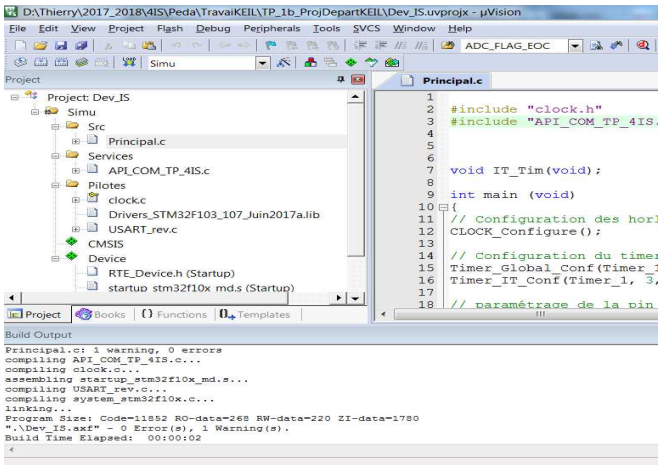
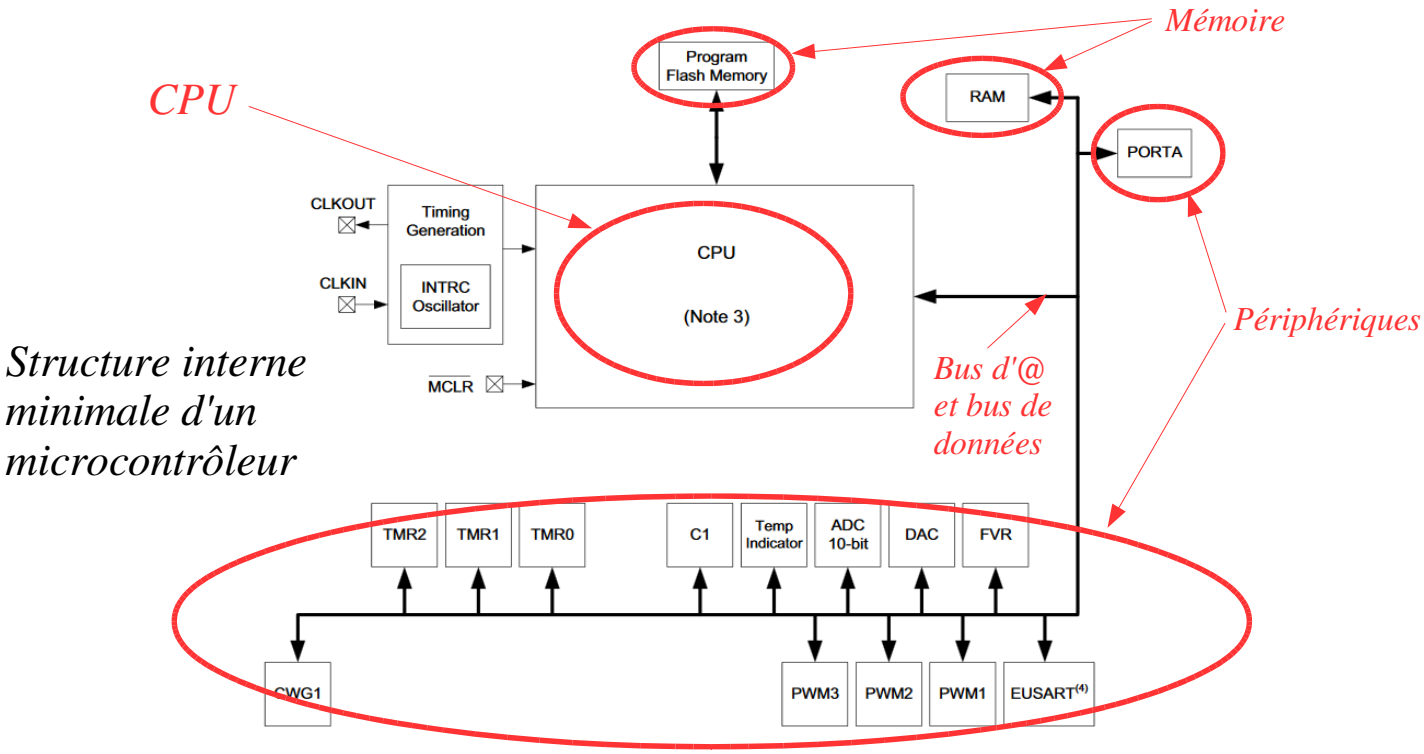
Microcontrôleur Hardware & Software
4 IDS

Hardware



Software

Microcontrôleur : Hardware

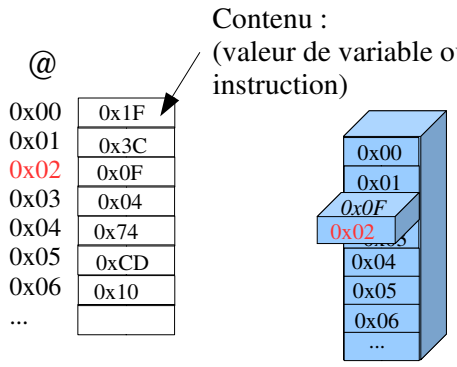


Structure interne minimale d'un microcontrôleur

Microcontrôleur : Hardware

Concept fonctionnel d'une mémoire

- Structurellement, une mémoire est un ensemble de ***registres*** (8 bits de large le plus souvent). Chaque registre code donc 1 ***octet***.
- Une mémoire possède une ***taille*** qui correspond au nombre d'octets qu'elle contient.
- Selon l'usage qui est fait, un octet dans une mémoire peut représenter une ***donnée*** (***valeur*** d'une ***variable***) ou peut représenter une ***instruction*** (la nombre stocké représente une instruction reconnaissable par le CPU et exécutable par le CPU).
- Les octets (valeurs de variable, ou instruction) transitent vers et depuis le CPU pour être traitées via un ***bus de donnée***.
- Chaque octet est identifié au sein de la mémoire par son ***adresse***. Le CPU utilise donc aussi un ***bus d'adresse***.
- Une mémoire, de manière simpliste peut être vue comme une « commode » contenant des tiroirs (adresses), chacun contenant un octet (data ou instruction).



