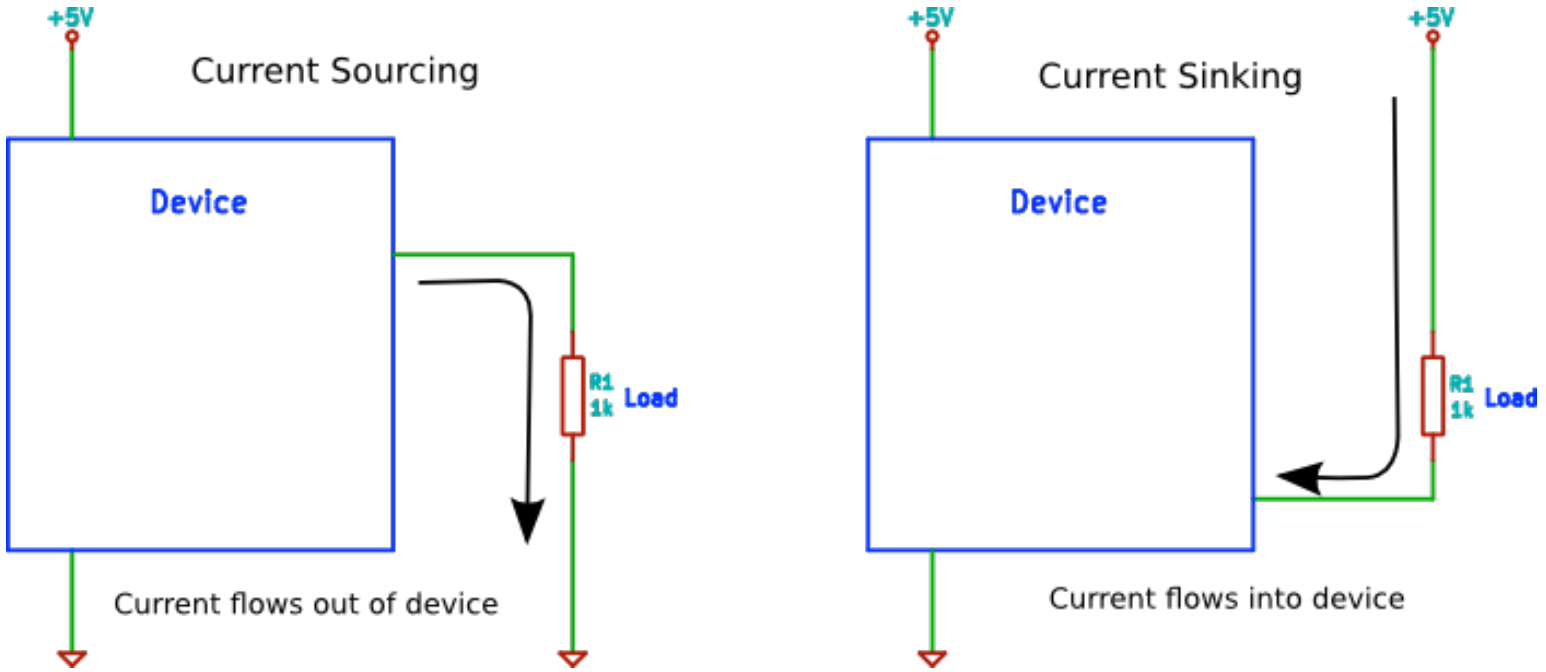
- Dois je placer un 0 ou un 1 sur I/O pin pour allumer une LED ?



- Au delà du fait de place 0 ou 1 au niveau de la sortie du microcontrôleur quelle différence au niveau courant faut-il noter ?
- Illustration (🡻)

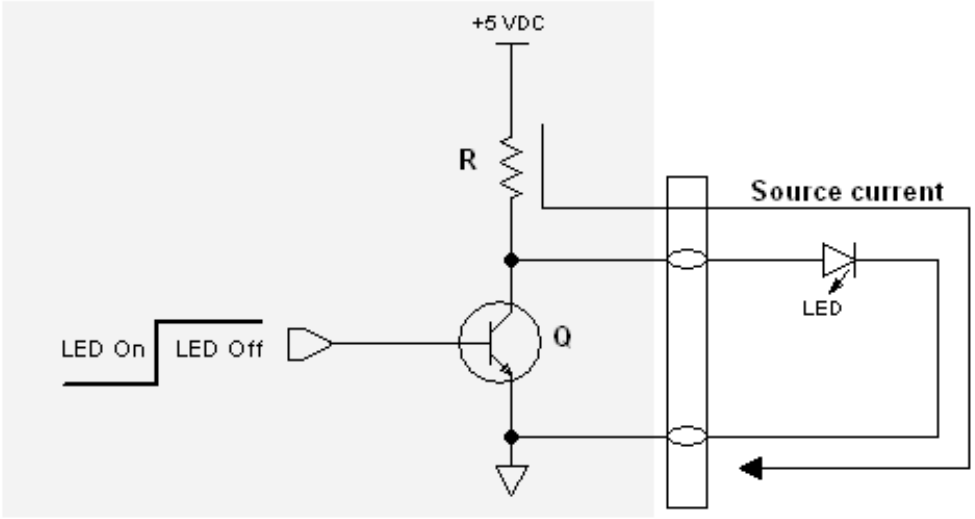
Quand placer un 1 ou un 0 en sortie d'un µC ?

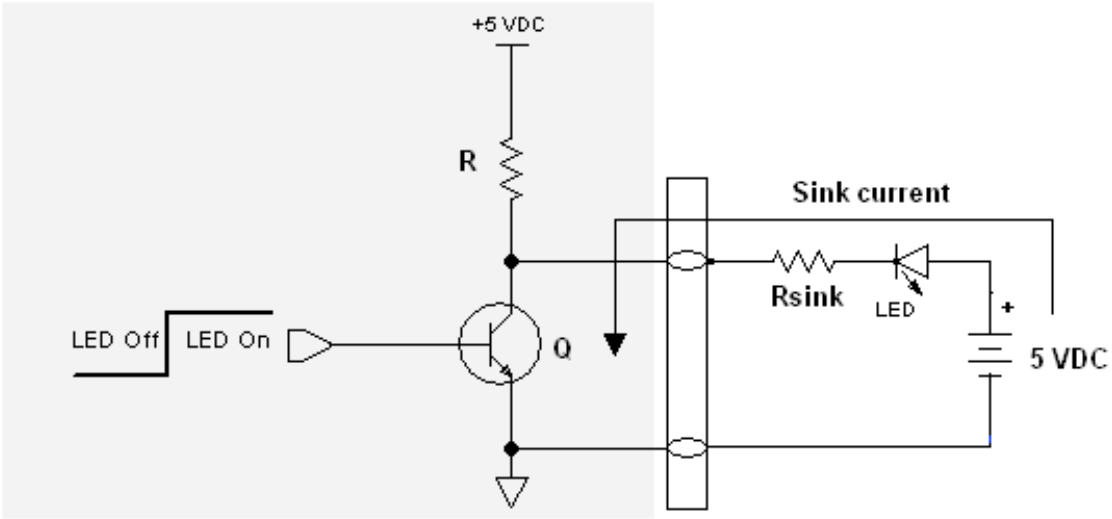
- Current sourcing and sinking



Quand placer un 1 ou un 0 en sortie d'un µC ?

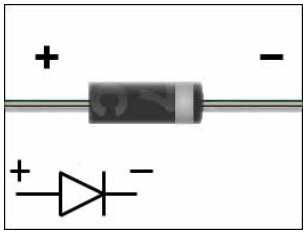
- Current sourcing and sinking

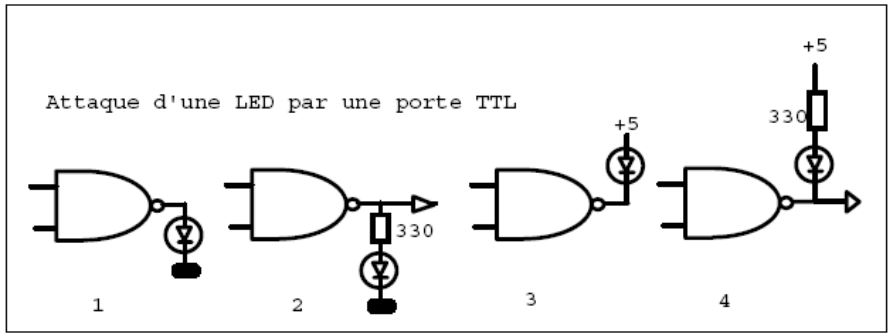




Quand placer un 1 ou un 0 en sortie d'un µC ?

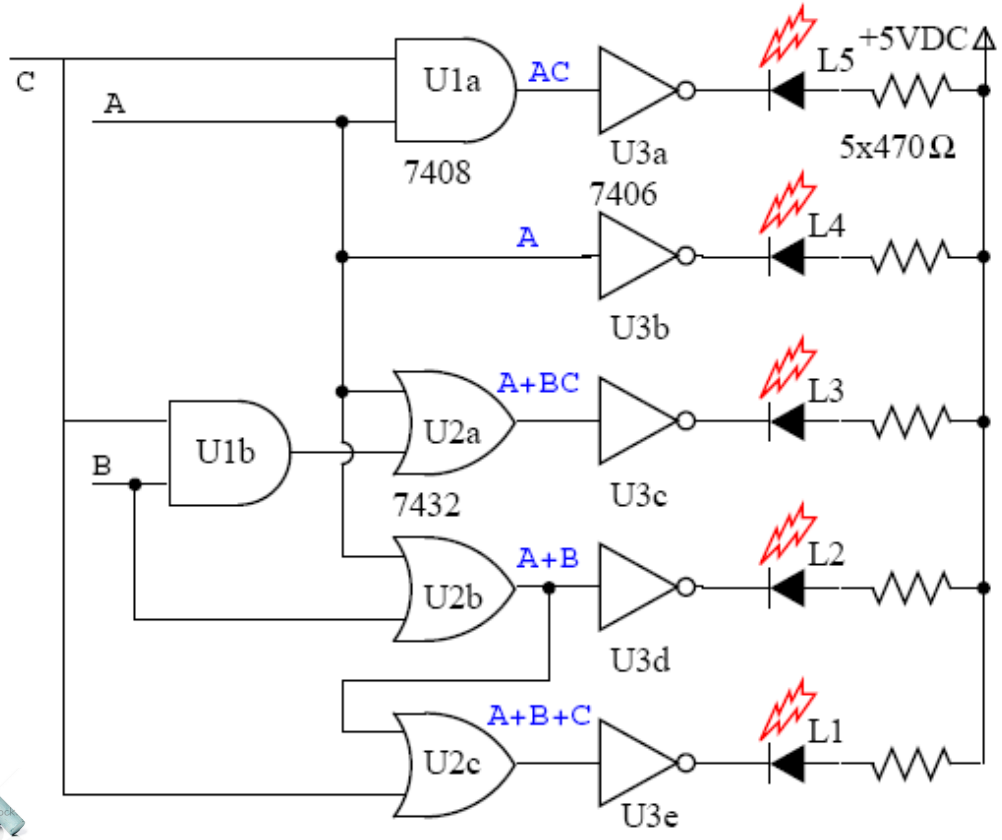
- **Notion de FAN OUT :**

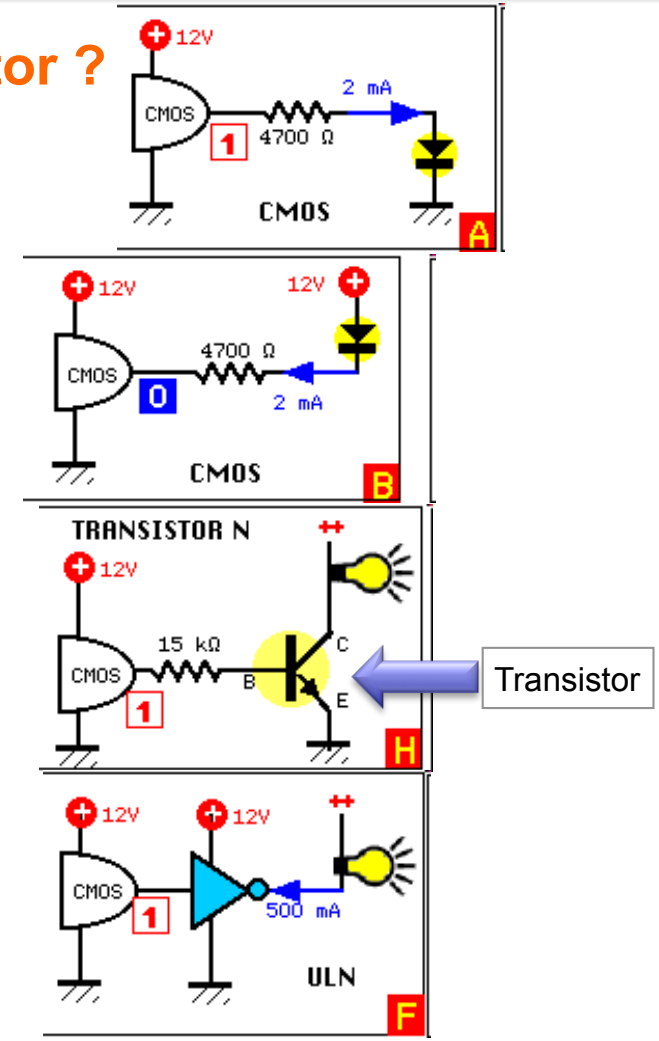


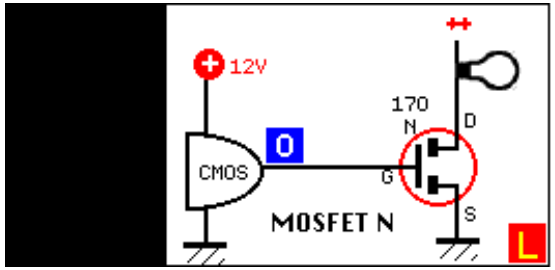


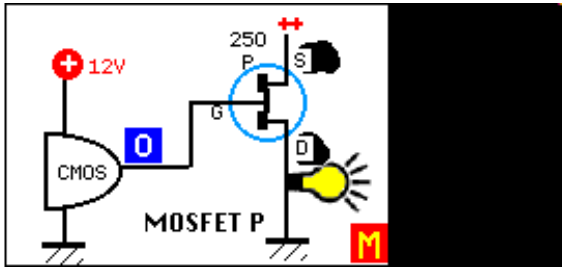
Quand placer un 1 ou un 0 en sortie d'un µC ?

