Thierry ROCACHER, INSA, DGEI

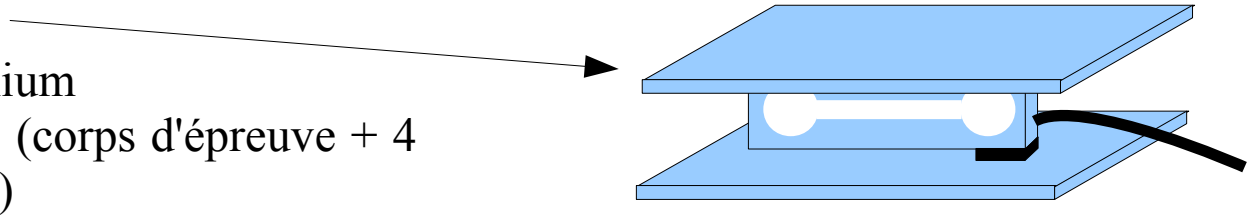
Introduction à l'électronique - Partie 2
4 IDS

- *Description fonctionnelle du système « balance électronique de précision »*
- *L'électronique numérique (Niveaux logiques, portes de base, bascule RS, bascule D)*
- *Eléments d'un micro-contrôleur (µC)*
  - *Le registre*
  - *L'unité arithmétique & logique (UAL)*
  - *La RAM*
- *Le format des nombres (Complément à 2, virgule fixe, virgule flottante)*
- *Le micro-contrôleur*
  - *Généralités*
  - *Exemple du µC : le C167 d'Infinéon*
  - *La CPU*
- *Analyse du programme de la balance électronique*
  - *Le langage d'assemblage*
  - *Le langage C*
- *Les périphériques de micro-contrôleur*
  - *Les ports d'entrées / sorties*
  - *L'ADC (généralités, ADC externe et interne)*
  - *Principes d'acquisition : la scrutation, l'interruption*

Description fonctionnelle du système

Constitution de la balance électronique :

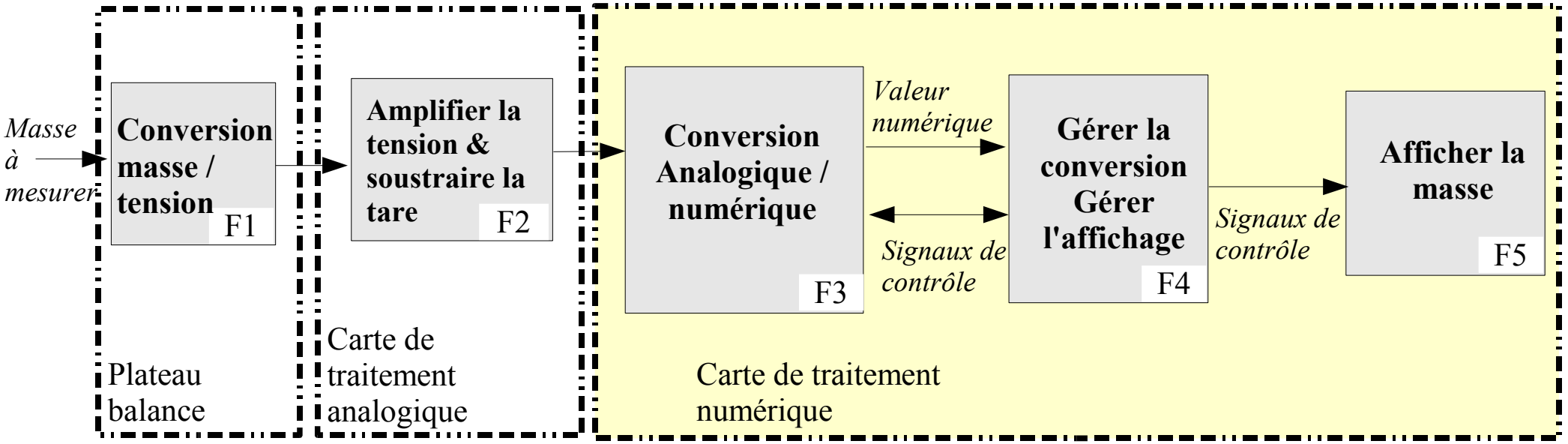
- Un plateau balance avec :
  - Un plateau d'aluminium
  - Un capteur de force (corps d'épreuve + 4 jauges de contrainte)
  - Un socle
- Une carte électronique de traitement analogique
- Une carte électronique de traitement numérique
- Une batterie



Spécifications :

- Résolution 1g
- Précision ≤ 1g
- Etendue de mesure 600g

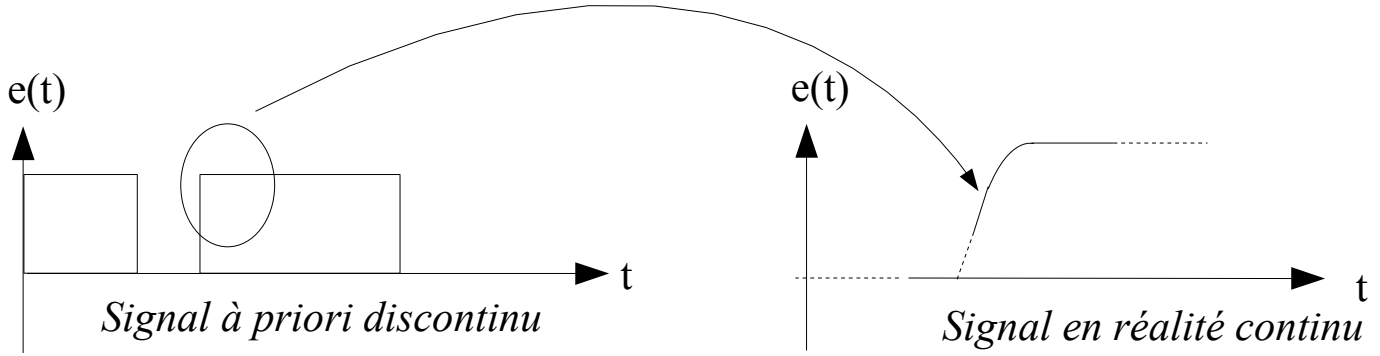
Description fonctionnelle du système



L'électronique numérique : les niveaux logiques

- **L'éléctronique analogique :**
  - les signaux électriques traités sont **continus**. Ce sont des tensions ou des courants. Le moindre µV ou nA a un impact sur les fonctions analogiques
  - Les fonctions associées sont des amplificateurs (homothétie de signaux), des filtres...

- **L'éléctronique numérique :**
  - Les signaux électriques sont, dans une première approche, discontinus, de forme carrée. A bien y regarder, les signaux sont continus, les angles sont en réalité arrondis.

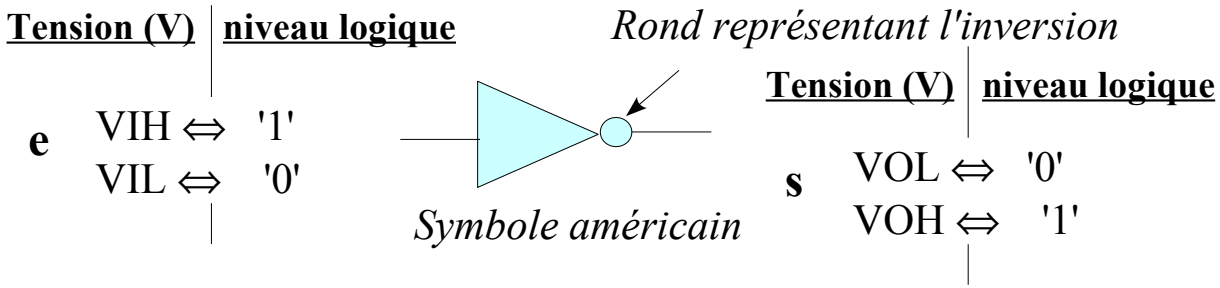


  - Un signal électrique numérique est **interprété** comme étant ***binaire***. S'il est au dessus d'une valeur **prédéterminée**, on dira que le signal est au ***niveau logique haut*** ou qu'il ***vaut '1'***. S'il est en dessous d'une valeur prédéterminée, le signal sera dit au ***niveau bas*** ou qu'***il vaut '0'***.