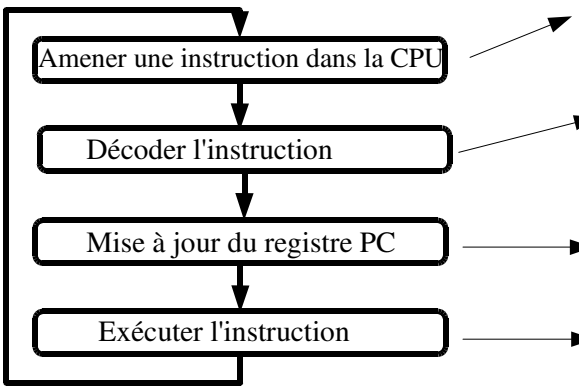
Si l'adresse 0x02 correspond à une variable, nous pouvons dire : la variable d'adresse 0x02 a pour valeur 0x0F.

Introduction à l'électronique 4 IDS

Microcontrôleur : Hardware

Le CPU (Central Processing Unit)

- Le CPU peut être relativement complexe. Mais sous sa forme minimaliste, il est en fait assez simple : c'est une séquenceur à 4 états !



Le CPU place sur le bus d'adresse la valeur contenue dans PC (*Program Counter*) : la mémoire fournit alors l'instruction rangée à cette adresse. Le CPU la transfère en son sein.

Le CPU analyse le code binaire de l'instruction et sait donc ce qu'il doit faire

Le PC est incrémenté pour préparer la prochaine recherche d'instruction

Le CPU procède à l'exécution

NB : le PC est un registre fondamental interne au CPU. Il contient l'adresse de la prochaine instruction à aller chercher en mémoire

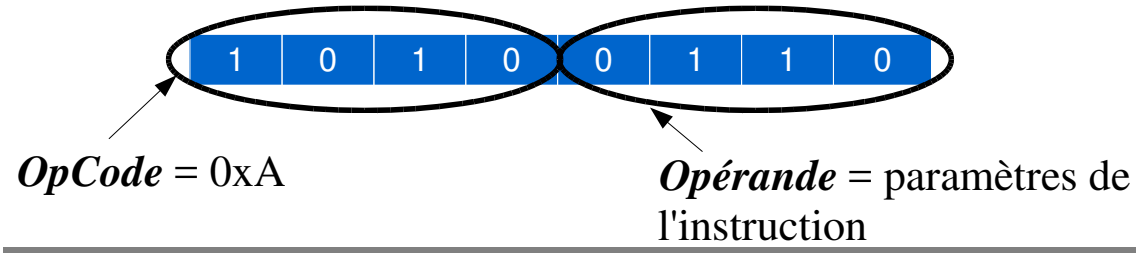
Microcontrôleur : Hardware

Le CPU : Le jeu d'instructions

Le CPU est caractérisé par son ***jeu d'instructions***. Il s'agit en fait de la palette d'actions dont il est capable.

Exemple :
- Ajouter deux valeurs contenues dans les registres i et j du CPU ,
- Sauter à une adresse si le résultat de l'opération qui précède est nul,
- Charger le registre interne n°i avec la valeur d'une variable d'adresse fixée,
...

Chaque **instruction** 8 bits (exemple que nous suivons depuis le début) possède des bits réservés regroupés dans un champ appelé ***OpCode***. Dans l'exemple ci dessous, on suppose que l'OpCode est fait sur 4 bits (le processeur a donc seulement 16 instructions dans son jeu) :

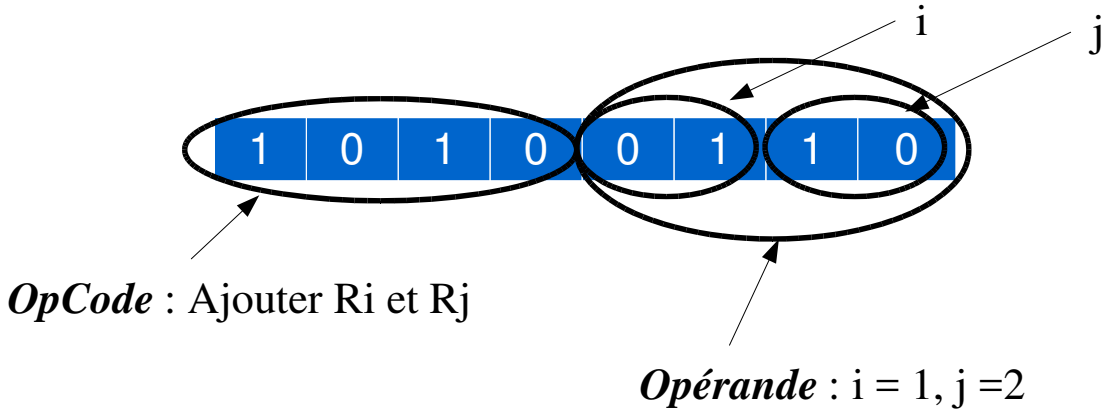


Microcontrôleur : Hardware

Le CPU : Le jeu d'instructions

Dans l'exemple précédent, le jeu d'instructions compte 16 instructions (2^4). Lorsque le CPU en est à l'étape du décodage, il prélève le champs ***OpCode*** de l'instruction en cours (ici 0xA) et la compare à une table de correspondance et en déduit qu'il faut faire une opération d'addition des registres i et j.

Ces derniers sont lisibles dans la partie opérande :



Microcontrôleur : Hardware

Périphérique de microcontrôleur

Pour faire simple :

- Le CPU c'est le cerveau du microcontrôleur (là où sont élaborées les commandes, là où sont interprétés les stimuli).
- Les périphériques sont les bras et les jambes du microcontrôleur. Ils permettent en particulier d'agir avec l'extérieur.

Exemple :

- Un **port d'entrée / sortie (GPIO)**, permet par exemple l'allumage d'une LED, la commande d'un relais, ou bien permet de connecter un bouton poussoir, un capteur de fin de position ….
- Une **UART** (Universal Asynchronuous Receiver Transmitter) permet d'établir une communication série avec un système externe : un GSM, un module Xbee, un second microcontrôleur …
- Un **ADC** qui permet de convertir une tension analogique en un nombre numérisé (discret donc) qui lui est proportionnel
- ….