- Autre exemple : **pourquoi piloter un transistor ?**









A généraliser au-delà
De la diode…moteur..
Transistor…

Quand placer un 1 ou un 0 en sortie d'un µC ?

- A retenir quand mettre 0 ou 1 en sortie quand une diode est connectée à la sortie :
  - Identifier les bornes A et K : V_{AK} doit être POSITIF pour que la diode s'allume
  - Attention au FAN OUT

DEFINITION

fan-out

Part of the Hardware glossary:

Fan-out is a term that defines the maximum number of digital inputs that the output of a single logic gate can feed. Most transistor-transistor logic (TTL) gates can feed up to 10 other digital gates or devices. Thus, a typical TTL gate has a fan-out of 10.

In some digital systems, it is necessary for a single TTL logic gate to drive more than 10 other gates or devices. When this is the case, a device called a buffer can be used between the TTL gate and the multiple devices it must drive. A buffer of this type has a fan-out of 25 to 30. A logical inverter (also called a NOT gate) can serve this function in most digital circuits.

Quand placer un 1 ou un 0 en sortie d'un µC ?

- A retenir quand mettre 0 ou 1 en sortie quand une diode est connectée à la sortie :

DEFINITION

fan-in

Like 2 Tweet 1 +1 0 Share Email

Part of the Hardware glossary:

Fan-in is a term that defines the maximum number of digital inputs that a single logic gate can accept. Most transistor-transistor logic (TTL) gates have one or two inputs, although some have more than two. A typical logic gate has a fan-in of 1 or 2.

In some digital systems, it is necessary for a single TTL logic gate to drive several devices with fan-in numbers greater than 1. If the total number of inputs a transistor-transistor logic (TTL) device must drive is greater than 10, a device called a buffer can be used between the TTL gate output and the inputs of the devices it must drive. A logical inverter (also called a NOT gate) can serve this function in most digital circuits.

Positionnement historique de l'électronique numérique

- **Quelques phrases à noter sur l'ancien cours :**
- **Bit oriented/ Byte oriented mnemonic du PIC16F690**
- **Stockage résultat d'un calcul dans W ou à une adresse f**
- **IRQ : registre de paramétrage (INTCON du PIC) à positionner en ASSEMBLEUR + circuit logique qui donne la liaison entre les différentes IRQs**
- **IRQ-Polling : répond à deux besoins :**
  - **soucis pendant l'exécution d'une instruction « Wait » ou pendant l'exécution d'une sous-fonction,**
  - **Économie d'énergie**

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**

- 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	-	-

µC µP

In this series of blogs I will be describing how to put an Arduino Diecimila into sleep mode, thus reducing the power consumption of the device, and detail several mechanisms for waking the Arduino. This can be quite useful (even necessary) when you are powering your Arduino via a battery and/or solar panel.

Four mechanisms for waking the Arduino from sleep will be covered:

- **via an external interrupt.** The Diecimila will wake up only when an external interrupt occurs.
- **via the UART** (USB serial interface). The Diecimila will remain asleep until data is received over the serial interface.
- **via an internal timer.** The Diecimila will periodically be woken up from sleep via Timer1, carry out an action and go back to sleep.
- **via the watchdog timer.** The Diecimila will periodically wake up from sleep via the Watchdog timer, carry out an action and go back to sleep. Note using the Watchdog for this provides the longest sleep time and lowest power consumption (see here).

Note: There are various versions of Arduino available (see the I/O Board section of here), we will be using the Arduino Diecimila with an ATmega168 micro-controller. The source code provided below may run on other versions of Arduino, but the power consumption values will most likely be different.

Document distribué en CM

Arduino Power Consumption

There are several devices on the Arduino Diecimila that consume battery power, including:

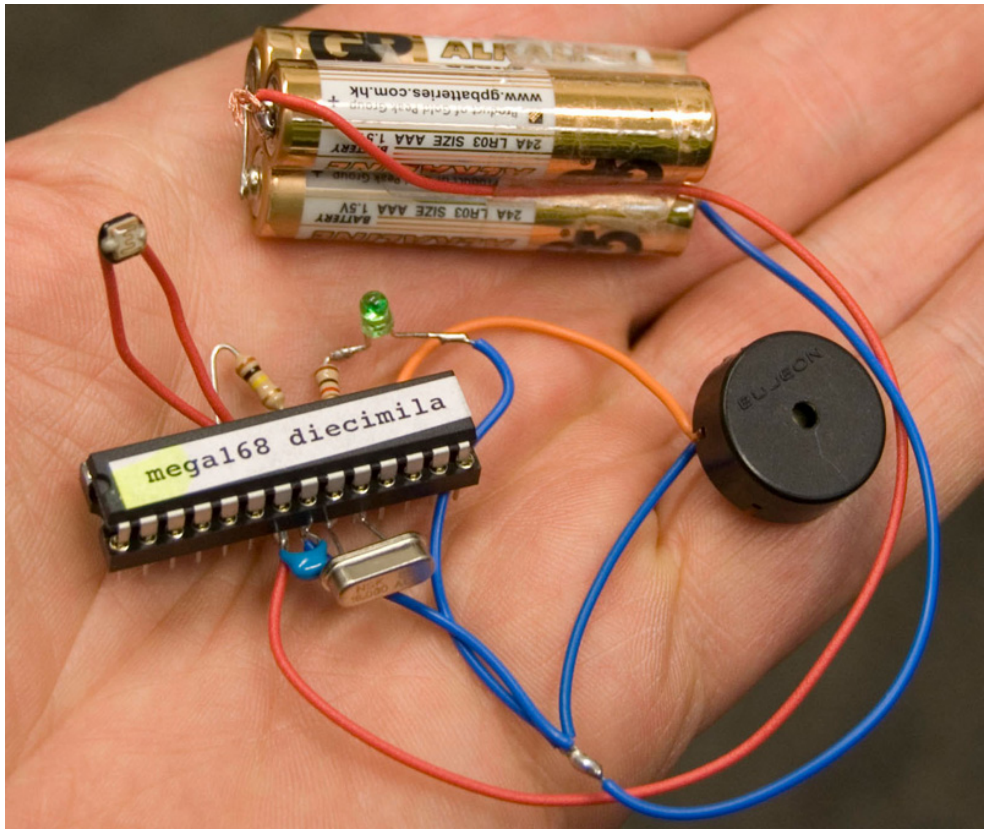
- ATmega168 micro-controller
- FT232RL USB UART
- The power regulator

The Arduino Diecimila I have uses about 35mAmps during normal operation and in power-down sleep mode about 15mAmps. There isn't a huge difference here, the main problem is that the power regulator draws 10mAmps, irrespective of the sleep state the Arduino is in. The ATmega168 micro-controller draws about 0.05 mAmps when in power-down sleep mode and 20mAmps during normal operation.

- D'autres éléments peuvent consommer de l'énergie… lesquels ?
- Une idée sur comment baisser encore plus la consommation de 15 mA (idée ⬇)

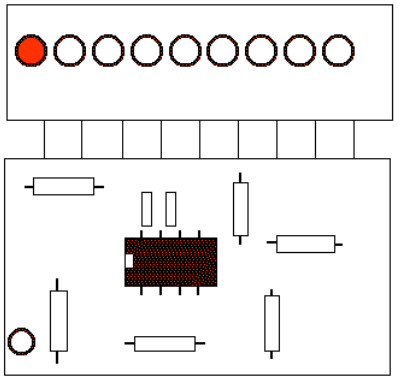
Document distribué en CM

- Une idée sur comment baisser encore plus la consommation de 15 mA (idée ⬇)



- **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). Soucis : la position de l'interrupteur ne va être prise en compte qu'à la fin du defilement des LEDS pour une programmation classique, qui lirait sa valeur en fin de cycle, pour changer de sens (si besoin)



- **Un autre 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
}
```

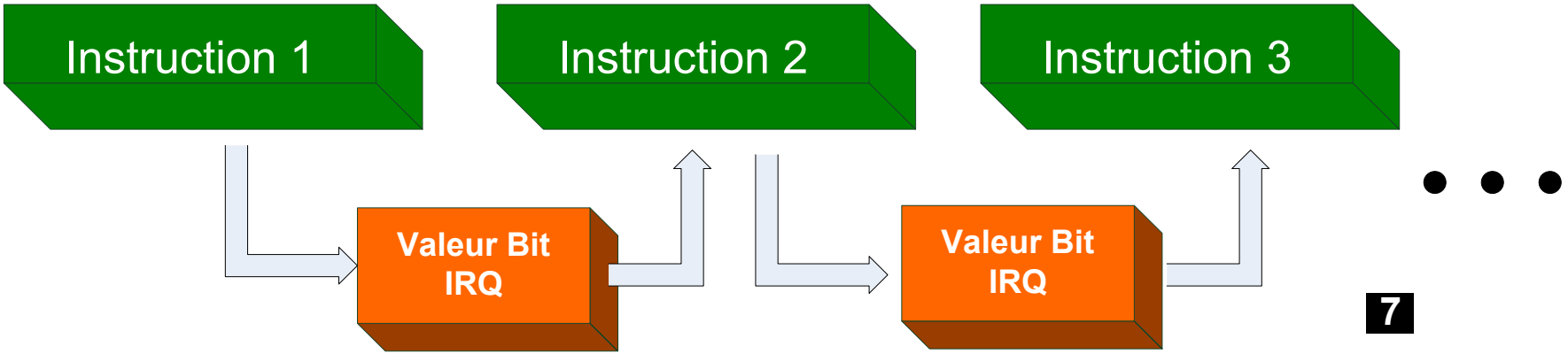
Expliquez les blocs de codes
Écrire tableau

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

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

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

- **Un autre Exemple IRQ – langage C – uC PIC**

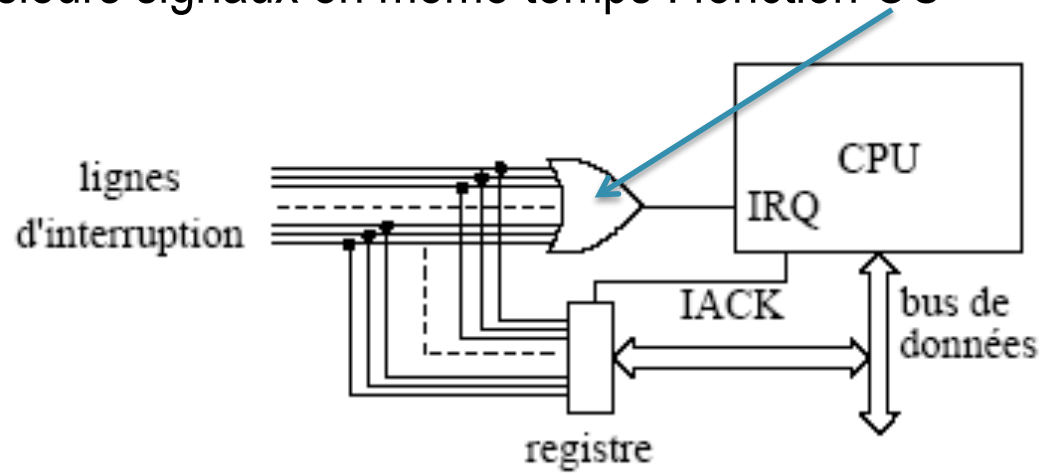
```
#include "16F877A.h"
#use delay(clock=4000000)

// ISR switches off output and waits for button **********
#int_ext
  void isrext()
  {    output_low(PIN_D0);
       delay_ms(100);
       while(input(PIN_B0));
  }