L'électronique numérique : les niveaux logiques

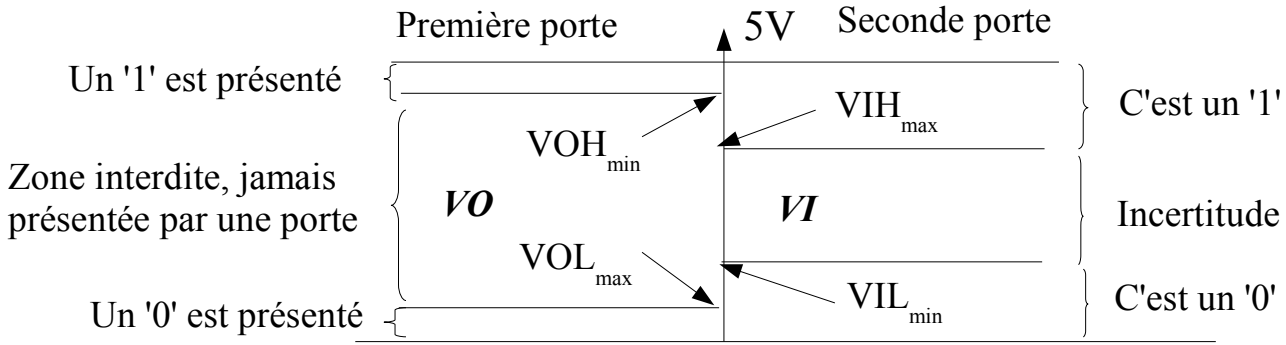
- Une famille logique est (entre autre) caractérisée par ces valeurs prédéterminées, des seuils, qui définissent les niveaux logiques. Parmi les grandes familles de circuits logiques, on trouve les TTL et les CMOS.

- Exemple: un inverseur de niveau (***porte logique***, qui donne en sortie le niveau inverse de celui d'entrée)



NB: Comme pour un AOP, afin d'alléger le graphique, on ne représente pas l'alimentation (0 & 5Vpar exemple), mais elle est bien présente !

- Quand e présente un niveau haut (tension VIH, niveau 1), alors la sortie indique une tension VOH, à savoir un niveau 1. (*VIH = V Input High, VOL = V Output Low*)



L'électronique numérique :un peu de logique combinatoire ...

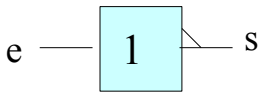
Fonction logique :

Comme en électronique analogique, l'électronique numérique est gérée par des fonctions. Celles-ci voient bien des tensions en entrée et sortie, mais ***elles travaillent*** en fait sur des ***niveaux logiques***. A toute fonction logique combinatoire, on peut associer une équation et une table de vérité.

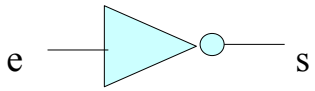
Les portes logiques fondamentales:

Porte *NON, NOT*

Symbole EU



Symbole US



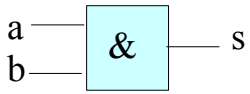
Équation : $s = \bar{e}$

Table de vérité

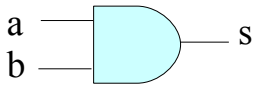
e	s
0	1
1	0

Porte *ET, AND*

Symbole EU



Symbole US



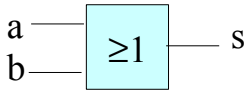
Équation : $s = a \,.\, b$

Table de vérité

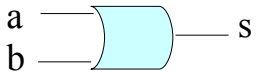
a	b	s
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Porte *OU, OR*

Symbole EU



Symbole US



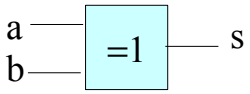
Équation : $s = a + b$

Table de vérité

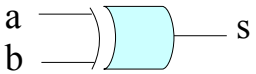
a	b	s
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Porte *OU exclusif, XOR*

Symbole EU



Symbole US



Équation : $s = a \oplus b$

Table de vérité

a	b	s
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

L'électronique numérique : un peu de logique séquentielle ...

Logique séquentielle :

Contrairement à la logique séquentielle, la sortie (les sorties) ne dépend pas uniquement des entrées. Elle dépend aussi de la sortie elle-même. Par exemple un compteur est une logique séquentielle.

Synchrone ou asynchrone ???

On dit d'un système séquentiel qu'il est synchrone si les sorties évoluent toutes ensembles, au front montant (ou descendant) d'un signal carré qu'on appelle horloge (Ck).
Un système séquentiel est asynchrone dans les autres cas...

La fonction séquentielle asynchrone de référence : la bascule RS (Set Reset)

Symbole

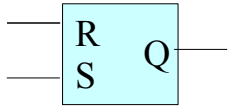


Table de vérité

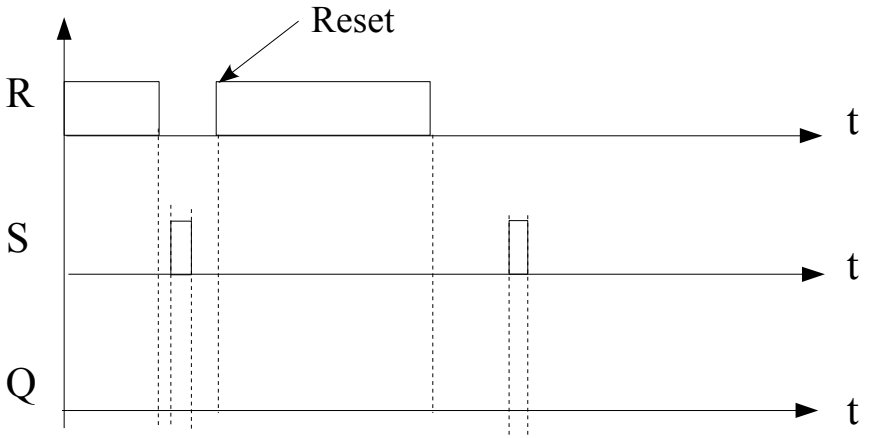
S	R	Q
0	0	**mem**
1	0	**1**
0	1	**0**
1	1	**?**

Memoire : Q reste inchangé

Le niveau dépend du type de bascle RS.
1 si Set prioritaire,
0 si Reset prioritaire

Exemple d'utilisation : point mémoire asynchrone...

Exemple de chronogramme



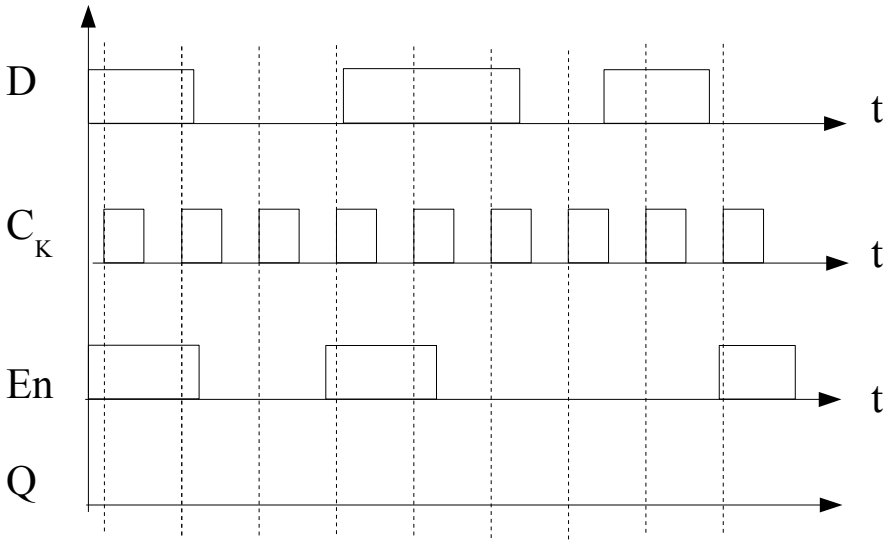
L'électronique numérique : un peu de logique séquentielle ...

La bascule D :

Mémorise un niveau logique (entrée D) sous 2 conditions :

- L'entrée de validation (En) est au niveau '1'
- Un front montant vient d'apparaître sur l'entrée Ck

Exemple de chronogramme



La fonction séquentielle synchrone de référence : la bascule D

Symbole

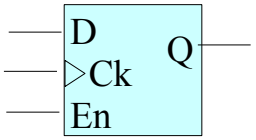


Table de vérité

Memoire : Q reste inchangé

En	D	Ck	Q
0	X	X	**mem**
1	X	1	**mem**
1	X	↓	**mem**
1	0	↑	**0**
1	1	↑	1

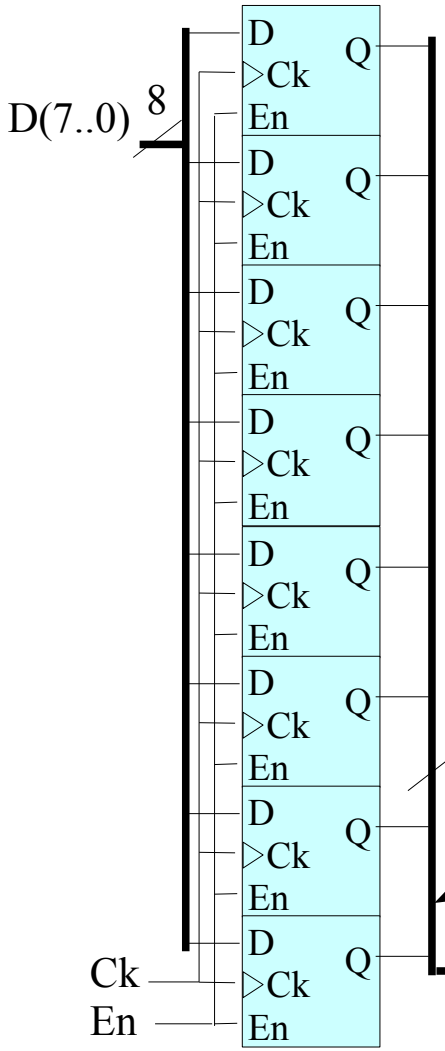
Front descendant de Ck

Front montant de Ck

X : Le niveau 1 ou 0, c'est sans importance

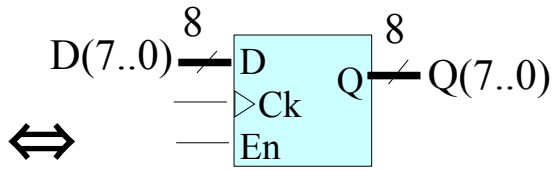
Exemple d'utilisation : point mémoire, compteur synchrone, registre...