Microcontrôleur : Hardware

Périphérique de microcontrôleur, Comment les utiliser ?

- L'utilisation des périphériques quels qu'ils soient doivent s'insérer dans l'architecture CPU du microcontrôleur. Ils sont donc reliés physiquement aux CPU par les bus de données et d'adresses,

- Du point de vue du programmeur, tout périphérique est une espace mémoire. En effet, aussi complexe que soit le programme exécuté par un microcontrôleur, TOUT se résume à des transferts de données entre le CPU, la mémoire et les périphériques.

Exemple d'une UART qui contient 3 registres donc vue par le CPU comme un petite RAM de seulement 3 octets :

@	
0xD0	0x10
0xD1	0x30
0xD2	0x1F

TRES IMPORTANT

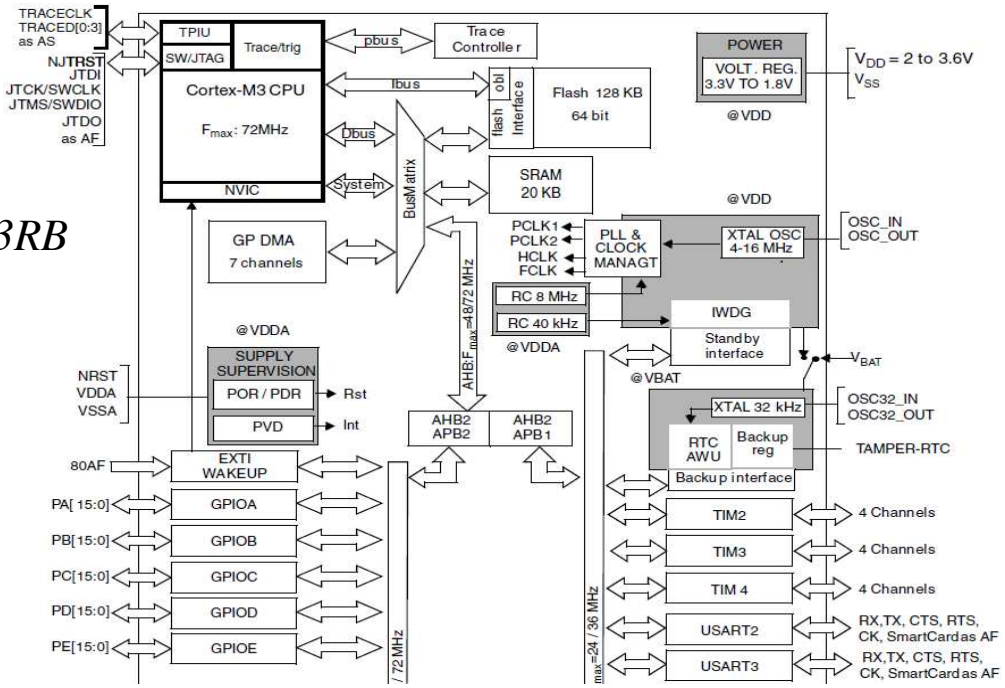
Contrairement à une simple RAM, chaque adresse a un rôle spécifique pour le périphérique. C'est précisément ce qui permet de contrôler le périphérique par programmation (écriture/lecture des registres) :

Rôles attribués :

- @ = 0xD0 : sa valeur correspond à la vitesse de transmission de l'UART,
- @ = 0xD1 : sa valeur est celle de l'octet à transmettre,
- @ = 0xD2 : sa valeur est celle de l'octet qui vient d'être reçu.

Microcontrôleur : Hardware

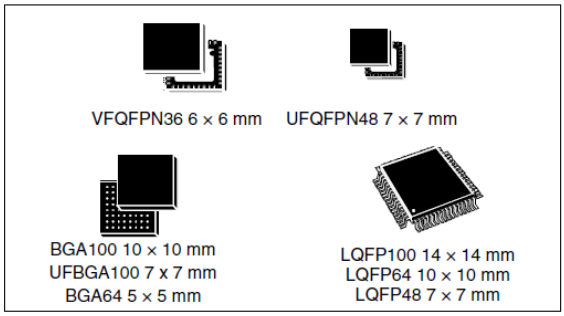
Le STM32F103RB



Microcontrôleur : Hardware

Features

- ARM® 32-bit Cortex®-M3 CPU Core
  - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 64 or 128 Kbytes of Flash memory
  - 20 Kbytes of SRAM
- Clock, reset and supply management
  - 2.0 to 3.6 V application supply and I/Os
  - POR, PDR, and programmable voltage detector (PVD)
  - 4-to-16 MHz crystal oscillator
  - Internal 8 MHz factory-trimmed RC
  - Internal 40 kHz RC
  - PLL for CPU clock
  - 32 kHz oscillator for RTC with calibration
- Low-power
  - Sleep, Stop and Standby modes
  - V_{BAT} supply for RTC and backup registers
- 2 x 12-bit, 1 µs A/D converters (up to 16



- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
- 7 timers
  - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
  - 16-bit, motor control PWM timer with dead-time generation and emergency stop
  - 2 watchdog timers (Independent and Window)
  - SysTick timer 24-bit downcounter
- Up to 9 communication interfaces
  - Up to 2 x I^2C interfaces (SMBus/PMBus)
  - Up to 3 USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)

Microcontrôleur : Software

Introduction

Un microcontrôleur est donc avant tout une architecture matérielle, composée :