// Main block initializes interrupt and waits for button **
void main()
{
  enable_interrupts(int_ext);
  enable_interrupts(global);
  ext_int_edge(L_TO_H);

  // Assembler block outputs high speed pulse wave *******
  #asm

    Start:
      BSF 8,0
      BCF 8,0
      GOTO Start

  #endasm

} // End of source code ***********************************
```

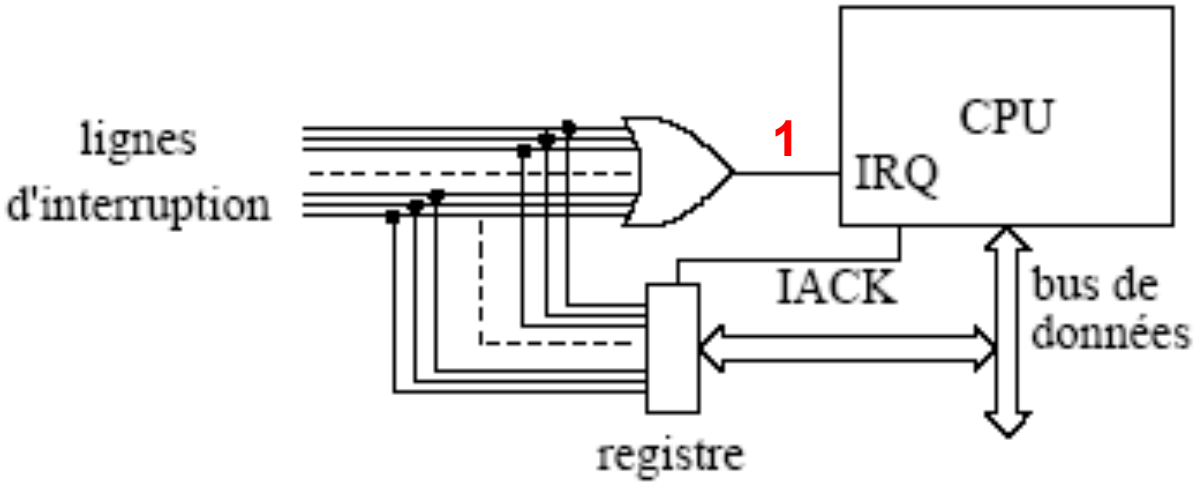
- Principe de la détection d'interruption par **scrutation** d'un digit particulier : Avant de passer à « l'instruction suivante », le µP vérifie l'état de ce bit pour interrompre son action au cas où.



- Scrutation possible de plusieurs signaux en même temps : fonction OU

Méthode dite de la **scrutation**

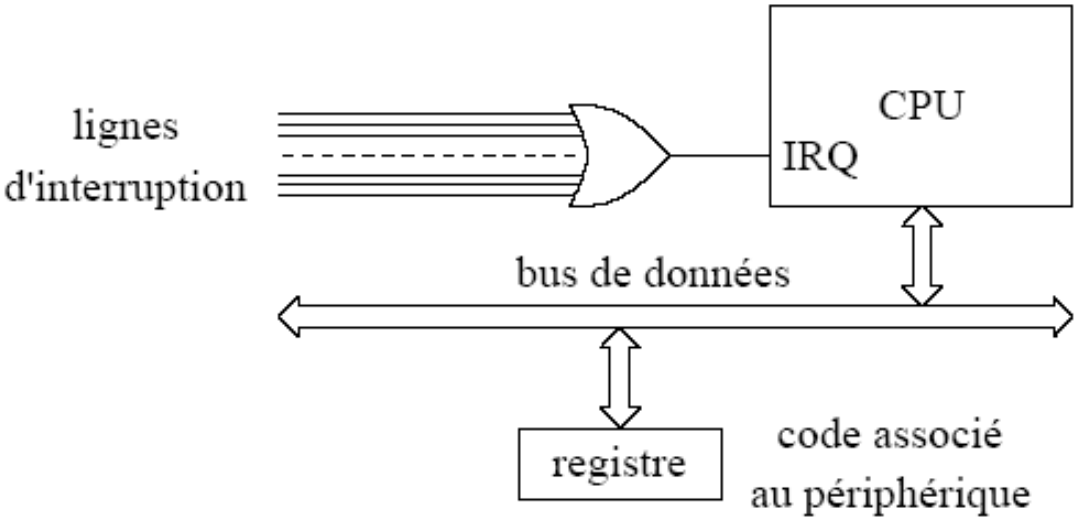




- Quand CPU détecte IRQ=1
- Sauvegarde de « l'état d'exécution » du programme en cours
- Émission d'un signal IACK qui charge en registre l'état des lignes, identification du demandeur d'IRQ et exécution d'un programme préparé à l'avance et spécifique au périphérique demandeur.
- Autre technique de détection d'IRQ : « *l'identification directe* » (⇩), le périphérique envoie un « code » directement au µP

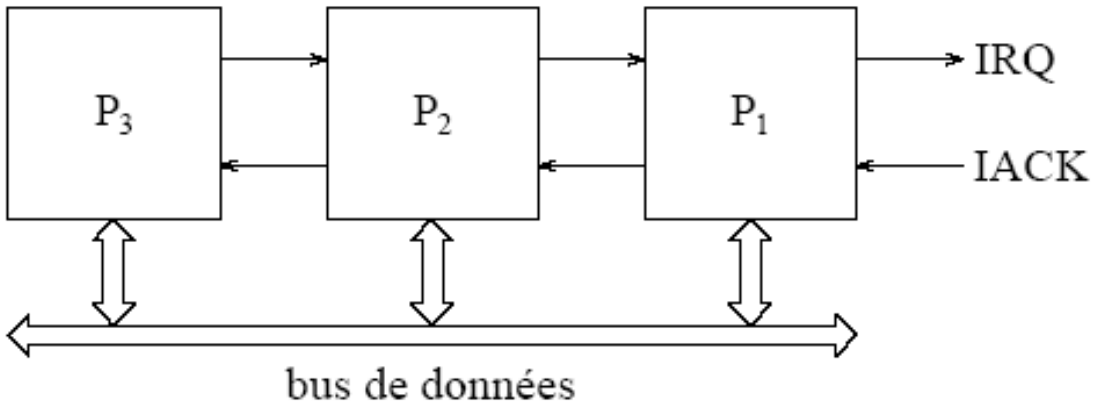
- « *l'identification directe* » (suite)
  - Envoi d'un code sur le bus de données

1



  - code utilisé pour exécuter une routine de service spécifique au périphérique « envoyeur » de l'IRQ.
  - Très souvent code envoyé défini sur le périphérique par un ensemble de petits interrupteurs à positionner au moment de son installation.

- « *l'identification directe* » (suite)
- Notion de « daisy chain » : pour éviter les conflits entre périphériques émettant des interruptions



- La priorité est définie par la position du périphérique sur la ligne d'interruption (daisy-chain ou chaînage de priorité).

Par exemple :

- le périphérique P3 ne peut faire passer sa demande (IRQ sur bus données) que si P1 et P2 n'ont aucune demande en cours.
- Dans le cas contraire le demandeur en amont empêche la transmission du signal de reconnaissance (IACK) émis par le CPU et qui autorise l'écriture du code sur le bus. La routine de service pour P3 pourra être interrompue à tout moment par une requête de P1 ou P2.