Eléments d'un µC, le registre : la brique d'un micro-contrôleur



Registre

C'est un groupe de 8, 16, 32 voire même 64 bascules D. Un registre possède donc une ***largeur*** de 8 à 64 bits. Il peut stocker (mémoriser) un ***nombre*** (variable d'un programme par exemple). Plutôt que de représenter n bascules D, on représente une seule, et les entrées D (et sorties Q) sont regroupées en ***bus***.



Registre 8 bits

- Chaque fil possède un ***rang***, de 0 à 7 (pour N = 8 bits). On dit que le fil de rang 0 est le ***bit*** de **poids faible** et celui de rang 7 est le **bit** de **poids fort**.
- Exemple de nombre stocké dans un registre :

  Q = 1011 0010 écriture, base ***binaire***

  1 ***octet*** (8 bits) =
  2 ***quartets*** (4bits)

- Base ***hexadécimale***

un quartet (4 éléments pouvant prendre 2 valeurs -0 ou 1-) peut prendre 2^4 combinaisons (valeurs) possibles. Avec les chiffres on peut associer 10 valeurs seulement. En ajoutant 6 lettres, on obtient le jeu de 16 **caractères hexadécimaux** 0 à F, de la base hexadécimale.

Eléments d'un µC, le registre : la brique d'un micro-contrôleur

- **Représentation d'un nombre transitant dans des registres : Relations entre bases *Binaires, Décimales & Hexadécimale***

Base Binaire	*Base Décimale*	*Base Hexadécimale*
0000	**0**	**0**
0001	**1**	**1**
0010	**2**	**2**
0011	**3**	**3**
0100	**4**	**4**
0101	**5**	**5**
0110	**6**	**6**
0111	**7**	**7**
1000	**8**	**8**
1001	**9**	**9**
1010	**10**	**A**
1011	**11**	**B**
1100	**12**	**C**
1101	**13**	**D**
1110	**14**	**E**
1111	**15**	**F**

- En binaire, chaque bit d'un nombre a un ***poids*** fonction de son rang. Le rang 0 a le poids 2^0, le rang 1 a le poids 2^1...le rang n a le poids 2^n. La somme de ces poids (chacun muliplié par 0 ou1) donne la valeur décimale :
  Exemple précédent : G = **1011 0010***b* (b pour *b*inaire)
  $G=1.2^7+0.2^6+1.2^5+1.2^4+0.2^3+0.2^2+1.2^1+0.2^0=$ **178***d*

- Ecriture en hexadécimal : Chaque quartet binaire est remplacé par son caractère hexadécimal :
  G = 1011 0010*b* = **B2***h* (*h* pour hexadécimal)

- Passage direct hexadécimal vers décimal: chaque caractère hexadécimal a un poids (comme en binaire) mais dans ce cas les poids sont 16^0, 16^1, 16^2...
  G = **B2***h* = $11.16^1+2.16^0=$ **178***d*

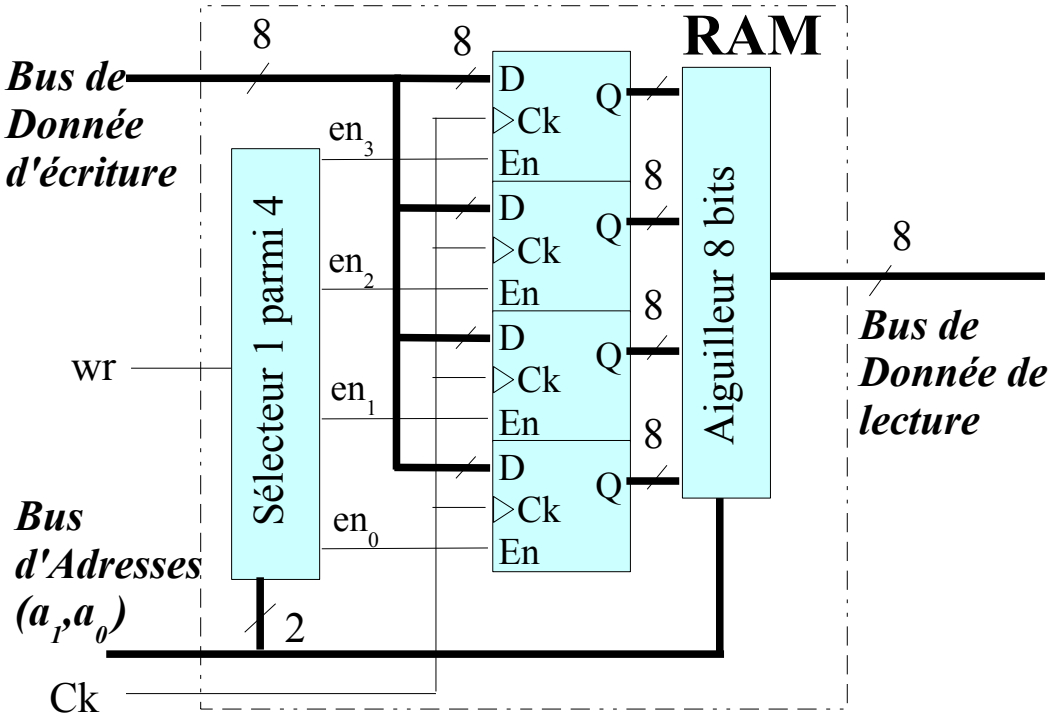
- La taille d'un registre est adaptée au nombre à traiter. Par exemple, des registres 16 bits permettront de traiter des nombres de 0 à 2^{16}-1, soit 0 à 65535

Eléments d'un µC, la RAM : un groupe de registres

- **RAM** : (Random acces Memory)

C'est un groupe de registres dans lesquels on peut ranger (*écrire*) ou venir chercher (*lire*) des nombres (*variables* d'un programme). Elle perd son contenu à l'extinction de son alimentation. Une RAM peut être vue comme un grande commode avec beaucoup de tiroirs (*registre*) dans lesquels on range des objets (*variable*).

Structure électronique : exemple RAM 4 Octets



- Le bloc « Sélecteur 1 parmi 4 » est une logique combinatoire qui obéit à la table de vérité :

wr	a_1	a_0	en3	en2	en1	en0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	x	x	0	0	0	0

- Le bus d'adresse (a_1,a_0) précise quel registre est autorisé en écriture. Au front de Ck, la donnée présente sur le ***bus de donnée d'écriture*** est mémorisée dans le registre correspondant (si wr = '1').
- L'aiguilleur 8 bits place l'une des sorties des 4 registres sur le ***bus de donnée de lecture,*** en fonction de la combinaison du bus d'adresse.

Vue fonctionnelle de la RAM

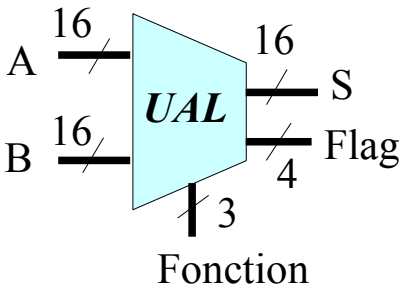
@	donnée
0	0x2F
1	0x1A
2	0x84
3	0x01

Eléments d'un µC, l'Unité Arithmétique & logique

- **UAL** (Unité Arithmétique & logique)

Il s'agit d'une logique combinatoire, plus ou moins complexe capable d'effectuer des opérations arithmétiques (addition, soustraction, multiplication...) et logique (ET, OU...).