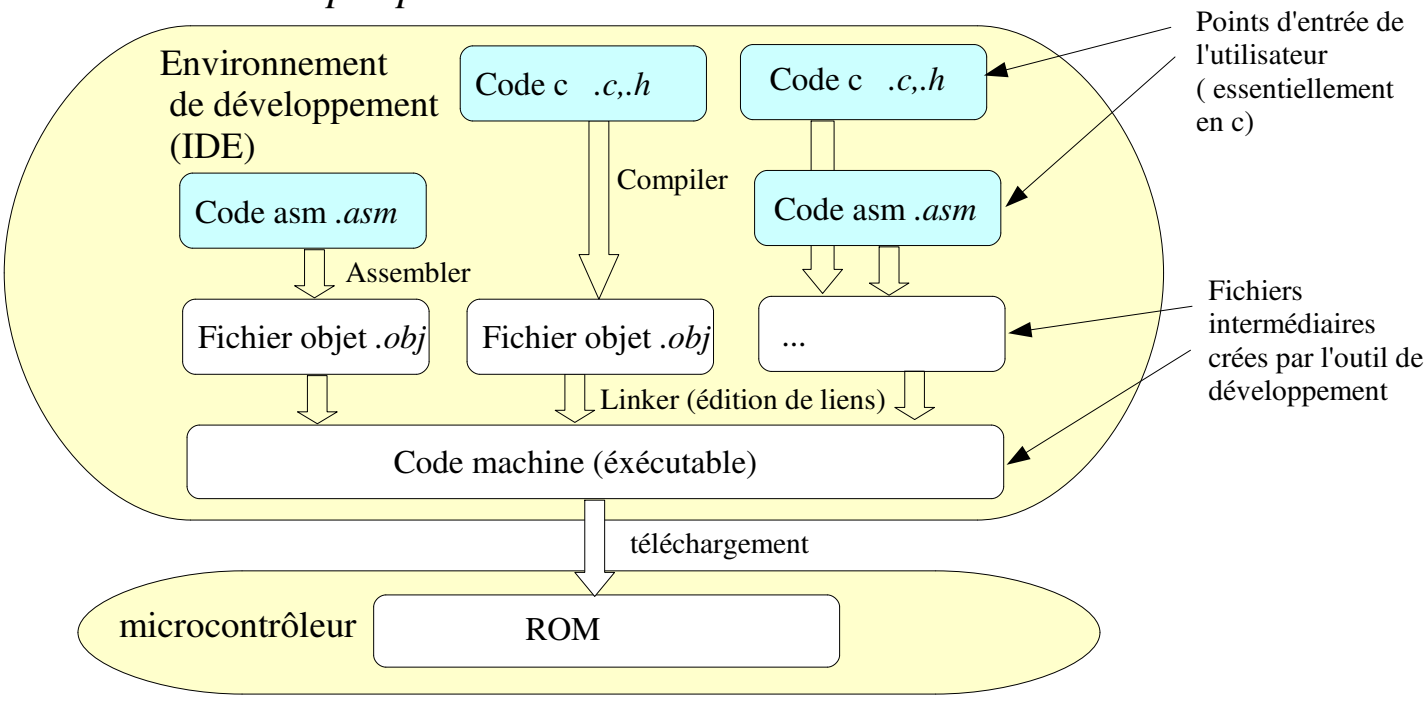
- D'un cœur, le CPU,
- De périphériques,
- **D'une mémoire (RAM et ROM)**

Comme nous l'avons vu, un microcontrôleur est un séquenceur qui a besoin d'un code exécutable (instruction avec op code, opérande, data …) pour réaliser un programme complexe : piloter une imprimante 3D par exemple.

Utiliser un microcontrôleur pour une application donnée consiste donc à ***remplir la ROM avec une suite d'instructions appropriées.***

Microcontrôleur : Software

Les diverses étapes pour obtenir un code exécutable



Microcontrôleur : Software

Les outils de programmation

- **Le compilateur C**
  Il traite un ***fichier source*** écrit en c pour en faire un fichier objet. Un fichier objet contient le code machine correspondant ainsi que des informations permettant d'établir des liens entre plusieurs fichiers objets (adresses de variables globales, de fonctions …)

- **L'assembleur**
  Il a le même rôle que le compilateur, il produit donc un fichier objet de la même nature, mais à partir d'un fichier source écrit en langage assembleur

- **L'éditeur de lien** (linker)
  Il prend en entrée de multiples fichiers objet : on peut donc générer le code global en le partageant en multiples fichiers source c ou assembleur. Il produit l'exécutable final, qui sera chargé en ROM sur le microcontrôleur.

Microcontrôleur : Software

Les drivers de périphériques

Comme nous l'avons vu, configurer un périphérique nécessite :

- De connaître le rôle du périphérique,
- De connaître ses principales caractéristiques,
- De connaître les registres de configurations (rôle de chaque registres, voir de chaque bit d'un registre),
- D'écrire un programme qui, en fonction des caractéristiques voulues du périphérique, écrit concrètement les bonnes valeurs au bon endroit.

→ Le constructeur de microcontrôleur (Hardware) fournit souvent (voire toujours) des **drivers** qui permettent d'utiliser n'importe quel périphérique simplement, à partir de fonctions écrites en c qui sont « parlantes » . Les deux dernières étapes citées précédemment (les plus lourdes pour un programmeur) sont donc supprimées !
→ Certaines entreprises développent elles-mêmes leurs drivers pour des raisons de fiabilité, de responsabilité en cas de défaillance.

Microcontrôleur : Software

Fichier .c et fichier .h

- Un fichier .c contient le programme écrit avec des instructions en langage c. Il contient des fonctions, des variables.

- ***Exemple :*** on souhaite écrire une fonction appelée *Addition* qui ajoute deux nombres de type int (32 bits), le corps de la fonction s'écrit :

```
int Addition (int a, int b)
{
int c ;      // déclaration d'une variable locale
c = a+b ; // calcul
return c   // retour de la fonction
}
```