- VOCABULAIRE Catégories :

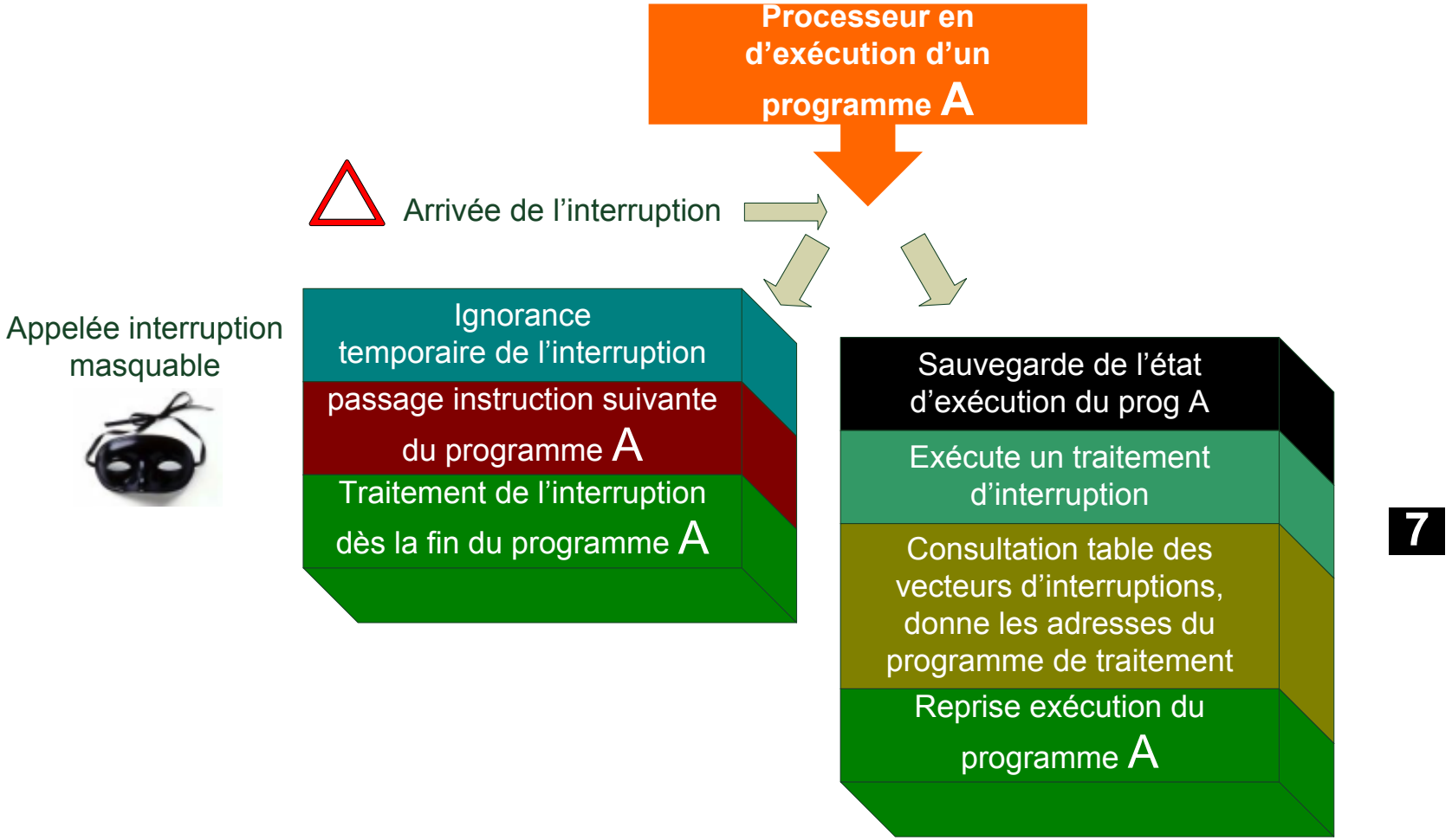
on rencontre interruptions

- Externes matérielles : vient d'un périphérique « externe au système » (imprimante, capteur, système d'alarme)

- Externes logicielles : vient du programme en cours d'exécution par exemple pour bénéficier des privilèges « SYSTEM ».

- Internes appelées « exceptions » : vient du uP pour signaler des erreurs lors de l'exécution d'un instruction (division par 0, erreur d'adressage, overflow…)

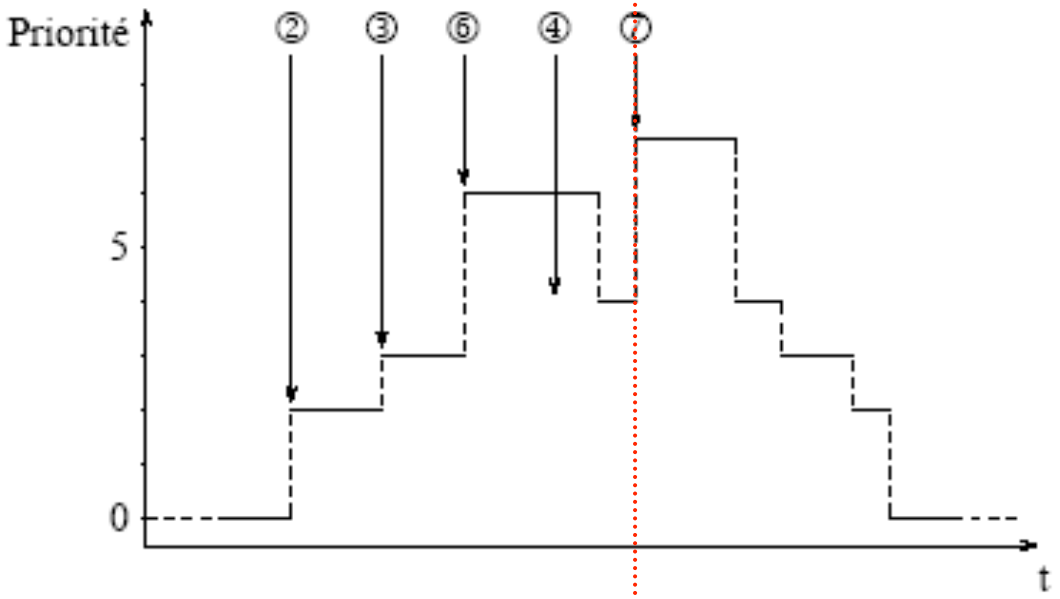
- Autre vocabulaire et principe : MASQUABLE



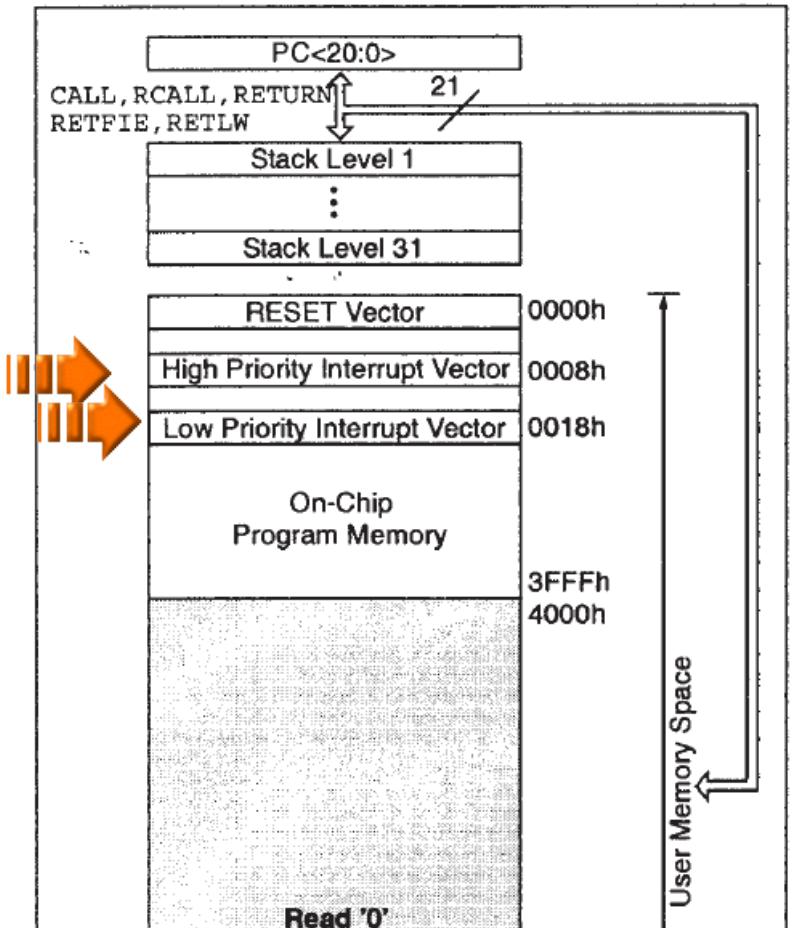
7

- Voyons apparaître la notion de « priorité d'interruption ».

- notion de « priorité d'interruption ».



enchaînement de ruptures de séquence provoquées par des interruptions hiérarchisées

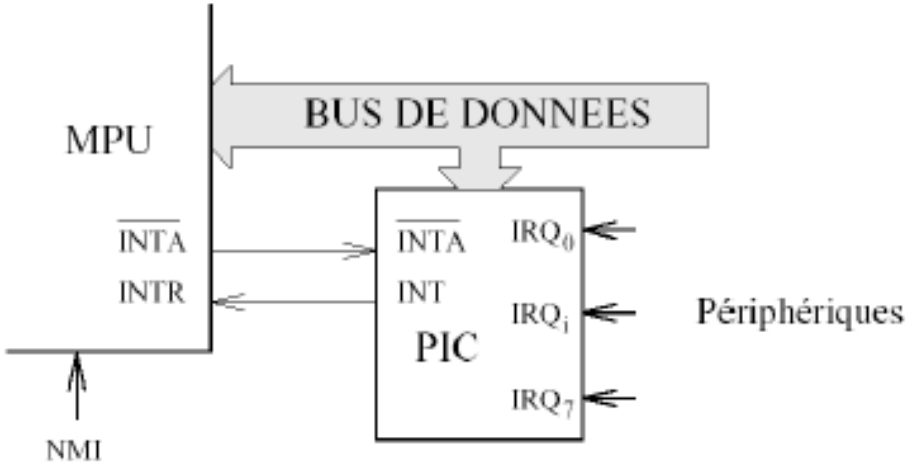


Sur la notion de priorité des interruptions, il peut exister des entrées spéciales au niveau du processeur :

- NMI utilisée pour envoyer au P une interruption non masquable (NMI, Non Maskable Interrupt). signal normalement utilisé pour détecter des erreurs matérielles (mémoire principale défaillante par exemple).
- INTR (Interrupt Request), demande d'interruption masquable. Utilisée pour indiquer au µP l'arrivée d'une interruption.
- INTA (Interrupt Acknowledge) : cette borne est mise à 0 lorsque le processeur traite effectivement l'interruption signalée par INTR (c'est à dire qu'elle n'est plus masquée).
- IF (Interrupt Flag = « drapeau »): les interruptions sont soit masquées soit autorisées, suivant l'état d'un indicateur spécial du registre d'état, champ IF.
  - si IF = 1, le processeur accepte les demandes d'interruptions masquables, c'est à dire qu'il les traite immédiatement ;
  - si IF = 0, le processeur ignore ces interruptions.

- Sur la notion de priorité des interruptions (suite) :
- Un **contrôleur d'interruption** peut aussi gérer les différentes requêtes d'interruption : Le contrôleur d'interruptions est un circuit spécial, extérieur au processeur, dont le rôle est de distribuer et de mettre en attente les demandes d'interruptions provenant des différents périphériques.