- **Exemple d'une UAL simple (16 bits)**



- Le bus *Fonction* permet de choisir l'opération à effectuer (8 dans ce cas). Par exemple '000*b*' correspond à l'addition, '001*b*' au ET logique...
- A et B sont les entrées de l'UAL. C'est par là que sont envoyés les ***opérandes*** (les valeurs à traiter). S est la sortie, le résultat de l'opération.
- Le champ *Flag* regroupe un certains nombre d'indicateurs (les *drapeaux*).

Citons parmi les plus utilisés :

  - **N** : c'est le drapeau (1 bit du bus Flag) qui indique que l'opération donne un résultat **négatif**
  - Z : c'est le drapeau qui indique que l'opération donne un résultat **nul**
  - C : indique qu'une opération d'addition par exemple provoque une retenue (***carry***)
  - Ov : C'est le bit d'***Overflow***. Il indique un dépassement de capacité. Le résultat ne tient pas dans les 16 bits requis (multiplication par exemple)

Les formats de nombre : le complément à 2

- L'UAL doit permettre de traiter des nombres relatifs. Il faut donc faire évoluer le codage binaire déjà vu vers un autre, que l'on qualifiera de codage entier **signé**.

Code complément à 10 en base décimale

Afin d'expliquer cette notion, prenons l'exemple d'un compteur à 4 chiffres. Il peut inscrire des nombres de 0 à 9 999.
Le nombre -1 est celui, qui, ajouté à +1 donne 0. Si on considère le nombre 9 999, on y ajoute +1 et on obtient 0 ! (en réalité 10 000 mais le 5ème chiffre n'est pas représenté) . Ainsi, le nombre 9 999 est équivalent à -1.
La **représentation d'un nombre négatif *a*** s'obtient en prenant le **complément à 9 de |*a*|** et en **ajoutant 1**.

Exemple, représentons -2 en complément à 10 :
|-2| = 2 ; complément à 9 (dans le format du compteur) : 0 002 ⇒ 9 997 ; Ajout +1: 9 997⇒ **9 998** ($=10^4$-2)

Code complément à 2 en base binaire

	9 998d	⇔ -2*d*
+	0 002*d*	
	0 000*d*	

La méthode reste la même : soit une taille de N bits et un nombre *a* négatif
- on prend le complément à 1 de |*a*|
- on ajoute 1

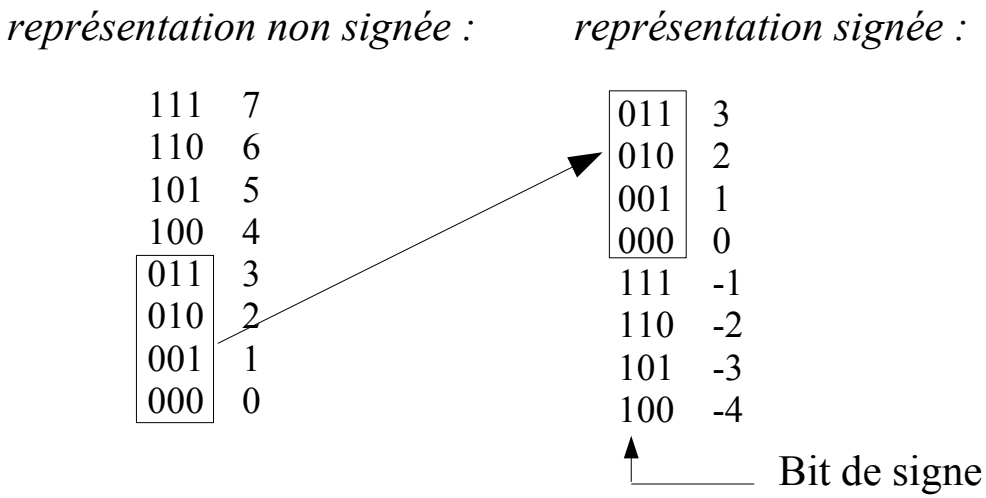
Exemple, soit à représenter -14 sur 8 bits, |-14| = 14 s'écrit 0000 1110, complément à 1 : 1111 0001,
on ajoute +1, ⇒ **1111 0010*b*** = F2*h* = (242*d*)
Une seconde méthode consiste à soustraire directement la valeur absolue de 2^N : $\mathbf{2^N}$ **- |a|** (2^8-14 = 256-14 = 242*d*)

Les formats de nombre : le complément à 2

- **Intérêt la notation complément à 2 :** Une soustraction de deux nombres peut se ramener à une **somme**.

- En langage évolué (C par exemple), on peut préciser le format des variables. On peut aussi spécifier s'il s'agit d'un format ***signé*** ou ***non signé***. Par exemple le nombre hexadécimal **ED84*h*** sera vu comme la valeur **60 804*d*** en contexte ***non signé***, mais bien ***-4 732d*** en contexte **signé**.

Exemple sur 3 bits :



Réciproquement, à quoi correspond par exemple FFA1h sur 16 bits ?

- C'est un nombre négatif, puisque son bit de poids fort est à 1
- En décimal non signé, FFA1*h* = 65441*d*
- On sait que 2^N-65441 = |a| = 95
- La valeur est donc ***-95 en 16 bits signé***
- On peut directement soustraire au nombre (interprété non signé) la valeur 2^N

Les formats de nombre : la virgule flottante

- **virgule fixe / virgule flottante**

Tout ce qui a été vu précédemment ne concerne que les nombres entiers. Pour coder des nombres réels deux options :

- ***Le format à virgule fixe*** : On associe M bits pour définir la partie entière et P bits pour la partie fractionnaire, avec bien sûr la contrainte M+P = N.

  Exemple : 0110 10.11 0000 0000*b* = 6B00*h* = +26,75*d*

  Inconvénients *:* - L'amplitude de codage est limité (augmenter M)
  - La granularité est forte (augmenter N)

  L'usage : Dans les applications professionnelles (utilisation de DSP), on adopte le standard 1.15 ou 1.31 qui signifie que les nombres sont dans l'intervalle [-1 ; +1] avec un grain de 2^{-15} (2^{-31})

- ***Le format virgule flottante :*** on définit un nombre de la manière suivante Nbre = ***Mantisse***.$2^{\textbf{Exposant}}$

Le standard IEEE-754 précise pour le format ***float 32 bits***:

- L'*exposant* est « décalé de 127 » cela signifie que la valeur effective est celle lue - 127.
- La *mantisse* est signée, 24 bits de taille.
- Intérêt du codage : l'étendue des nombres est très importante, la granularité est très fine