Cette fonction sera écrite dans un fichier appelée *math.c*

Microcontrôleur : Software

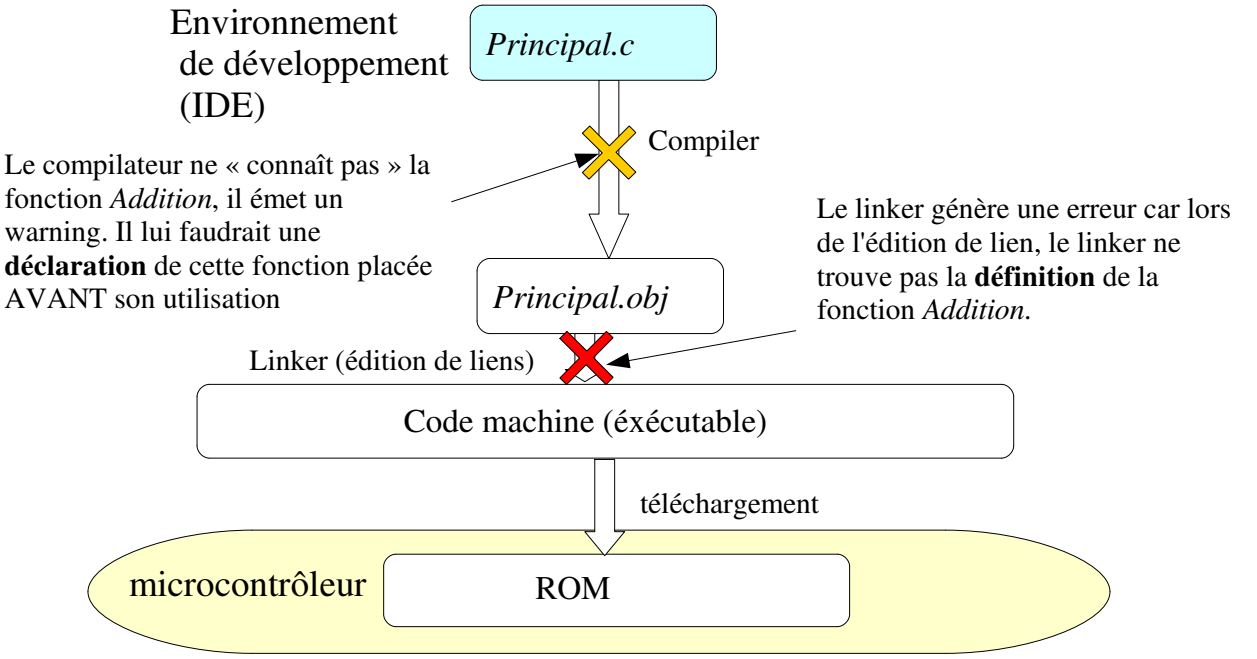
Fichier .c et fichier .h

- Supposons que l'on écrive un second fichier, *principal.c* dans lequel on place la fonction principale du projet (la fonction main) :

```
void main(void)  // void se traduit par « rien ». La fonction ne renvoie rien...
{
int n1,n2,n3 ;
n1=10 ;
n2=5 ;
n3 = Addition (n1,n2) ; // appel à la fonction Addition
}
```

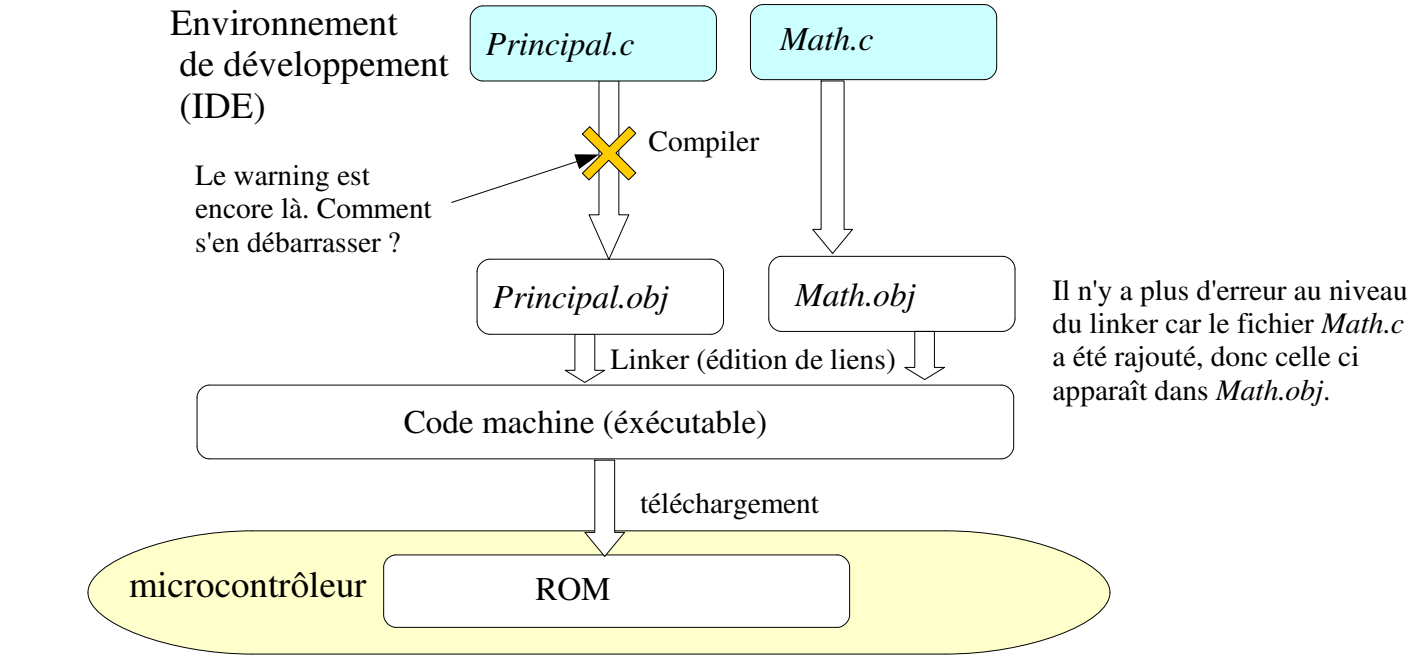
Microcontrôleur : Software

Fichier .c et fichier .h



Microcontrôleur : Software

Fichier .c et fichier .h



Microcontrôleur : Software

Fichier .c et fichier .h

- Première solution : on ajoute le **prototype** de la fonction Addition AVANT la fonction main. Cela fait office de **déclaration** → le compilateur ne génère plus de warning.

```
int Addition (int a, int b) ; // déclaration de la fonction

 void main(void)  // void se traduit par « rien ». La fonction ne renvoie rien...
{
int n1,n2,n3 ;
n1=10 ;
n2=5 ;
n3 = Addition (n1,n2) ; // appel à la fonction Addition
}
```

NB : ne pas confondre ***déclaration*** et ***définition*** !!

Microcontrôleur : Software

Fichier .c et fichier .h

- Seconde solution : L'auteur qui fournit le fichier *math.c* fournit aussi le fichier d'entête *math.h* dans lequel seront **déclarées** les fonctions **définies** dans *math.c,* que l'auteur souhaite voir utiliser par un autre.