- Il gère les demandes d'interruption envoyées par les périphériques, de façon à les envoyer une par une au processeur (via INTR). Avant d'envoyer l'interruption suivante, le contrôleur attend d'avoir reçu le signal INTAck, indiquant que le processeur a bien traité l'interruption en cours.



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
5. Autres modélisations
6. **Pipeline**

Nous avons vu que le processeur réalise de manière séquentielle les opérations suivantes :

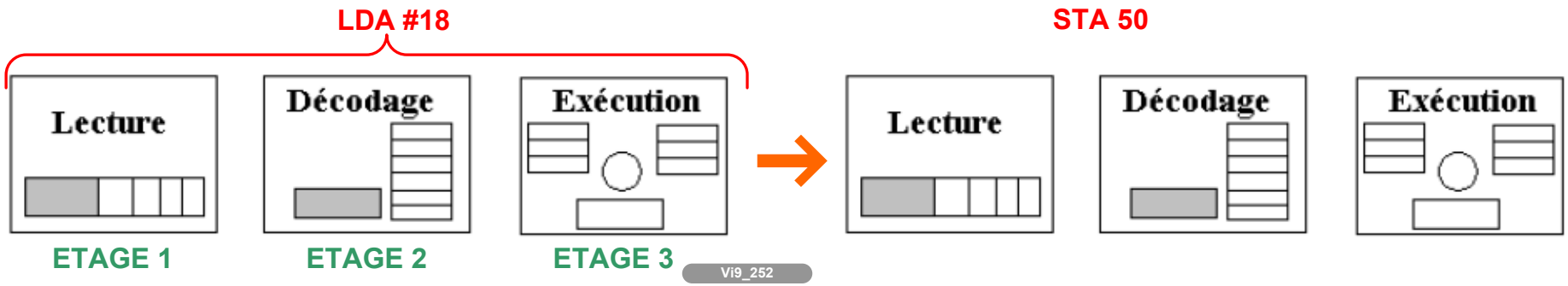
- 1. **lire en mémoire** l'instruction à exécuter
- 2. **effectuer** le traitement correspondant (calcul+stockage)
- 3. **passer** à l'instruction suivante.
- 4…….ainsi de suite

- Soit à exécuter :

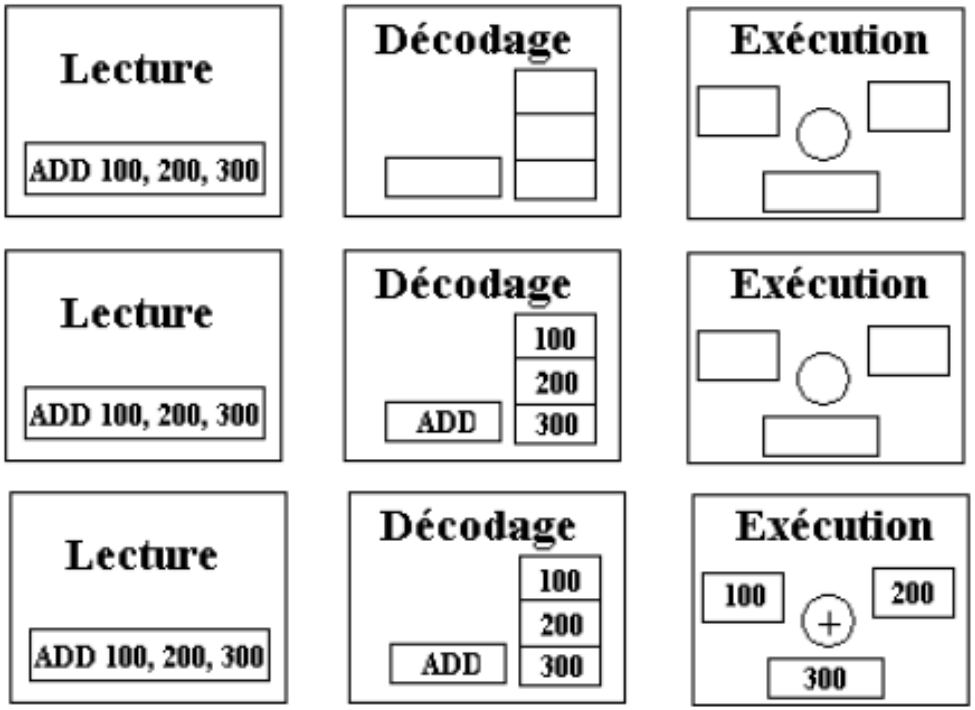
Exemple de programme en PM
LDA #18 ; *{chargement de l'accumulateur avec la valeur 18}* **STA 50** ; *{rangement de l'accumulateur dans la mémoire n°50}* **LDA #5** ; *{chargement de l'accumulateur avec la valeur 5}* **STA 51** ; *{rangement de l'accumulateur dans la mémoire n°51}* **ADD 50** ; *{addition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

Pour l'exécution : chaque instruction lue, décodée et exécutée

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



- On remarquera que :
- Pendant le temps d'activation d'un étage, les deux autres restent inactifs.
- Il faut attendre la fin du traitement de l'instruction « en cours » pour pouvoir passer au traitement de l'instruction suivante

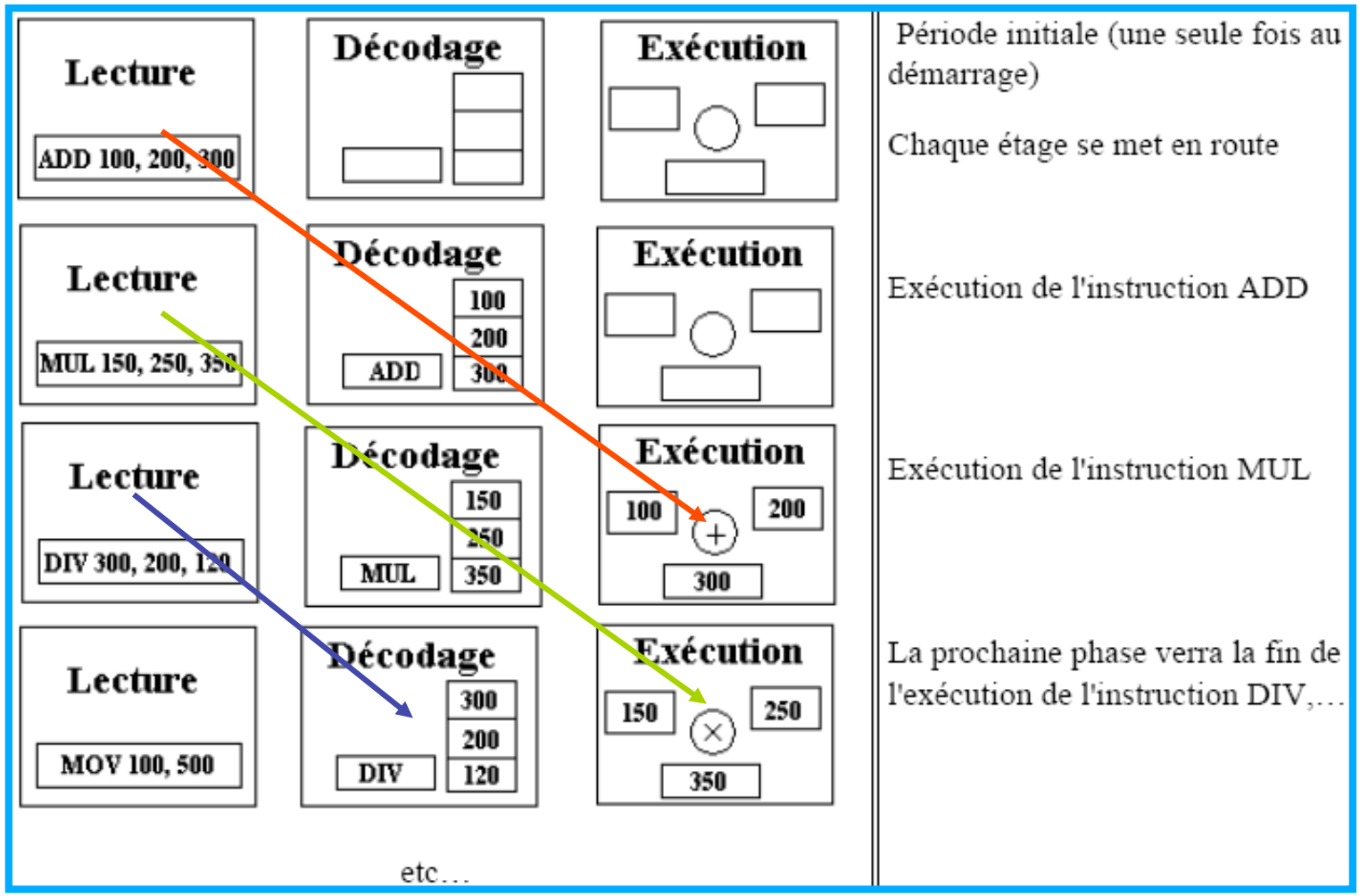


- L'architecture pipe-line = optimiser les temps d'attente de chaque étage = commencer le traitement de l'instruction suivante dès que l'étage a été libéré par l'instruction en cours
- ⇨ durant chaque phase, tous les étages sont occupés à fonctionner (chacun sur une instruction différente).

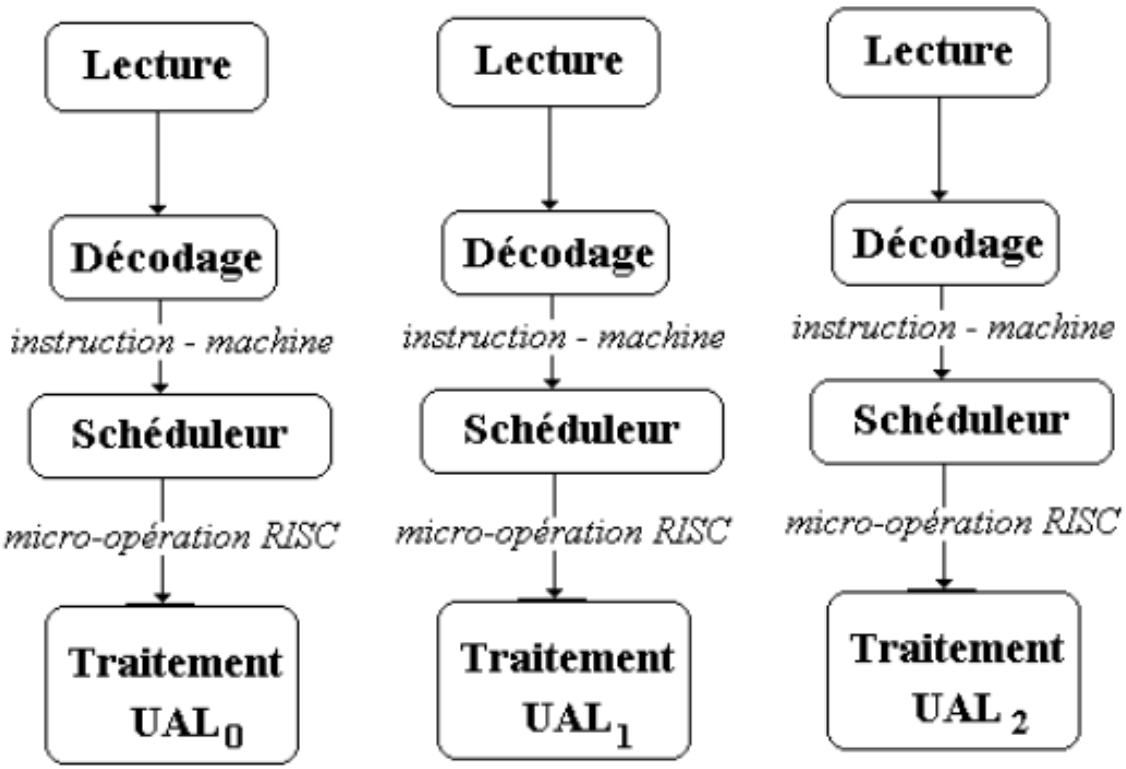
- A un instant **t0** donné
  - l'étage d'exécution travaille sur les actions à effectuer pour l'instruction de rang **n**,
  - l'étage de décodage travaille sur le décodage de l'instruction de rang **n+1**,
  - et l'étage de lecture sur la lecture de l'instruction de rang **n+2**.
- Cette technique **accélère l'éxécution d'un programme**