Le micro-contrôleur : généralités

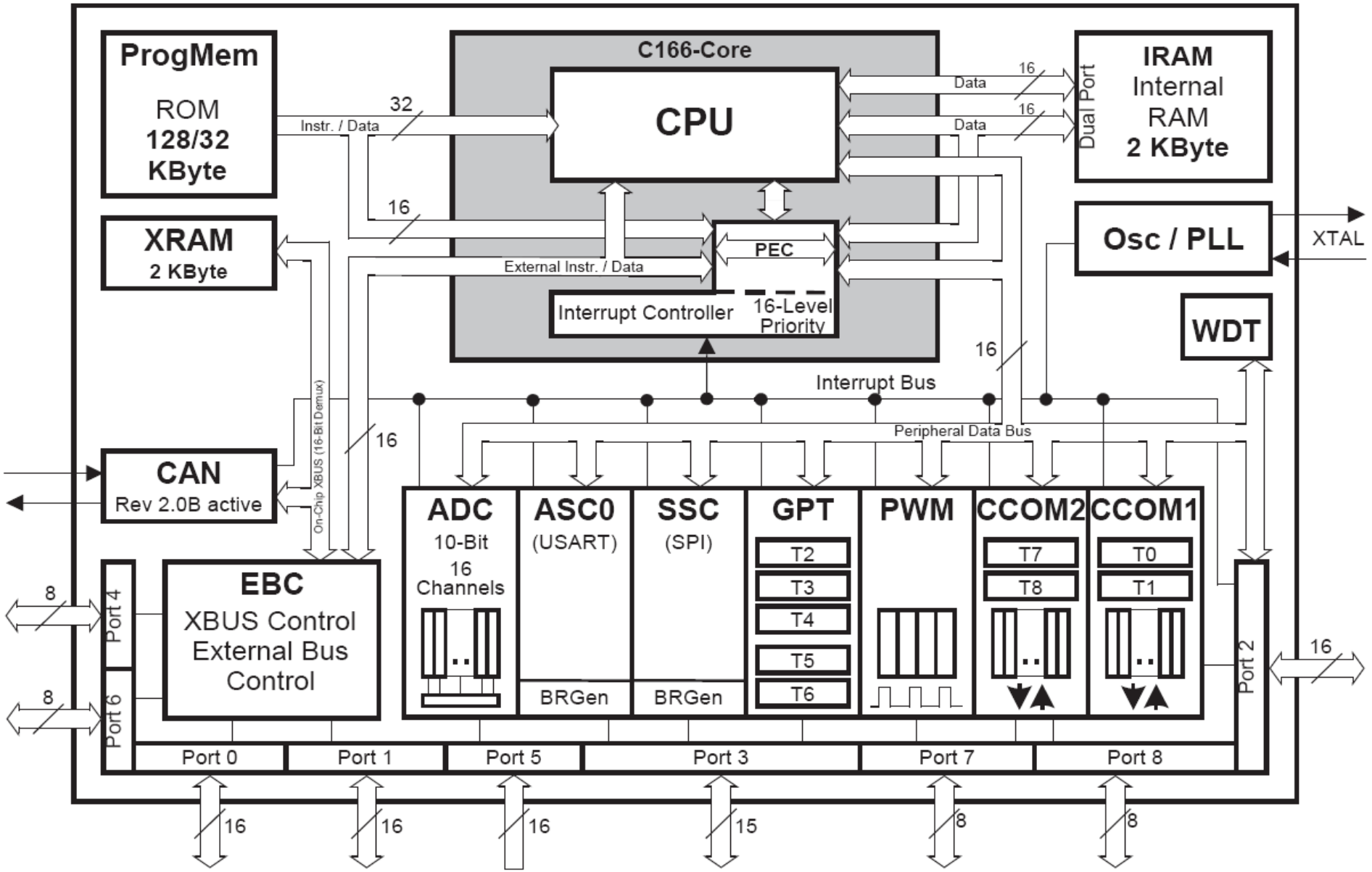
Rôle

C'est un circuit électronique entièrement basé sur des fonctions de l'électronique numérique, qui a pour rôle de contrôler un processus physique (injection sur un moteur thermique, télévision, réfrigérateur, ...y en a partout !...). Il ***éxécute*** un ***programme*** qui est composé d'instructions élémentaires : le ***jeu d'instructions***. Il est doté d'interfaces capables d'échanger avec l'extérieur : les ***périphériques***.

Constitution

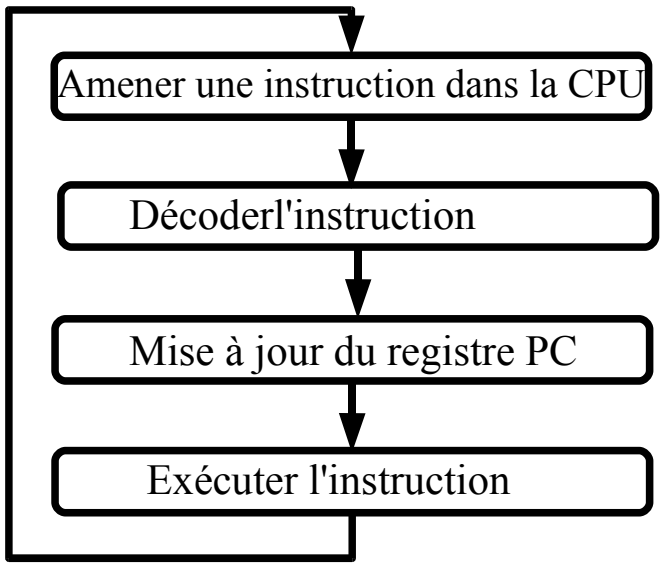
- Le dispositif ***d'horloge*** : C'est le coeur du µC. Il fixe la cadence de fonctionnement. Basé sur un quartz.
- La ***RAM & la ROM*** : La ROM permet de ranger le programme. La RAM permet de ranger les variables amenées à changer en cours d'exécution de programme. Possible grâce aux ***bus d'adresse*** et de ***données***.
- La ***CPU*** (Central Processing Unit) : C'est le cerveau du µC. Elle contient l'UAL. Elle ***contrôle*** l'exécution du programme. Elle travaille au rythme de l'horloge. Son jeu d'instructions lui permet de :
  - ***Opérer des transferts*** d'un registre à un autre, de la ROM à la RAM...
  - ***Effectuer des opérations*** grâce à l'UAL
  - ***Faire des tests*** sur des flags ou tout autre bit ou registre pour créer des structures de programme du type *si ... alors*, ou bien *Faire... tant que*, etc...
- Les ***périphériques*** : Ce sont les bras et les jambes du µC. Ils permettent la communication avec l'extérieur sous la forme, par exemple, d'un bus de 16 fils (port parallèle) ou d'un seul fil (port série)...

Le micro-contrôleur : Le C167 de Infinéon

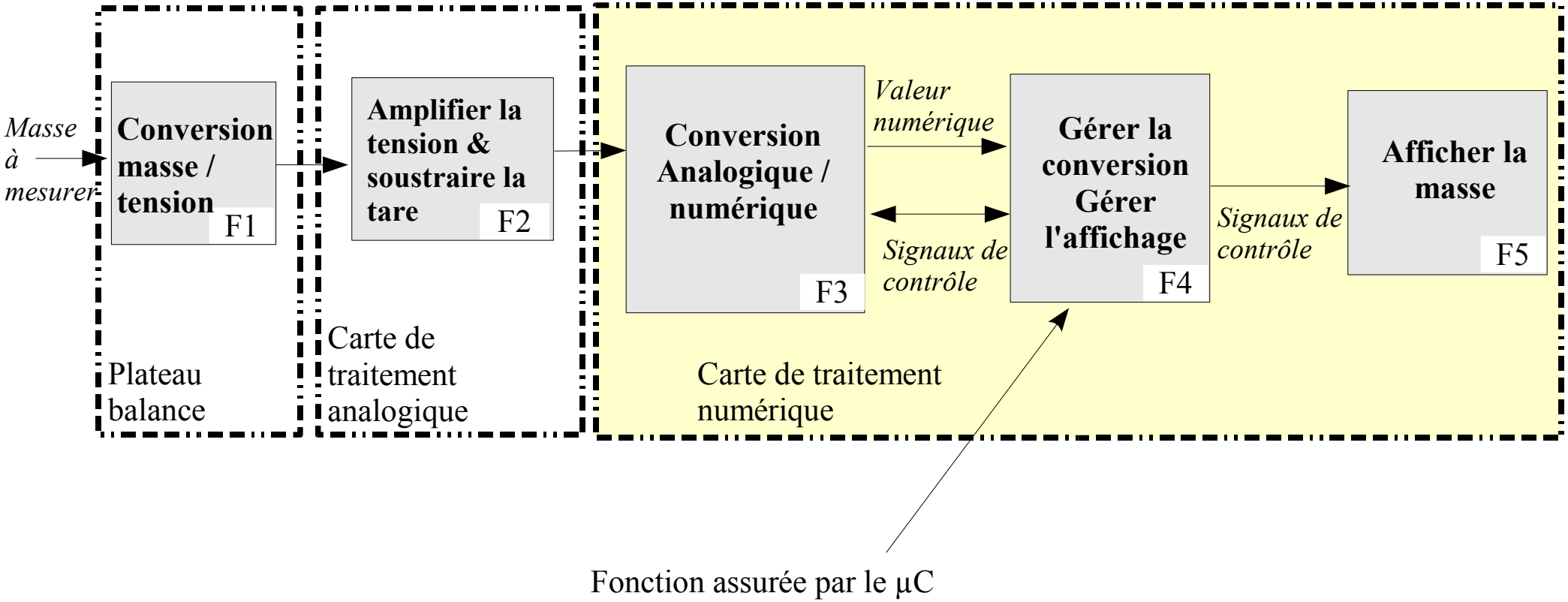


Le micro-contrôleur : la CPU

- **Séquencement de base d'une CPU**:
  - Faire toujours :
    - **Amener une instruction** depuis la ROM jusqu'à la CPU. Pour cela le registre spécial ***PC*** (***P*ointeur *P*rogramme**), interne à la CPU mémorise l'adresse ROM de l'instruction à traiter.
    - **Décoder l'instruction** qui vient d'être mémorisée dans la CPU (le code hexadécimal correspond forcément à l'une des instructions du jeu d'instructions de la CPU).
    - **Mise à jour du registre PC** : le PC pointe maintenant sur la prochaine instruction à traiter
    - **Exécuter l'instruction**
  - Fin Faire



Analyse du programme de la balance



Analyse du programme de la balance

Algorithme global :

Faire toujours

Acquérir la grandeur numérique, image de la masse (ADC)
Créer les 3 chiffres : centaines, dizaines, unités à partir de la masse
Afficher sur le LCD les 3 chiffres
Temporiser 1s

Fin Faire

Détail de la partie « Créer les 3 chiffres » : centaines, dizaines, unités

Soient les variables suivantes :
Masse, Cent, Diz, Unit

Algorithme *:*
cent ⇐ Quotient (*Masse* / 100)
Masse ⇐ Reste (*Masse* / 100)
diz ⇐ Quotient (*Masse* / 10)
unit ⇐ Reste (*Masse* / 10)