Voici typiquement à quoi ressemble le .h :

math.h

```
#ifndef ADDITION_H
#define ADDITION_H

int Addition (int a, int b) ;

#endif
```

Directives de compilation

NB : en bleu, les directives de compilation ne SONT PAS des instructions c.
Comme son nom l'indique, une directive de compilation « oriente », « dirige » le compilateur dans son travail.

Sans rentrer dans le détail, ces directives OBLIGATOIRES dans un .h évitent les multiples inclusions

Microcontrôleur : Software

Fichier .c et fichier .h

- Seconde solution (suite) :

Reste à l'auteur de *Principal.c* à inclure le fichier .h dans son fichier. Cela se fait avec la directive de compilation #include :

Principal.c

```
#include « math.h »

void main(void)
{
int n1,n2,n3 ;
n1=10 ;
n2=5 ;
n3 = Addition (n1,n2) ;
}
```

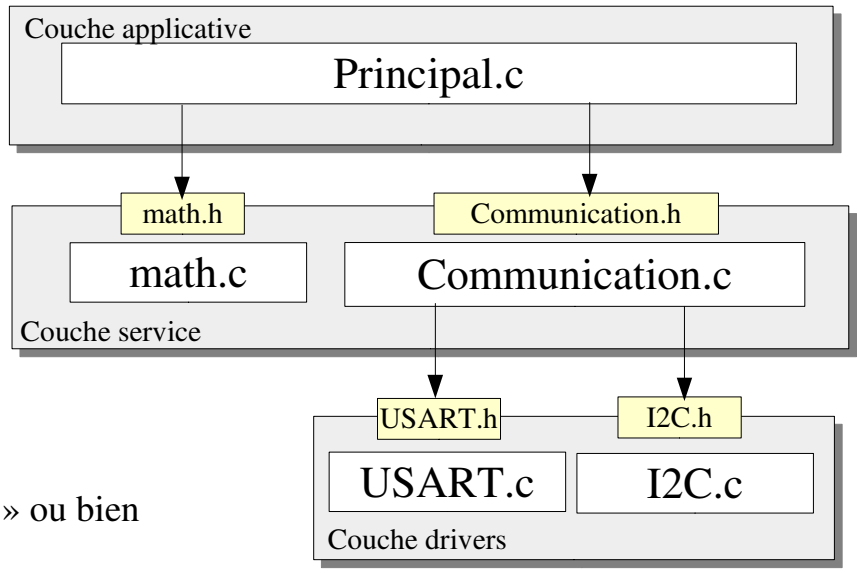
La directive *#include « math.h »* n'est qu'un simple copier-coller dans le fichier *Principal.c*

C'est beaucoup plus cohérent, et ça évite à l'utilisateur d'écrire toutes les déclarations de fonctions de math.h.

Microcontrôleur : Software

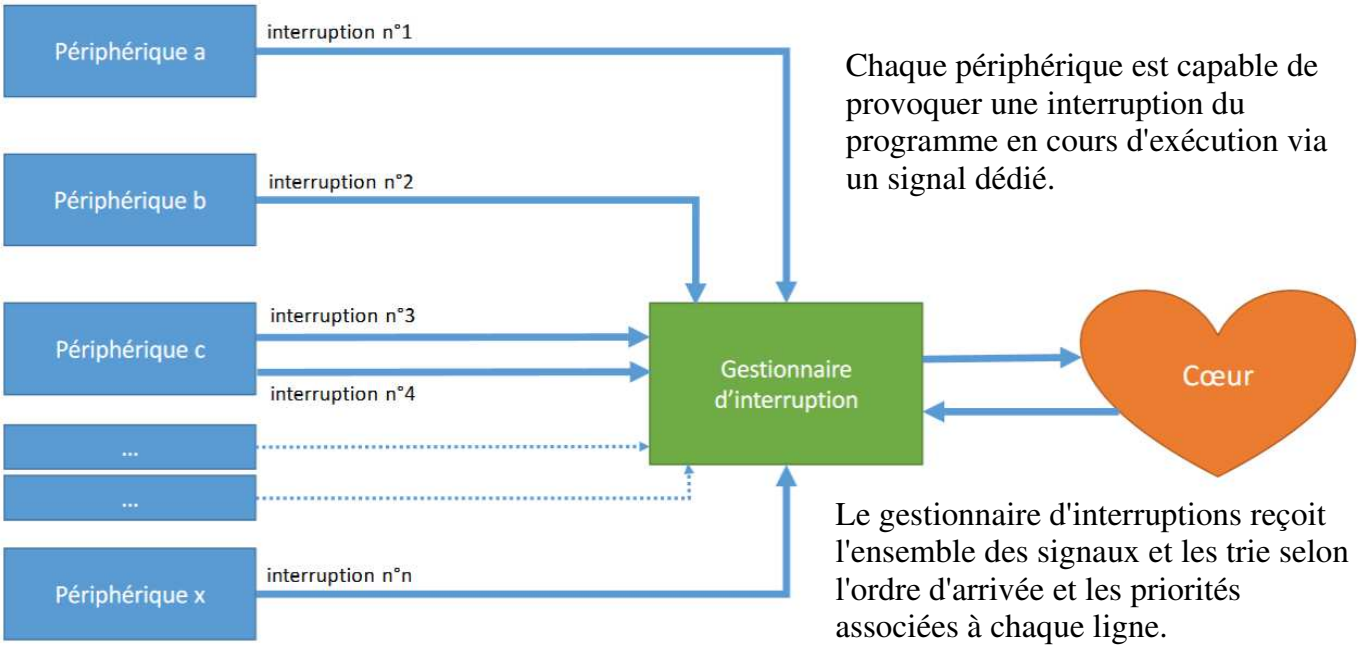
Fichier .c et fichier .h , structuration logicielle en couches

L'intérêt de disposer de fichiers *.c* et *.h* est de pouvoir faire très simplement de la programmation modulaire. En fonction des diverses dépendances on peut établir une structure logicielle en plusieurs couches :



Lire « *Principal.c* inclut *math.h* » ou bien « *Principal.c* utilise *math.h* »

Microcontrôleur : Les interruptions



Chaque périphérique est capable de provoquer une interruption du programme en cours d'exécution via un signal dédié.