- Le nombre d'unités différentes constituant le pipe-line s'appelle le **nombre d'étages du pipe-line**.
- Représentation (🡻)

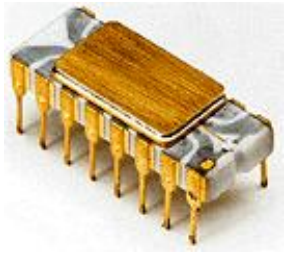
- traitement 4 instructions selon un pipe-line à 3 étages (lecture, décodage, exécution)

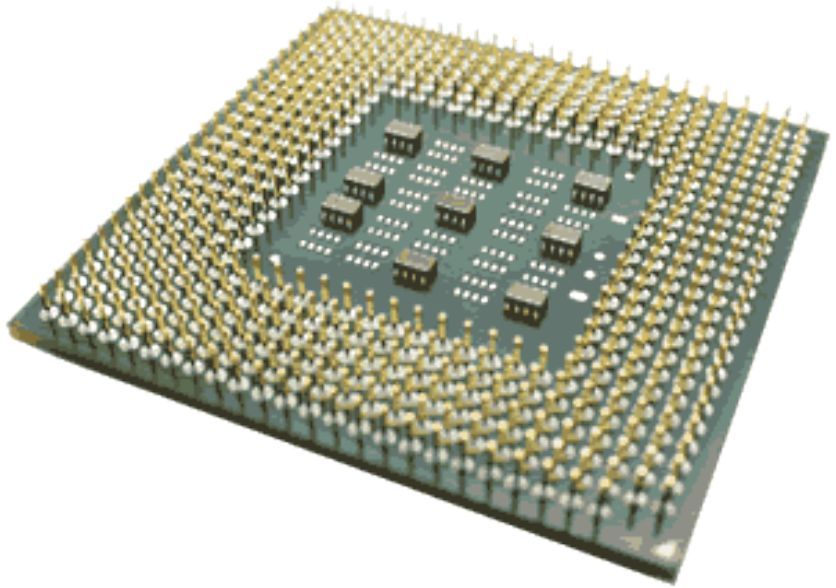


- processeur est **super-scalaire** = possède plusieurs pipe-lines indépendants dans lesquels plusieurs instructions peuvent être traitées simultanément. Dans ce type d'architecture apparaît la **notion de parallélisme**.



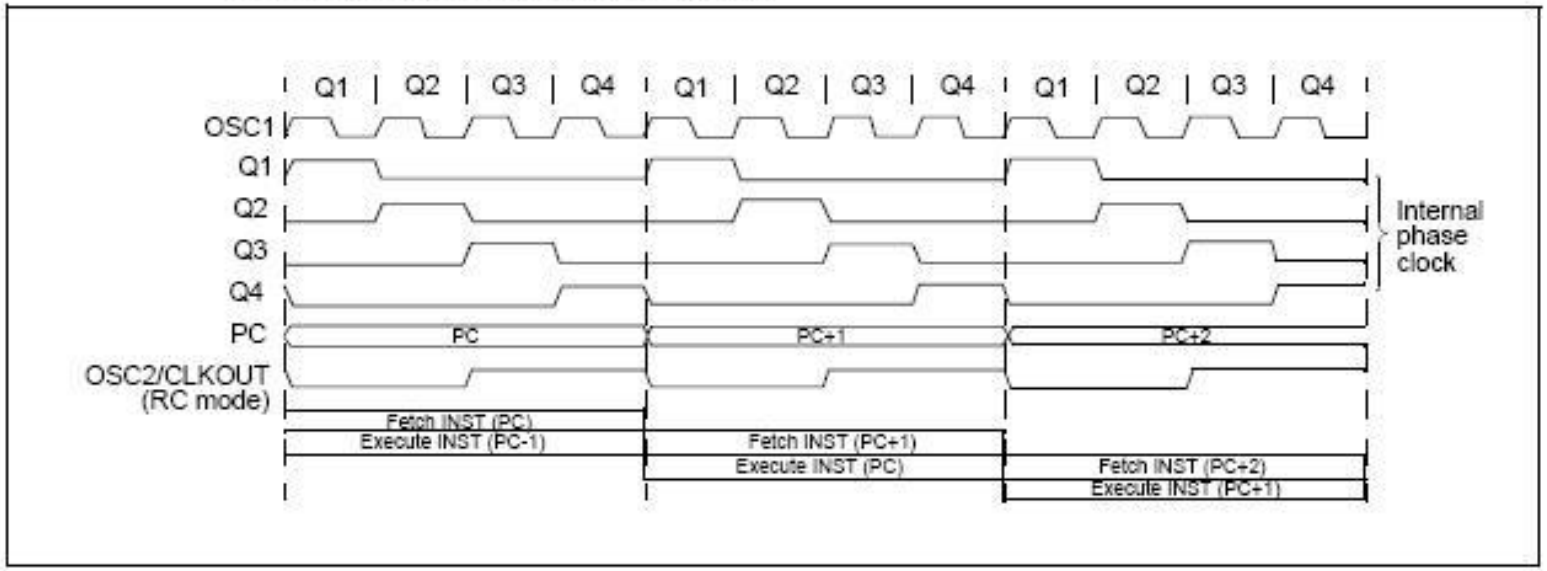
- **Le pentium IV = pipe-line à 20 étages** et pipe-line **super-scalaire**.
- **L'AMD 64 Opteron 3 pipe-lines d'exécution** identiques pour les calculs en entiers et de **3 pipe-lines spécialisés** pour les calculs en virgules flottante.



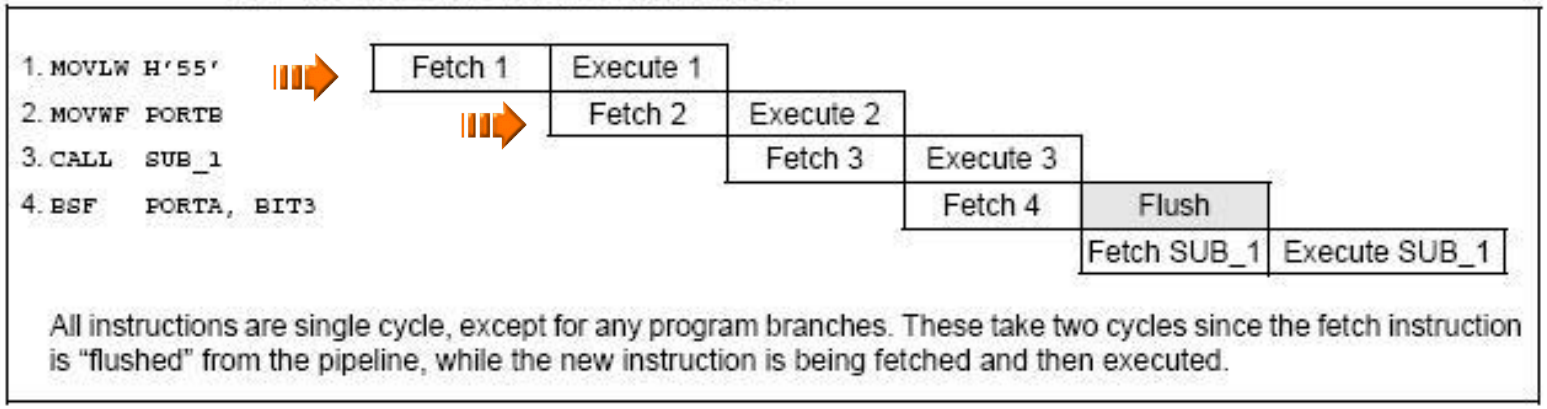


Pour le PIC16C (pipe-line 2 étages) :

PIC CLOCK/INSTRUCTION CYCLE



PIC INSTRUCTION PIPELINE FLOW



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline, while the new instruction is being fetched and then executed.

- **Vocabulaire anglais associé cette notion :**

- If, however, program memory has its own address and data bus, separate from data memory (i.e. a Harvard structure), then there is no reason why a CPU cannot be designed so that while it is executing one instruction, it is already fetching the next. This is called pipelining. Pipelining works best if fetch and execute cycles are always of the same duration, such as a RISC structure gives. This fairly simple design upgrade gives a doubling in execution speed!

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

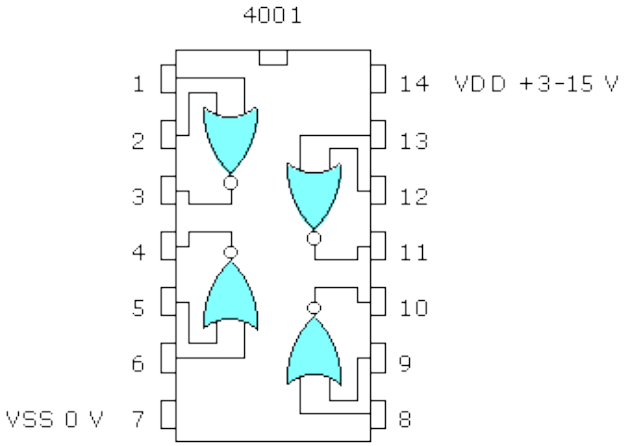
IV. Les différents composants du système et leur caractéristiques

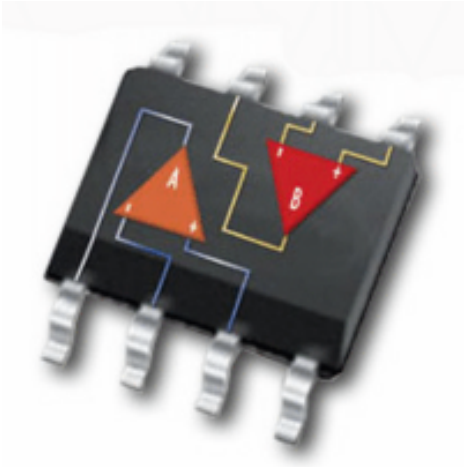
a. Les mémoires

b. Le processeur :

1. modélisation primaire
2. L'horloge
3. Jeu d'instruction
4. **Les interruptions**
5. **Les I/Os multifonctions**

- Basiquement : nous pouvons opter pour 1 patte = 1 entrée ou 1 sortie (il peut y en avoir plusieurs) d'une certaine fonction.

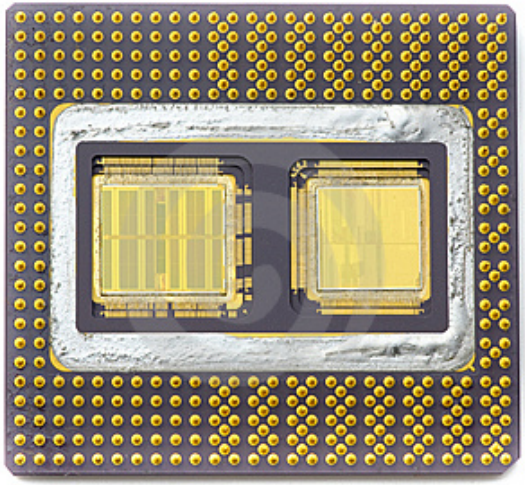


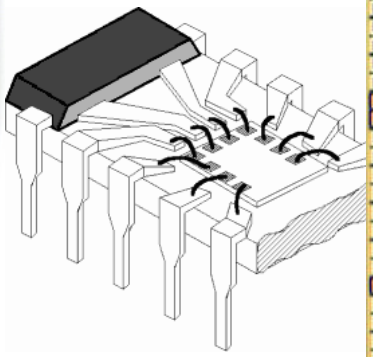


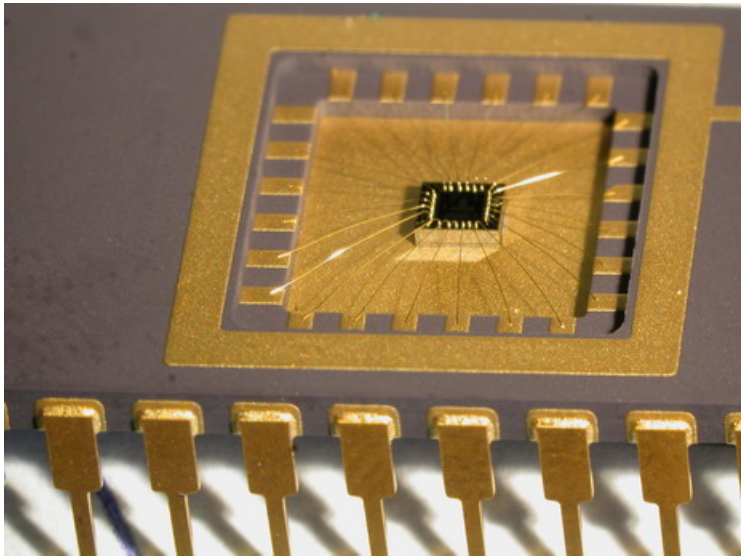
- Mais le soucis : problème de place