Avant exécution:	***Après exécution :***
Masse : 548	Masse : 48
Cent : 0	Cent : 5
Diz : 0	Diz : 4
Unit :0	Unit :8

Instructions du µC
- Movw(Move Word) : déplace un mot de 16 bits d'un registre vers un autre
- Divu (Divide Unsigned) : opère la division d'un mot de 16 bits par un mot de 16 bits

Opérandes de l'instruction *variables sur lesquelles porte l'instruction*

Programme ***assembleur***

```
; Cent <= Masse /100
Movw    MDL,Masse
Movw    R0,#100
Divu    R0   ; MDL <= Quotient (MDL / R0)
             ; et MDH <= Reste (MDL / R0)
Movw    R1,MDL
Movw    Cent, R1
; Masse <= Reste (Masse /100)
Movw    R1,MDH
Movw    Masse, R1

; Cent <= Masse /10
Movw    MDL,Masse
Movw    R0,#10
Divu    R0   ; MDL <= Quotient (MDL / R0)
             ; et MDH <= Reste (MDL / R0)
Movw    R1,MDL
Movw    Diz, R1

; Unit <= Reste (Masse /10)
Movw    R1,MDH
Movw    Unit, R1
```

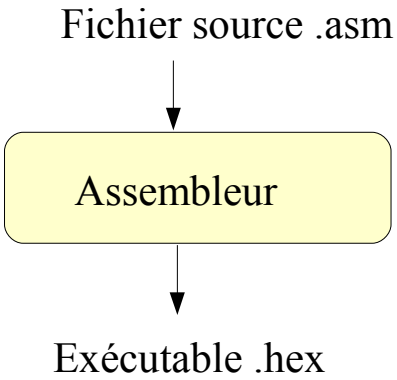
L'assembleur et le langage d'assemblage

Le langage d'assemblage :

Une **instruction est un code hexadécimal** rangé en ROM. Dès que ce code pénètre dans la CPU, il est reconnu comme étant une instruction particulière (par exemple placer une valeur dans un registre). Or un code d'instruction du type *E6F06400h* totalement incompréhensible signifie R0 ⇐ 100*d*...
On adopte donc un verbiage explicite, dit ***langage d'assemblage***, qui permet de manipuler les instructions sans se soucier des codes d'instruction.
Par exemple , movw R0,#100 se lit (avec un peu d'habitude...) « *place dans le registre R0, la valeur 100* »

L'assembleur :

C'est le logiciel qui prend en entrée le programme écrit en langage d'assemblage, puis qui le ***traduit*** en langage machine (code hexadécimaux obscurs...pour nous...).



Le téléchargement :

L'écriture de programme d'assemblage, l'obtention de ***fichier exécutable***, se fait grâce à un environnement de développement, IDE (KEIL par exemple) . Le fichier exécutable (qui contient les codes hexadécimaux doit être ensuite ***loger en ROM***.
C'est l'opération de ***téléchargement*** qui se fait souvent par le ***port série*** ou plus récemment par le ***port USB***. Le câble série est relié à la carte électronique qui porte le µC.

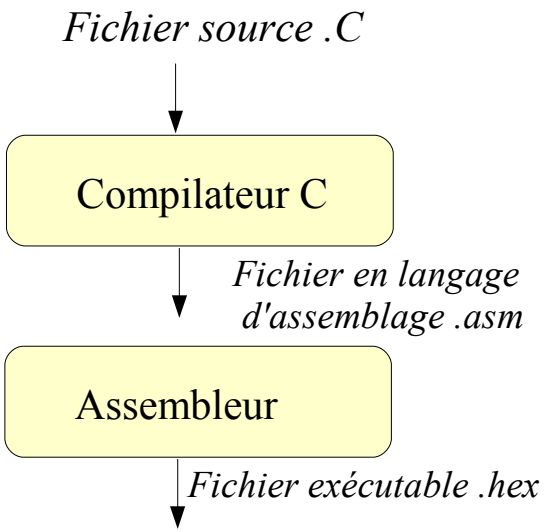
Langage de haut niveau : le C

- **Nécessité d'un langage de haut niveau :**

Chaque µC possède son propre jeu d'instructions et donc son propre langage d'assemblage ⇒ difficulté de prise en main d'un nouveau µC. Le langage C a été choisi pour être un standard en matière de programmation du µC.

- **Compilation :**

Afin d'unifier les programmations de µC, les fournisseurs proposent systématiquement avec le µC toute une série d'outils dont un **compilateur C**.
Il permet une écriture en langage C (avec quelques légères retouches) et une ***traduction en langage d'assemblage*** puis en langage machine.



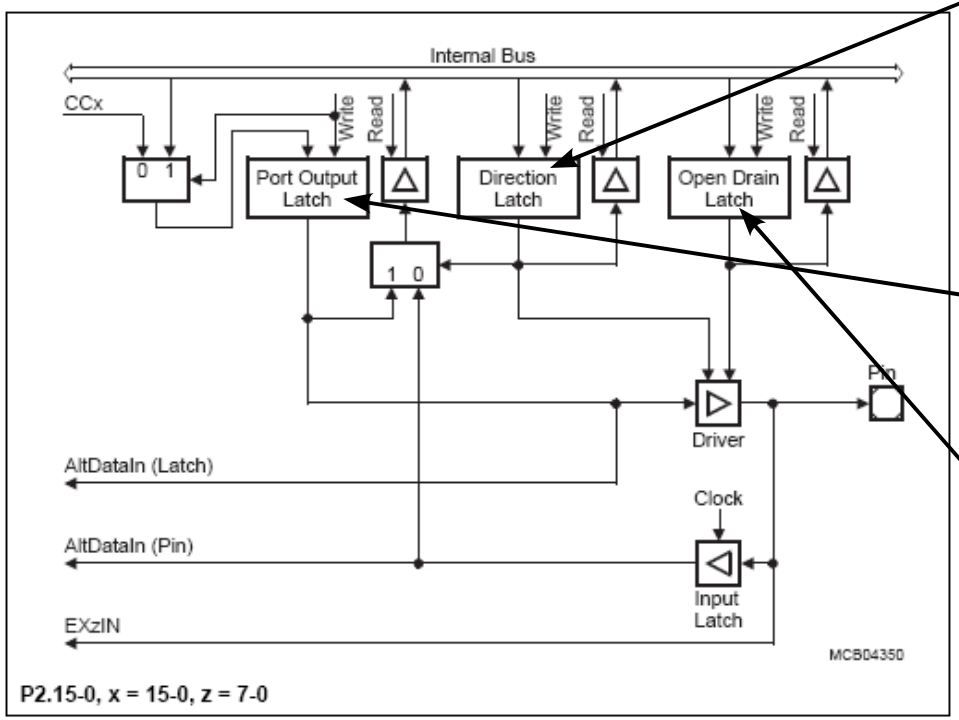

```
void main(void)
{
int Masse,Cent,Diz,Unit;

Cent = Masse/100;
Masse = Masse-100*Cent;
Diz=Masse/10;
Unit=Masse-10*Diz;

while(1)
{}
```

Périphérique de µC : Les ports d'E/S

- **Port d'entrées / sorties**

C'est un ***dispositif logique***, ***interne*** au µC, qui propose plusieurs ***broches*** qui peuvent servir ***d'entrées*** ou de ***sorties*** indépendamment. Un µC possède souvent plusieurs ports. La taille d'un port varie usuellement de 8 à 16 bits.