Le gestionnaire d'interruptions reçoit l'ensemble des signaux et les trie selon l'ordre d'arrivée et les priorités associées à chaque ligne.

Figure 1 : Positionnement du gestionnaire d'interruption

Microcontrôleur : Les interruptions

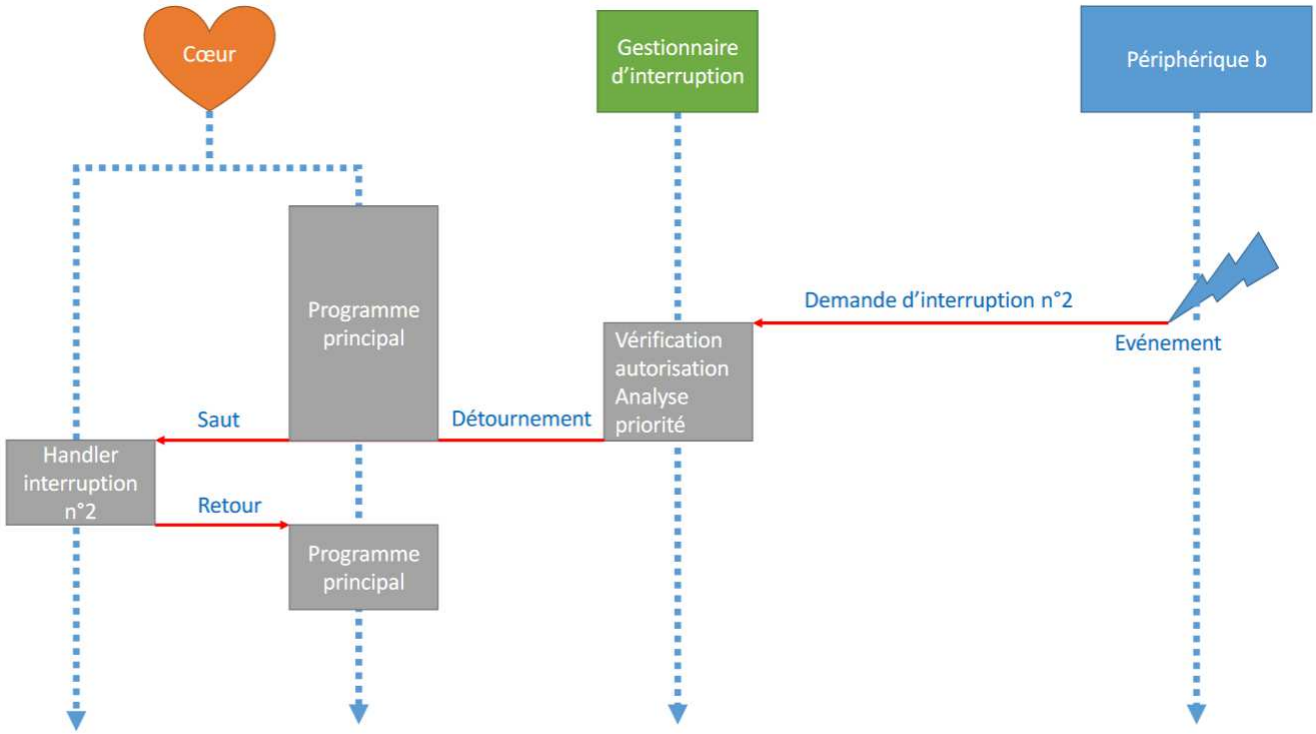


Figure 2 : Phases temporelles d'un traitement d'interruption

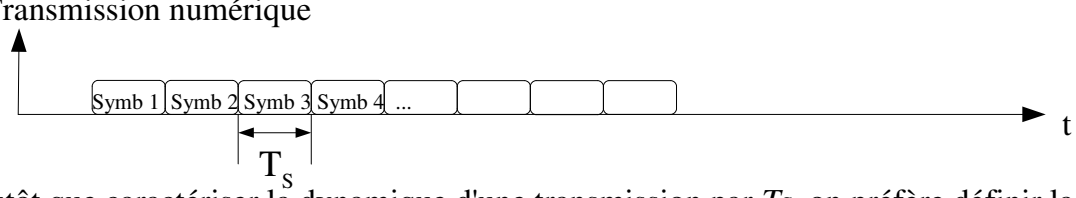
Microcontrôleur : UART

- L'UART *(Universal Asynchronuous Receiver Transmitter)* fait partie des systèmes de **transmissions numériques**,

- Une transmission numérique est une succession d'envois ou de réceptions de **symboles**,

- Un symbole est le **plus petit élément indivisible et stable,** transmis (ou reçu). Il porte un certain nombre de **bits** et possède une durée fixe, notée T_s (durée de symbole) durant laquelle le symbole ne change pas,

Transmission numérique



- Plutôt que caractériser la dynamique d'une transmission par *Ts*, on préfère définir la **vitesse de transmission**, ***R***, exprimée en ***bauds*** (***bds***).
Exemple : *une transmission de R= 9600 bds veut dire que 9600 symboles sont transmis (ou reçus) en une seconde, si un symbole contient 4 bits, le* ***débit binaire*** *vaut 38400 bits/s.*

Microcontrôleur : UART

- UART : C'est le périphérique le plus **élémentaire** qui permette d'**envoyer et de recevoir des données**,

- Les données sont émises un octet après l'autre, un **seul bit à la fois** : c'est une **liaison série** (Terminal série),