  - si j'ai 10 fonctions sur la puce du µC,
  - si chaque fonction réquisitionne 10 pattes ⇨ 10*10 pattes + pattes GND + pattes Vcc d'alimentation ⇨ il faut un « grand boîtier » pour avoir le nombre suffisant de pattes !

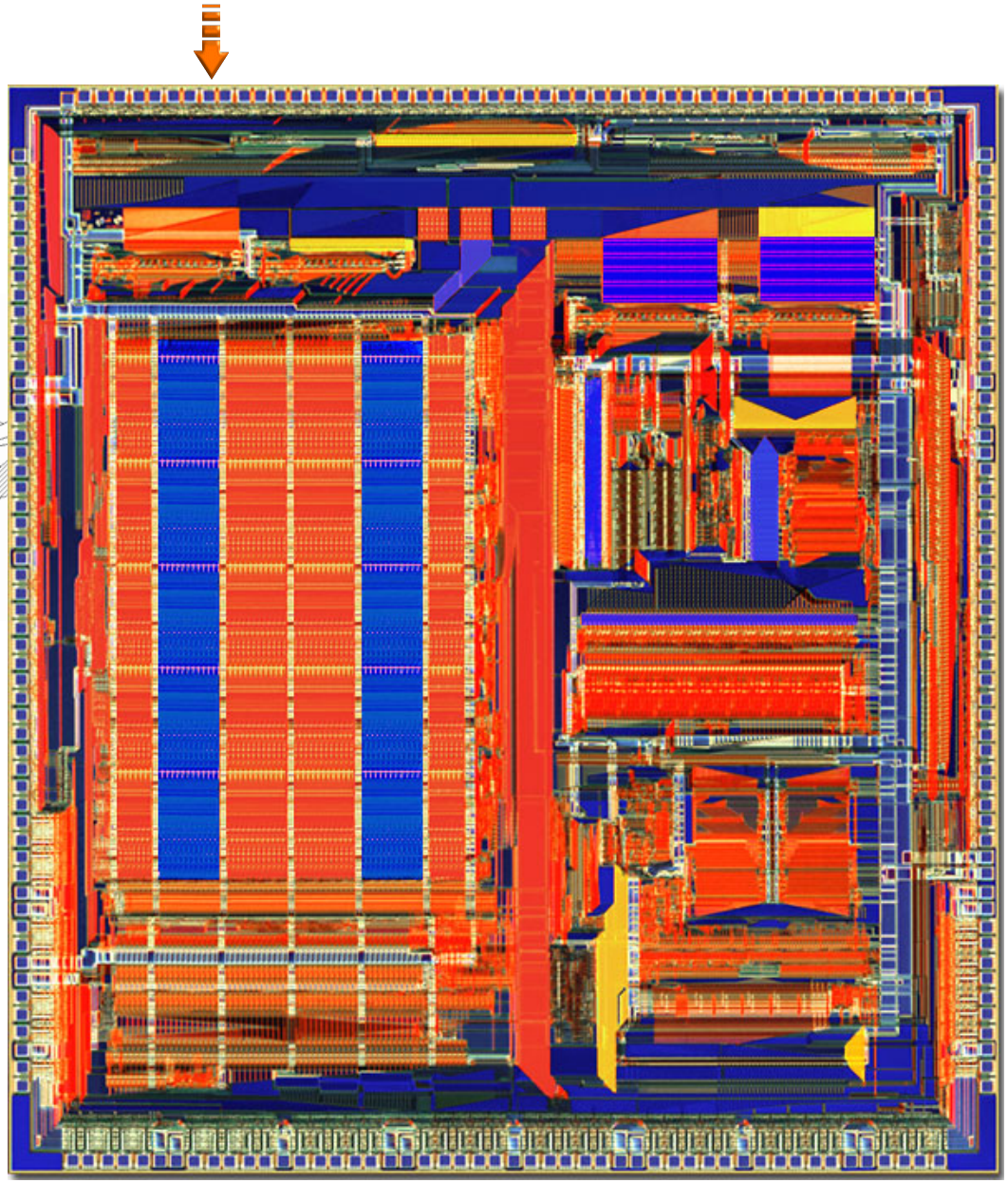
Les entrées/sorties du µP

- « grand boîtier »

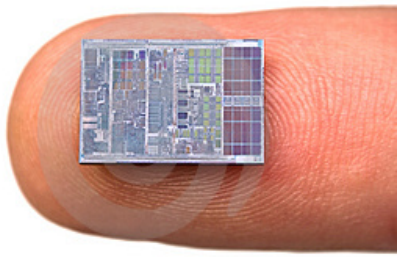


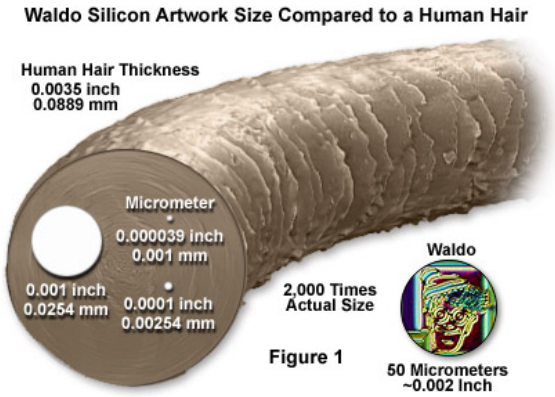


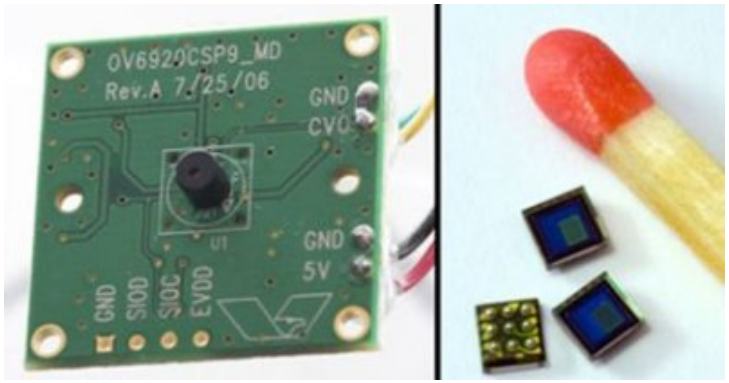




- « grand boîtier » ⇨ ceci est contraire au souhait ne MINIATURISATION des circuits et des systèmes !



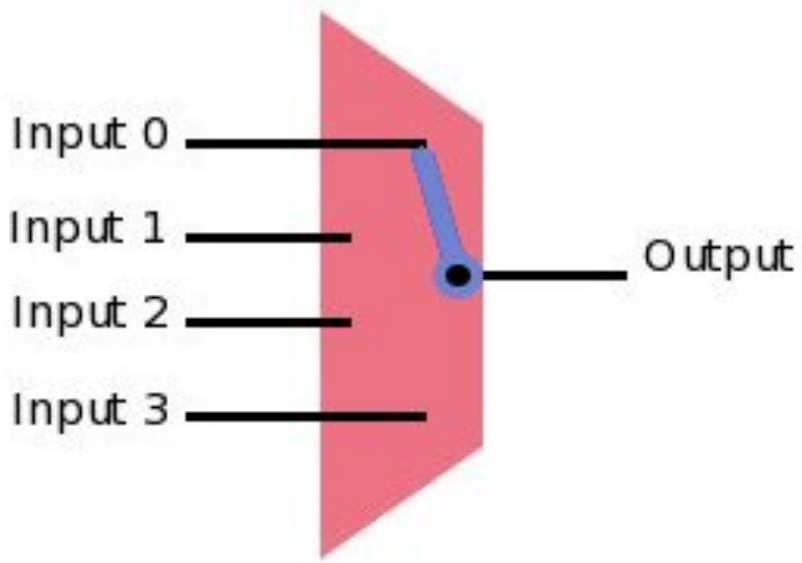




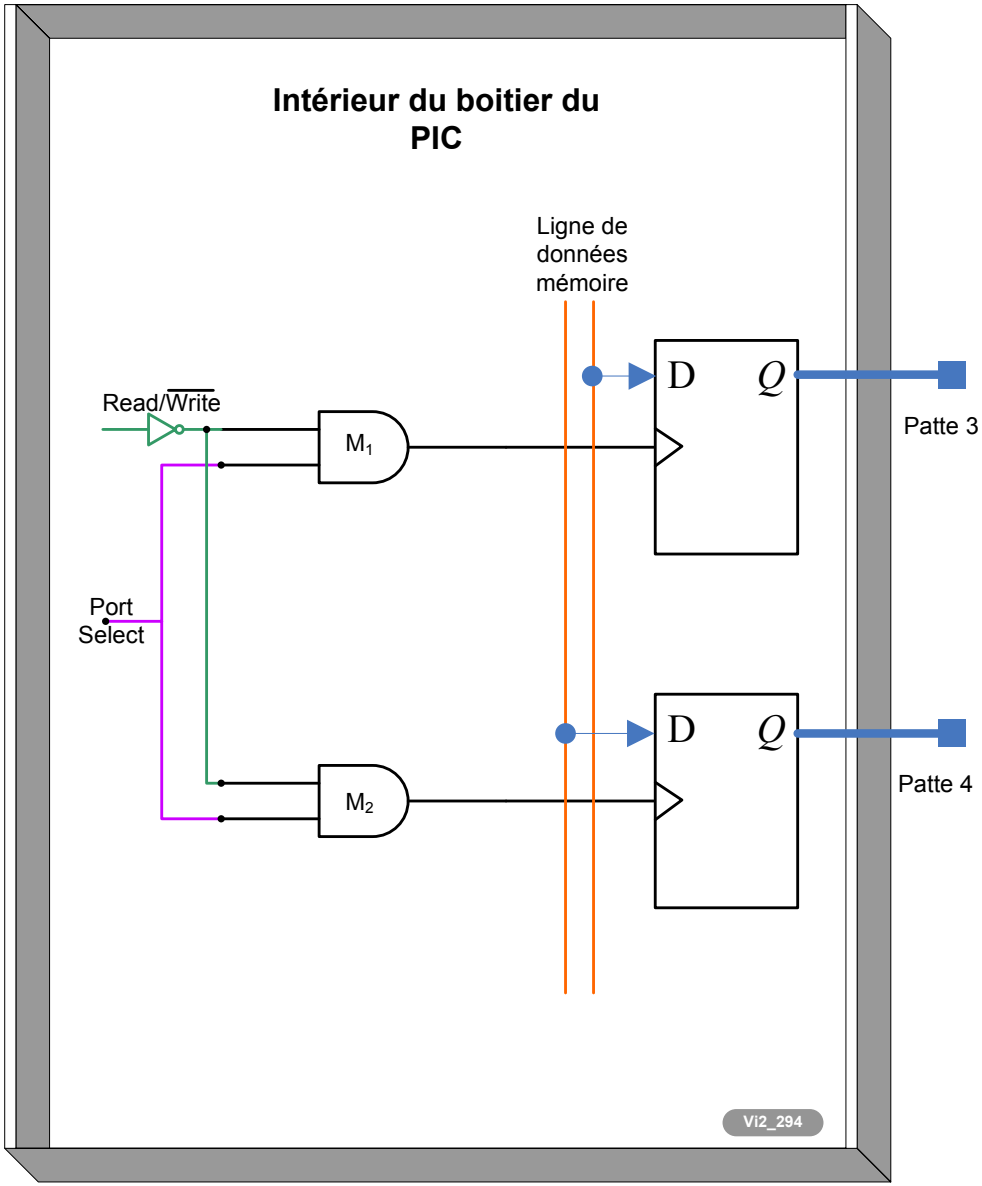
World's Smallest Digicam Is the Size of a Pin-Head : the OV6920 from TDC compared with a match-head

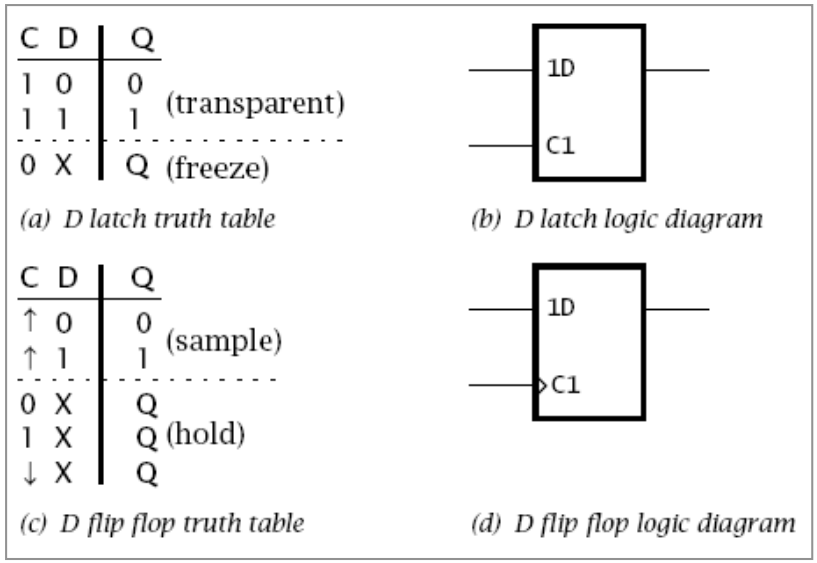
- **SOLUTION ????**

- **SOLUTION : multiplexage µC**

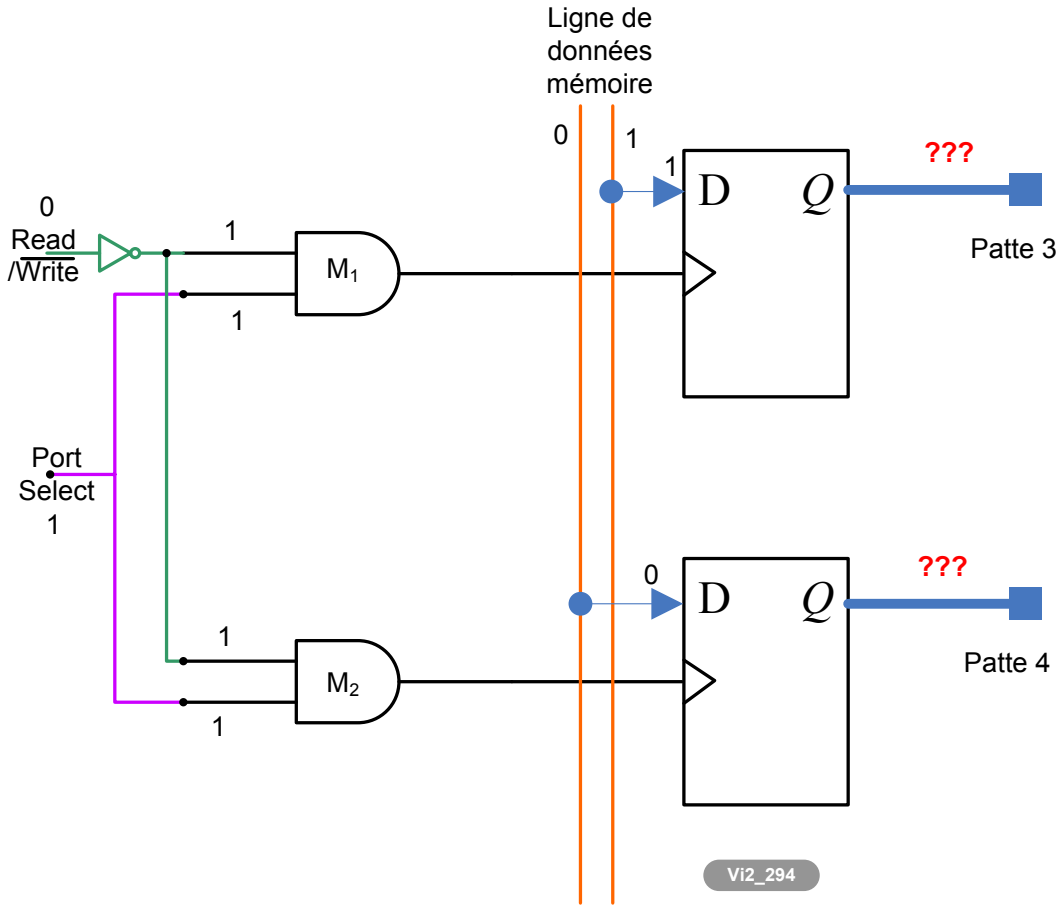


- Comment faire sortir une information numérique de la puce ???



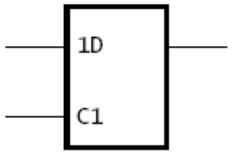


- Configuration 1 : quelles sont les valeurs sur les pattes de sortie ?
- Les bascules ici sont des bascules D LATCH !



C	D	Q	
1	0	0	(transparent)
1	1	1	
0	X	Q	(freeze)

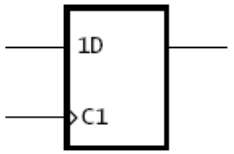
(a) D latch truth table



(b) D latch logic diagram

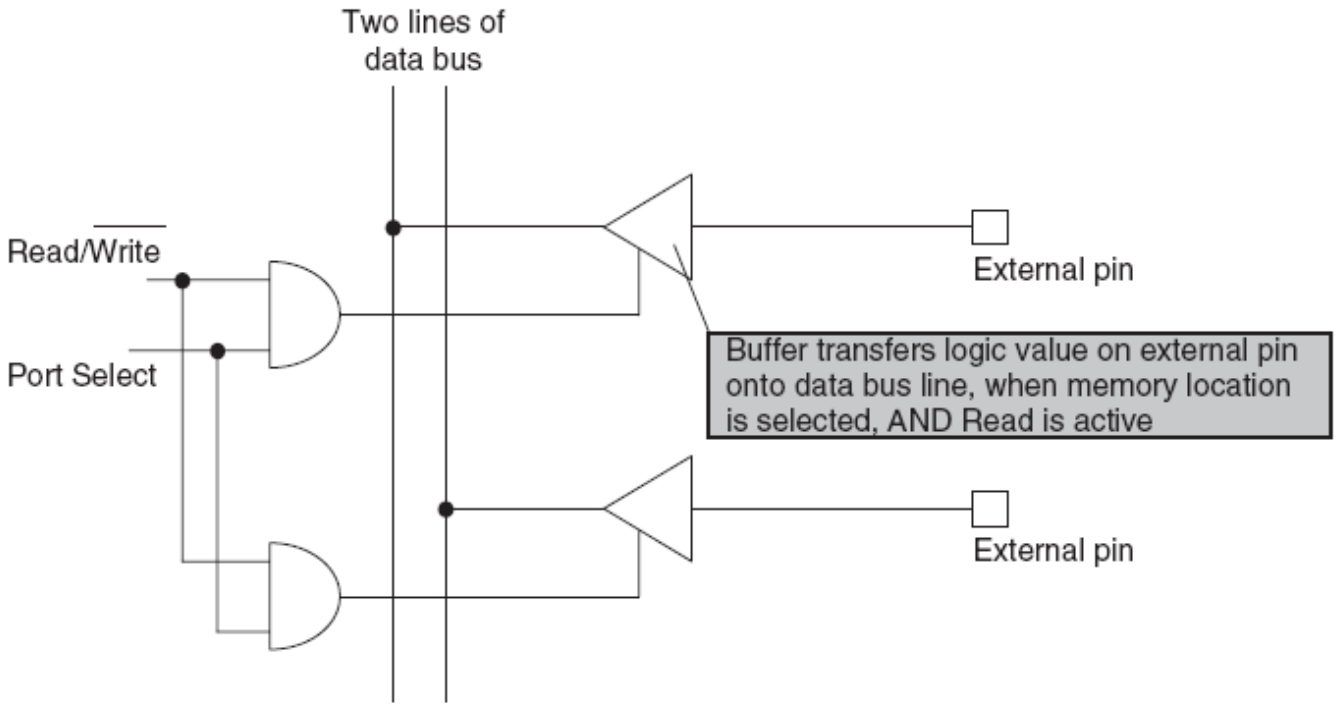
C	D	Q	
↑	0	0	(sample)
↑	1	1	
0	X	Q	
1	X	Q	(hold)
↓	X	Q	

(c) D flip flop truth table



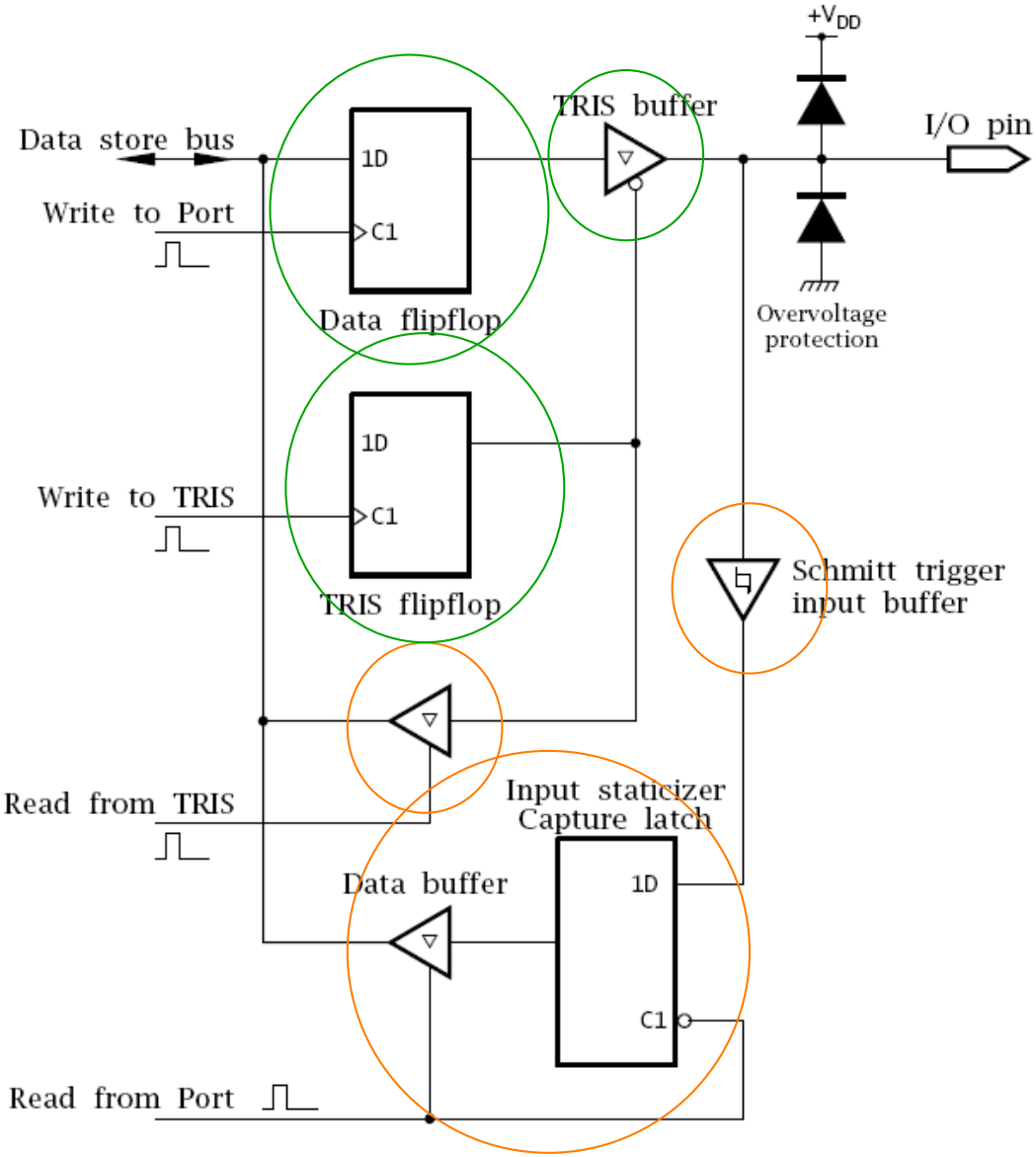
(d) D flip flop logic diagram

« Injection » d'une information en entrée sur une des pattes du µC

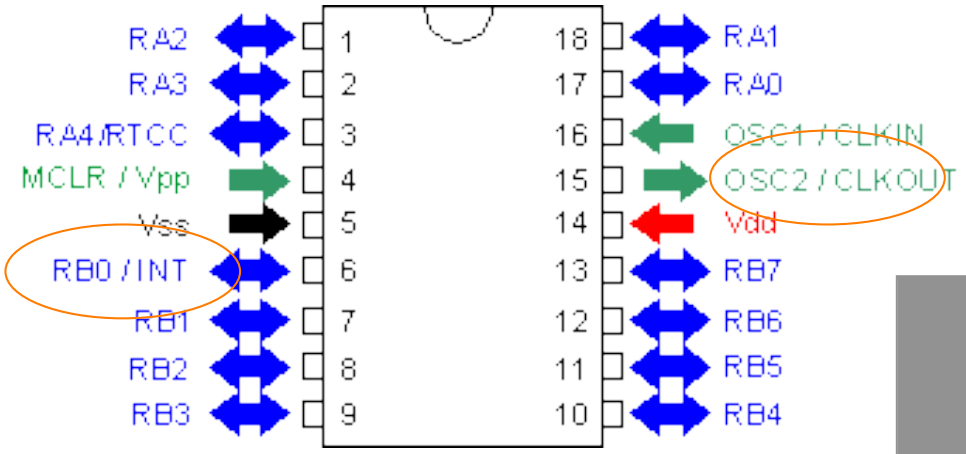


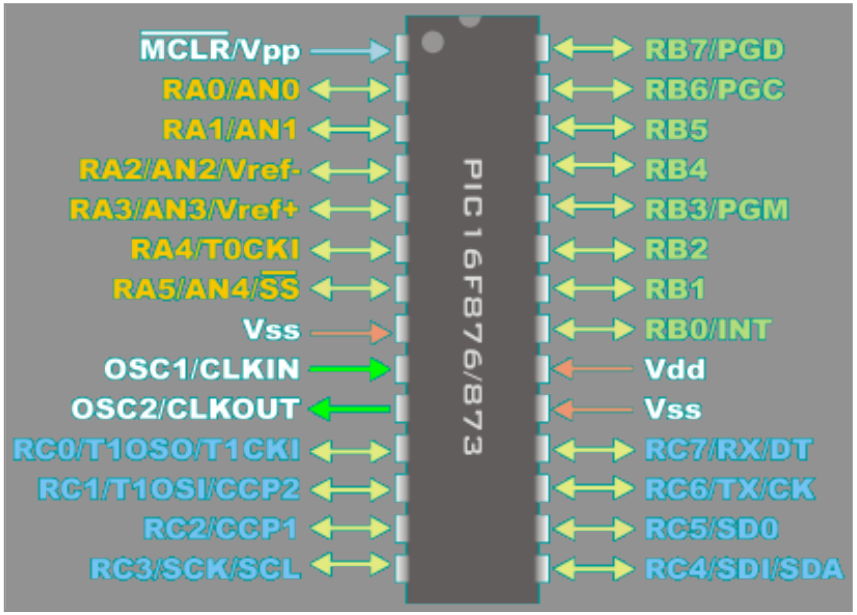
Two bits of a possible digital input port

- Identifier voie d'entrée (orange) / voie sortie (vert)

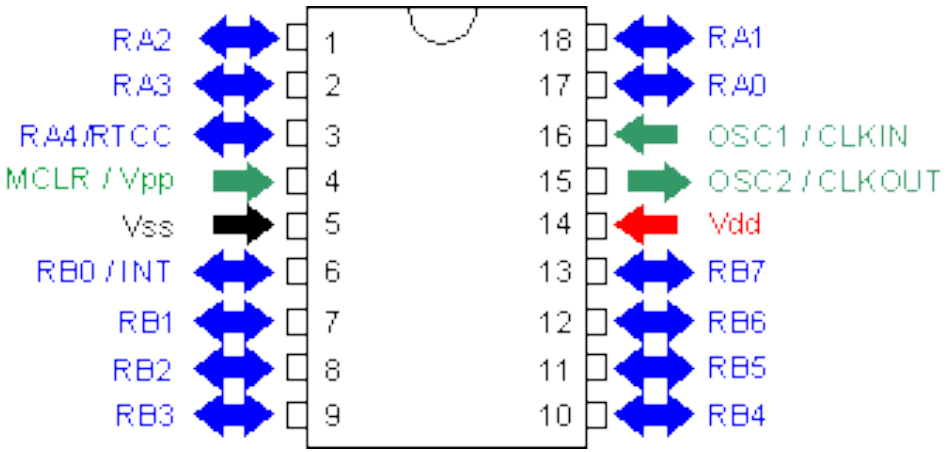


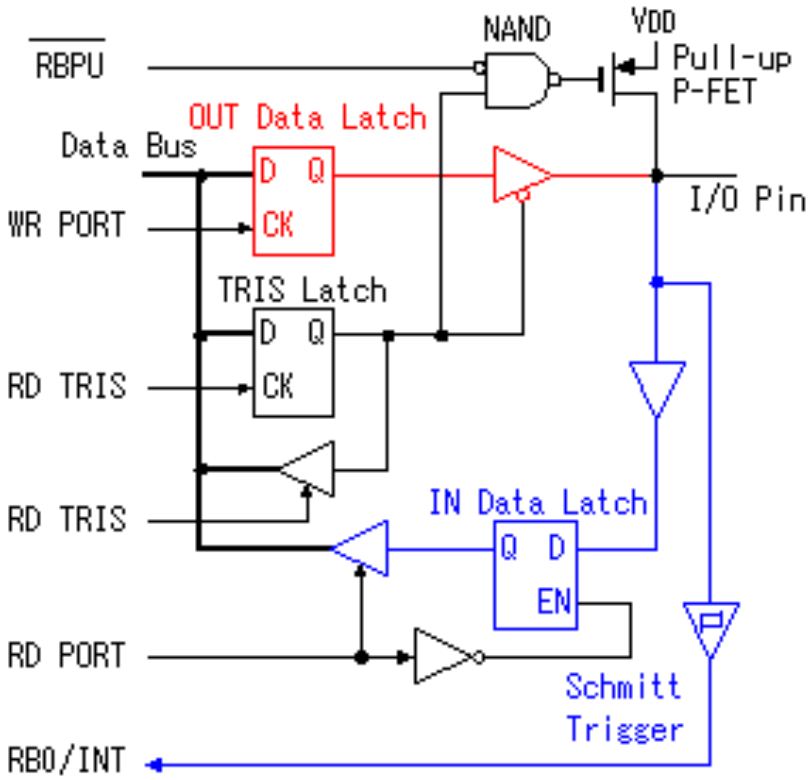
- Un µC
  - **Doit être le plus petit possible** ⇔ avoir peu de pattes ⇔ attribuer **plusieurs fonctions à une même patte**





- Exemple de paramétrage de portA
  - Il existe dans le µC des **circuits aiguillage** au niveau des **I/Os**





  - Les signaux de commande RDTRIS, TRIS LATCH….vont permettent de choisir si le PORTA est configuré en **entrée ou en sortie**