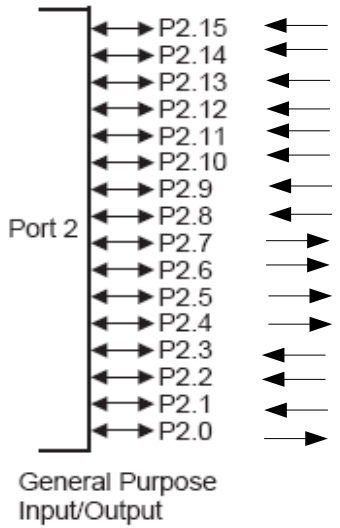
- **Exemple de structure de port :**



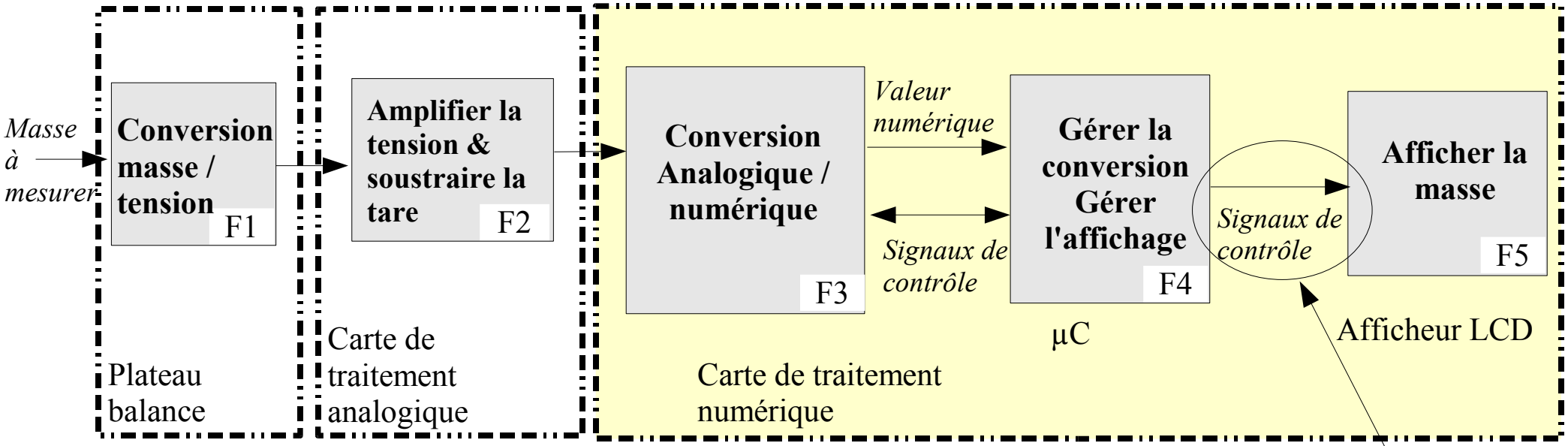
Direction Latch : registre 16 bits qui fixe la direction de chaque broche du port. Si le bit *i* de ce registre est à '1', alors la *broche i* est en sortie. Sinon c'est une entrée.
Exemple : 0x00F1

Port Output Latch : registre 16 bits qui fixe le niveau logique, sur chaque broche en sortie.

Open Drain Latch : registre 16 bits qui fixe les technologies de chaque broche configurée en sortie

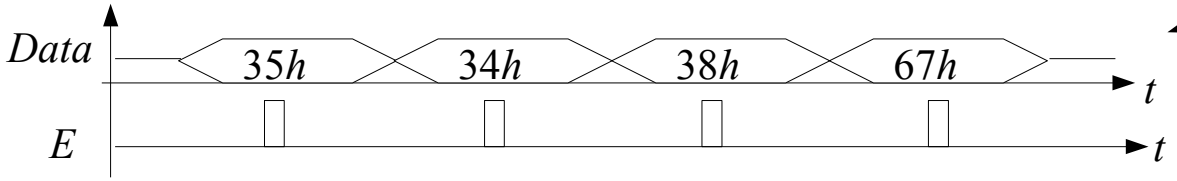


Périphérique de µC : Exemple d'utilisation des ports d'E/S



- **Fonctionnement de l'afficheur à cristaux liquides**
  - 8 bits (*Data*) permettent de transmettre (en parallèle) l'octet qui représente le code ASCII du caractère à transmettre.
  - 1 bit de contrôle, *E*, permet de valider (afficher sur écran) le caractère correspondant au code ASCII présent sur le bus *Data*.

Port P2



Cette séquence affiche successivement les caractères 5,4,8 et g : ***458g***

Périphérique de µC : Exemple d'utilisation des ports d'E/S

Algorithme global :

Faire toujours
    Acquérir la grandeur numérique, image de la masse (ADC)
    Créer les 3 chiffres : centaines, dizaines, unités à partir de la masse
    Afficher sur le LCD les 3 chiffres
    Temporiser 1s
Fin Faire

Algorithme d'affichage LCD:

- Configuration du Port P2 :
    P2(7..0) sortie (Data) P2.8 sortie (E)
- Construire le code ASCII des 3 chiffres à afficher
- Envoyer les 3 code ASCII et celui de la lettre g

Remarque :

La communication entre le µC et l'afficheur LCD est dite ***parallèle,*** puisque les 8 bits de donnée sont transmis d'un bloc, tous en parallèle.

Programme en C

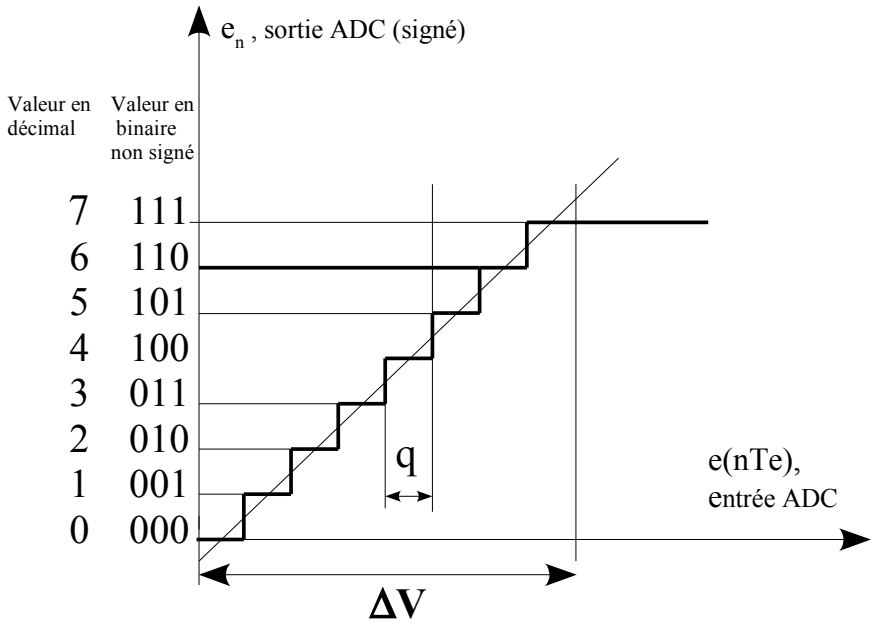
```
// Configuration port2 : P2(9..0) en sortie
DP2=DP2|0x01FF;

// Codage Centaine
CodeASCII=0x30 + Cent;
// Sortie du code ASCII sur P2(7..0)
P2=P2|CodeASCII;
CodeASCII=0xFF00|CodeASCII;
P2=P2& CodeASCII;
// pulse sur E
P2=P2|0X0100;
P2=P2&0XFEFF;

// Codage Dizaine
CodeASCII=0x30 + Diz;
// Sortie du code ASCII sur P2(7..0)
P2=P2|CodeASCII;
CodeASCII=0xFF00|CodeASCII;
P2=P2& CodeASCII;
// pulse sur E
P2=P2|0X0100;
P2=P2&0XFEFF;

// Codage Dizaine
CodeASCII=0x30 + Unit;
// Sortie du code ASCII sur P2(7..0)
P2=P2|CodeASCII;
CodeASCII=0xFF00|CodeASCII;
P2=P2& CodeASCII;
// pulse sur E
P2=P2|0X0100;
P2=P2&0XFEFF;

// Codage de la lettre g
CodeASCII='g';
// Sortie du code ASCII sur P2(7..0)
P2=P2|CodeASCII;
CodeASCII=0xFF00|CodeASCII;
P2=P2& CodeASCII;
// pulse sur E
P2=P2|0X0100;
P2=P2&0XFEFF;
```

Périphérique de µC : l'ADC (Analog to Digital Converter)

- **Conversion Analogique / Numérique :**

C'est un dispositif mixte, qui permet de **transformer** une **tension analogique** en une **valeur numérique** sur **N bits**. Plus N est important, plus la représentation est fine (***granularité fine***). La tension est discrétisée en un nombre d'intervalles $\mathbf{2^N}$. Chaque intervalle de tension est identique : ***q***, le ***quantum***.

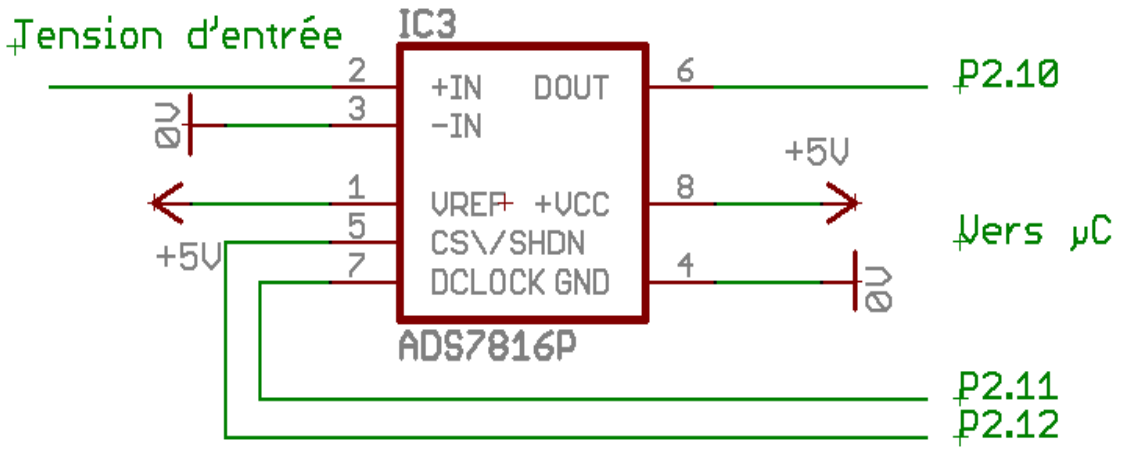
- **Caractéristiques principales :**
  - ***La résolution*** : C'est le nombre de bits N, ou encore le nombre d'intervalles, ou encore le quantum.
  - ***Le temps conversion*** : souvent exprimée en µs, c'est le temps qui sépare la demande d'acquisition de l'obtention du résultat
  - ***La plage de conversion*** : C'est la gamme de tension que l'on peut traiter (ΔV sur le schéma).
  - ***Le Format des données*** : Signé, non signé...