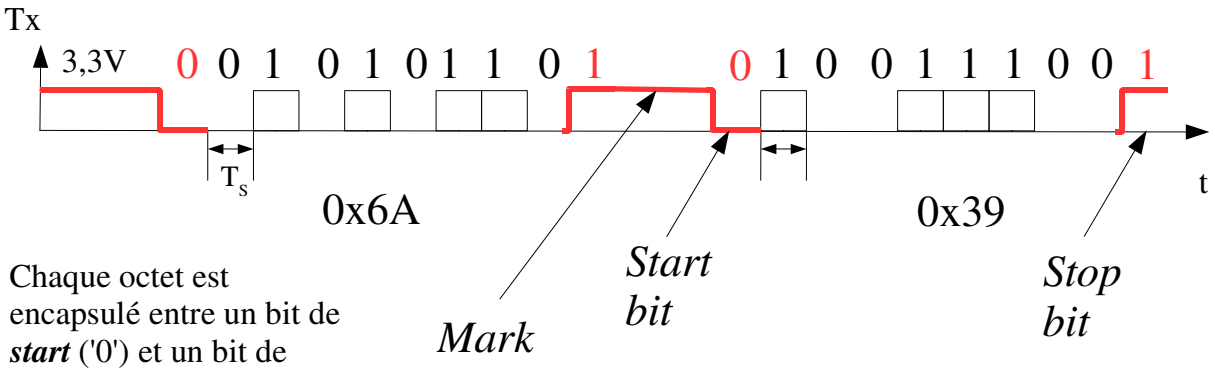
- Chaque symbole de la transmission ne compte donc qu'un seul bit. Le **débit binaire est donc égal à la vitesse de transmission** (c'est un cas particulier),

- L'UART possède **3 fils de connexion** élémentaires (d'autres peuvent être utilisés pour la signalisation) :
  - ***0V*** (la référence de tension),
  - ***Tx*** (**T**ransmission),
  - ***Rx*** (**R**éception).

Microcontrôleur : UART

- Format de transmission :

Exemple, soit à transmettre l'octet 0x6A (0b 0110 1010), puis 0x39 (0b 0011 1001). Le lsb (Least Significant Bit) est transmis en premier :



Chaque octet est encapsulé entre un bit de ***start*** ('0') et un bit de ***stop*** ('1'). Entre les deux, la ligne est au repos (***Mark***) égal à '1'

Le bit de ***start*** permet au récepteur de savoir qu'un nouvel octet est envoyé.

Microcontrôleur : ADC

- Un ADC permet la conversion analogique numérique,
- Exemple de principe, l'ADC à pesées successives :
  - Fonctionne comme les anciennes balances (avec des poids donnés)
  - Exemple :

La balance :

5 poids :
160g
80g
40g
20g
10g

Étendue de mesure :
160 + … = 2 . 160 -10 = 310g

Résolution :
10g

L'ADC :

5 poids :
2V
1V
0,5V
0,25V
0,125V

Étendue de mesure :
2 + … = 2 . 2 -0,125 = 3,875 V

Résolution :
125mV

Microcontrôleur : ADC

Un ADC à pesée successive est composé de :

- Un comparateur analogique (la balance …)
- Des tensions de références (les poids)
- Un sommateur
- Un séquenceur (système qui à partir des comparaison prend une décision)
- Un registre qui donne le résultat, ici 5 bits (ensemble des poids disposés sur la balance en fin de pesée ...)

Mesure d'une tension (en 5 coups, ADC 5 bits)

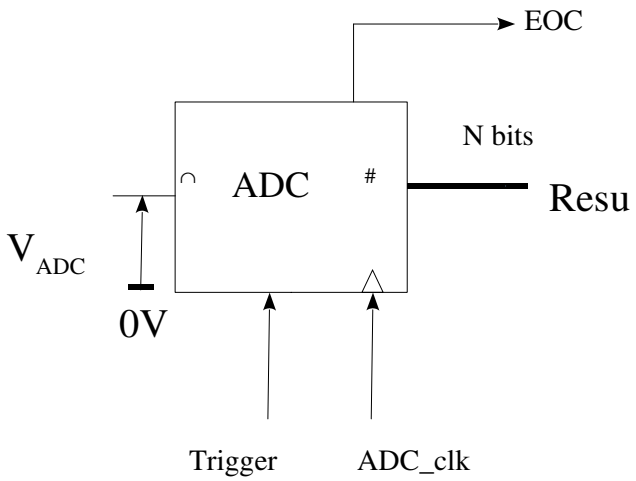
Ex U = 1,2V

1. essai avec un poids de 2V : 0b10000. U > 2V ? → Non → 0b00000
2. on ajoute un poids de 1V : 0b01000. U > 1V ? → Oui → 0b01000
3. on ajoute un poids de 0,5V : 0b01100. U > 1+0,5V ? → Non → 0b01000
4. on ajoute un poids de 0,25V : 0b01010. U > 1+0,25V ? → Non → 0b01000
4. on ajoute un poids de 0,125V : 0b01010. U > 1+0,125V ? → Oui → 0b01001

→ le résultat est 0b01001, soit 9. La valeur de tension équivalente mesurée est 9.0,125 = 1,125V → l'erreur est inférieure ou égale à la résolution (125mV ici)

Microcontrôleur : ADC

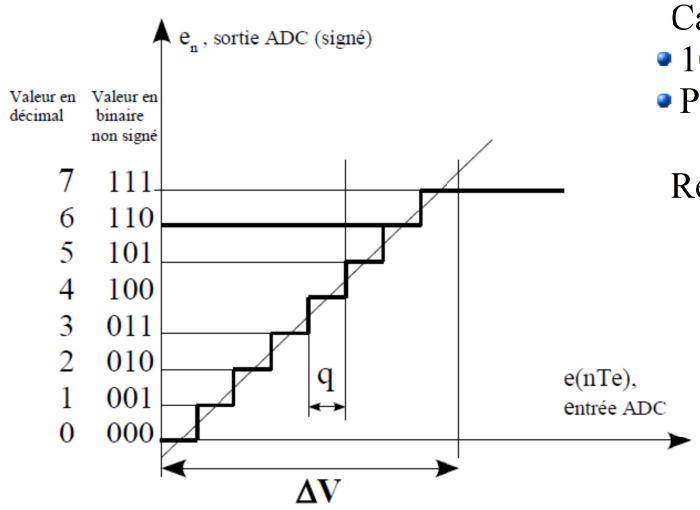
- Symbole associé à un périphérique ADC :



- Lancer la conversion (via l'entrée Trigger)
- Attendre la fin de conversion (12 steps)
- Lire le résultat

Microcontrôleur : ADC

- Loi de conversion A/N :



Exemple de conversion A/N :

Caractéristiques de l'ADC
- 10 bits de résolution
- Pleine échelle égale à 5V

Resu = $1{,}2 \cdot 2^{10} / 5 = 245$ (decimal)
= 0x0F5
= b 1111 0101