- Ces signaux sont stockés dans des registres spéciaux du µC : Special Function Register (voir partie suivante « cartographie mémoire »

- Pour pouvoir utiliser le portA pour piloter des LEDs; il faudra :
  1. **Paramétrer le portA en sortie** via son registre de contrôle TRISA : tous les digits de TRISA doivent être à « 0 ».
  2. Envoyer les bons signaux de commande au registres correspondant au portA

  ⇨ Voir exemple MPLAB, PicSimulator

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**
5. **Les I/Os multifonctions**
6. **ADC**

- **Présentation du document : « Le module ADC du PIC 16F88 »**
  - Rappel fonctionnement de la mémoire en BANK0, BANK1, …. (voir diapo suivante)
  - Exemple utilisation ADC – langage C – PIC (voir diapo suivante)

Le module ADC du PIC 16F88

- 1- Caractéristiques du convertisseur ADC
- 2- Mise en oeuvre
  - **Etape 1 : Configuration**
    - 1-1- Choix des canaux d'entrées
    - 1-2- Choix des tensions de référence
    - 1-3- Choix du format du résultat de la conversion
    - 1-4- Choix de la fréquence d'horloge du convertisseur
    - 1-5- Mise en service de l'interruption du convertisseur
  - Etape 2- Mise en service du convertisseur
  - Etape 3- Sélection du canal à échantillonner
  - Etape 4- Attente pendant la phase d'acquisition
  - Etape 5- Lancement de la phase de conversion
  - Etape 6- Attente de la fin de la phase de conversion

- Notion de BANK : but ne pas avoir une mémoire centrale linéaire en terme de liste d'adresse, ajout d'un MUX avec RP0 et RP1 pour diminuer la taille du bus adresse.

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
15h	CCPR1L
16h	CCPR1H
17h	CCP1CON
18h	RCSTA
19h	TXREG
1Ah	RCREG
1Bh	CCPR2L
1Ch	CCPR2H
1Dh	CCP2CON
1Eh	ADRESH
1Fh	ADCON0
20h	General Purpose Registers
7Fh	96 bytes

Bank 0

Addr.	Name
80h	INDF
81h	OPTION_REG
82h	PCL
83h	STATUS
84h	FSR
95h	WPUB
96h	IOCB
97h	VRCON
98h	TXSTA
99h	SPBRG
9Ah	SPBRGH
9Bh	PWM1CON
9Ch	ECCPAS
9Dh	PSTRCON
9Eh	ADRESL
9Fh	ADCON1
A0h	General Purpose Registers
FFh	80 bytes

Bank 1

Addr.	Name
100h	INDF
101h	TMR0
102h	PCL
103h	STATUS
104h	FSR
	General Purpose Registers
	96 bytes
17Fh	

Bank 2

Addr.	Name
180h	INDF
181h	OPTION_REG
182h	PCL
183h	STATUS
184h	FSR
	General Purpose Registers
	96 bytes
1EFh	

Bank 3