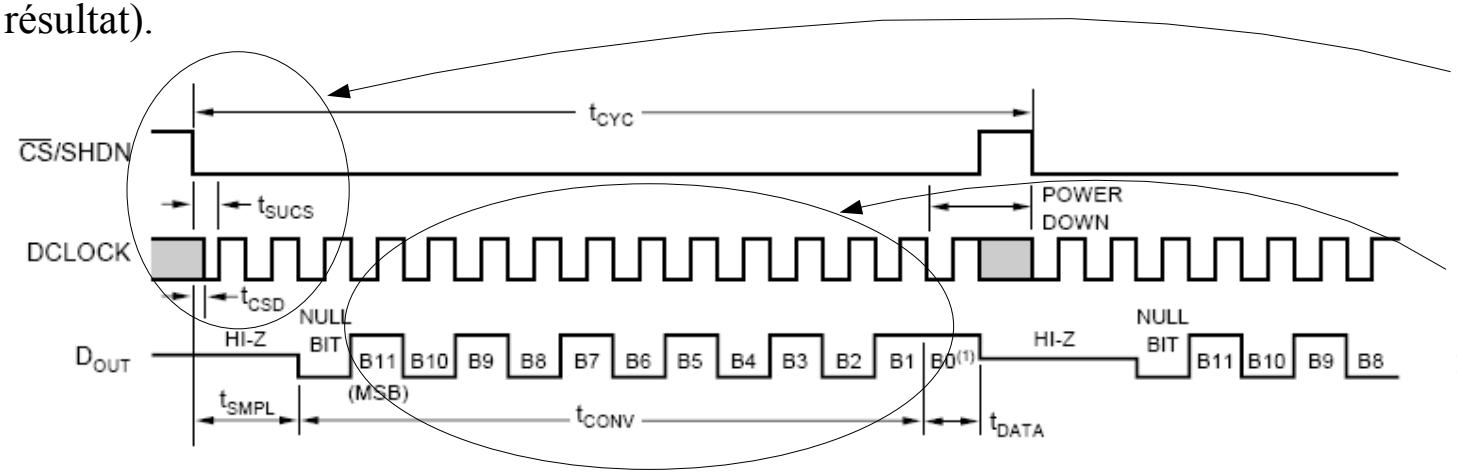
- L'ADC est un périphérique qui peut être interne ou externe.

Périphérique de µC : l'ADC (exemple d'ADC externe)

- ADC de type ADS7816
- 12 bits
- Fs_{Max} = 200 kHz (Tconv = 5µs)
- Interface série
- Plage d'entrée réglable (100mV à 5V)
- Format binaire naturel (non signé)



- P2.10 est en entrée (récupération des bits du nombre converti)
- P2.11 et P2.12 sont en sortie. Il contrôlent l'ADC (lancement de conversion, lecture du résultat).



Le front descendant de */CS* lance l'acquisition (elle dure 3 impulsions de *DCLOCK*.)

Les 12 fronts suivant de *DCLOCK* synchronisent la sortie des bits du résultat de conversion.

Cette gestion peut être faite par le µC, via le port P2.

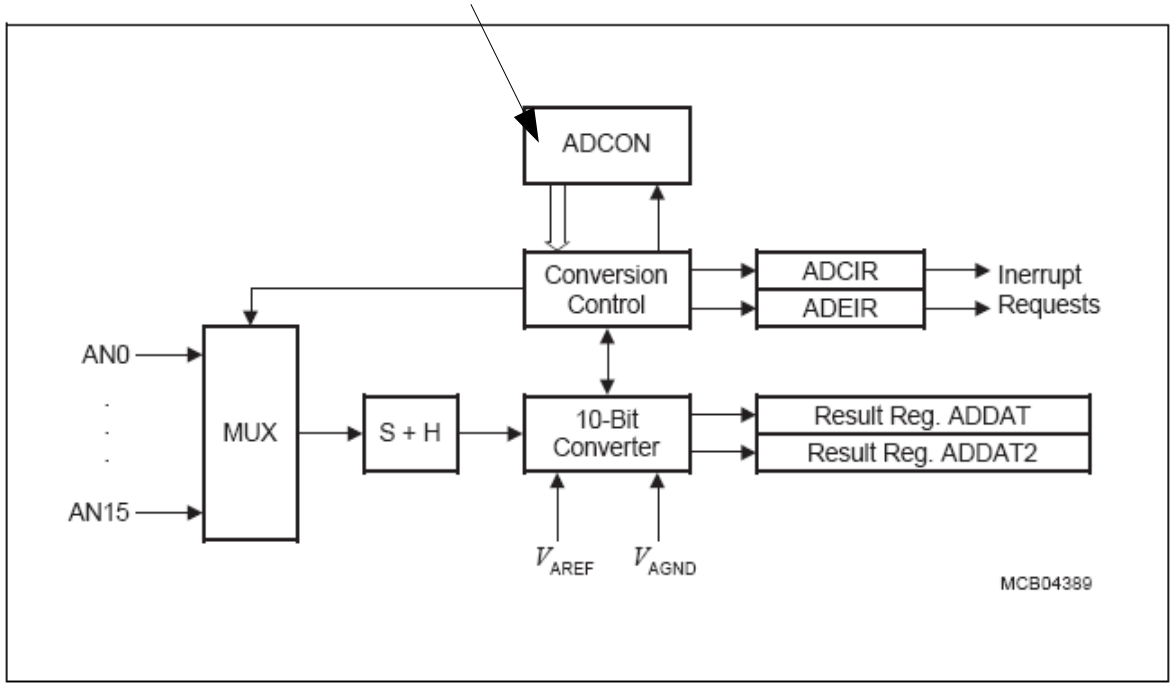
Périphérique de µC : l'ADC (exemple d'ADC interne)

L'ADC interne au µC (C167) possède un certain nombre de registres de configuration

Exemple de configuration :
la tension arrive sur la **voie 5** de l'ADC

- ADCH = 5
- ADST : c'est le bit de démarrage de l'ADC.
- ADBSY: c'est le bit (flag) qui indique l'ADC est en train de travailler

Deux possibilités de gestion de l'ADC :
- ***Scrutation***
- ***Interruption***



ADCON
ADC Control Register **SFR ($FFA0_H$/$D0_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		ADSTC		AD CRQ	AD CIN	AD WR	AD BSY	AD ST	-	ADM		ADCH			
rw		rw		rwh	rw	rw	rwh	rwh	-	rw		rw			

Périphérique de µC : Gestion ADC par scrutation

Algorithme global :

Faire toujours
  Acquérir la grandeur numérique, image de la masse (ADC)
  Créer les 3 chiffres : centaines, dizaines, unités à partir de la masse
  Afficher sur le LCD les 3 chiffres
  Temporiser 1s
Fin Faire

Algorithme d'acquisition par scrutation

- Configuration de l'ADC (ADCON=0x0005)
- Demarrage conversion : ADST=1
- Attendre la fin de conversion (ADBSY repasse à 0)
- Masse <= ADDAT&0x03FF

Masque pour ne garder que les 10 bits résultant

ADDAT
ADC Result Register **SFR ($FEA0_H/50_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				-	-	ADRES									
rwh				-	-	rwh									

Programme en C :

```
// Configuration ADC
ADCON=0x0005;

//Lancement acquisition
ADST=1;

// Attente résultat
while(ADST==1)
{}

// Lecture résultat
Masse=ADDAT&0x03FF;
```

Inconvénient de la scrutation :

le **µC perd du temps** pendant la conversion

Périphérique de µC : Gestion ADC par interruption

Interruption :
Il s'agit d'un ***déroutement*** de la tâche qu'exécute la CPU, non prévue. Les périphériques de µC sont capables de procéder à des ***demandes d'interruption***. Une ***fonction d'interruption*** associée est alors traitée par la CPU. Ensuite, le programme en cours reprend.