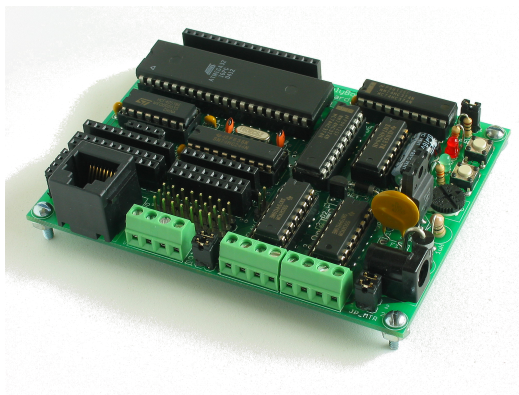
Microcontrôleur

Introduction

Nicolas Delanoue

Université d'Angers - Polytech Angers

POLYTECH°
ANGERS

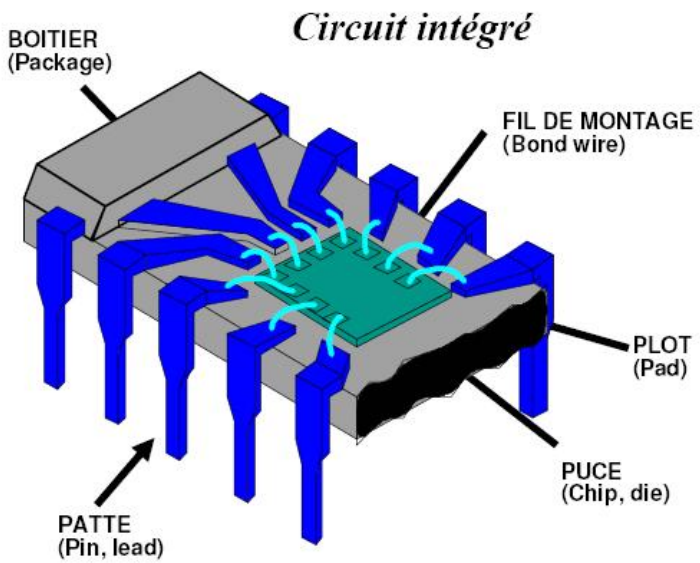


Définition

Un microcontrôleur est un circuit intégré contenant

- un microprocesseur,
- de la mémoire
- des périphériques

C'est donc un système programmable.



Plusieurs fabricants :

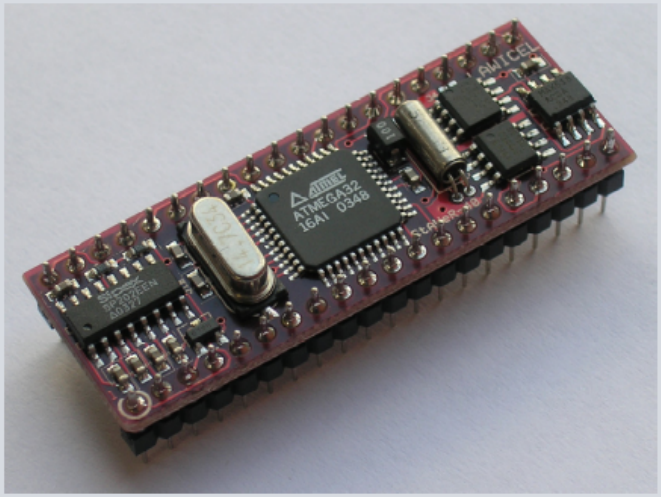
- Microchip (PIC)
- Atmel (ATMega et 8051)
- Intel, Philips, Siemens, Dallas, ST ...

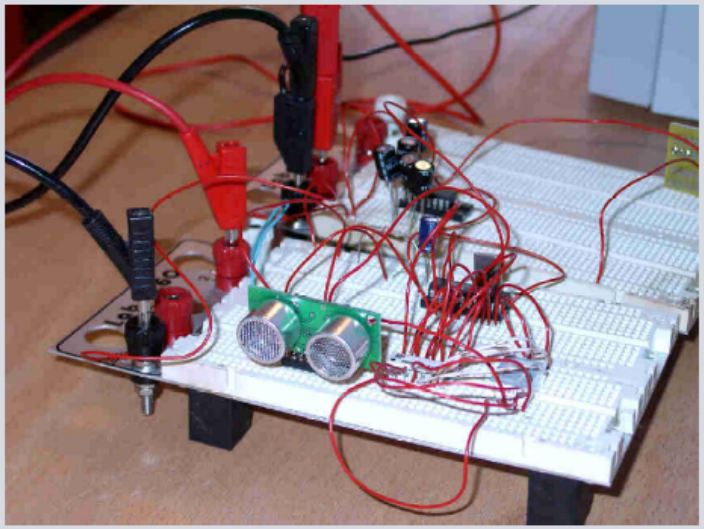
Vocation des microcontrôleurs

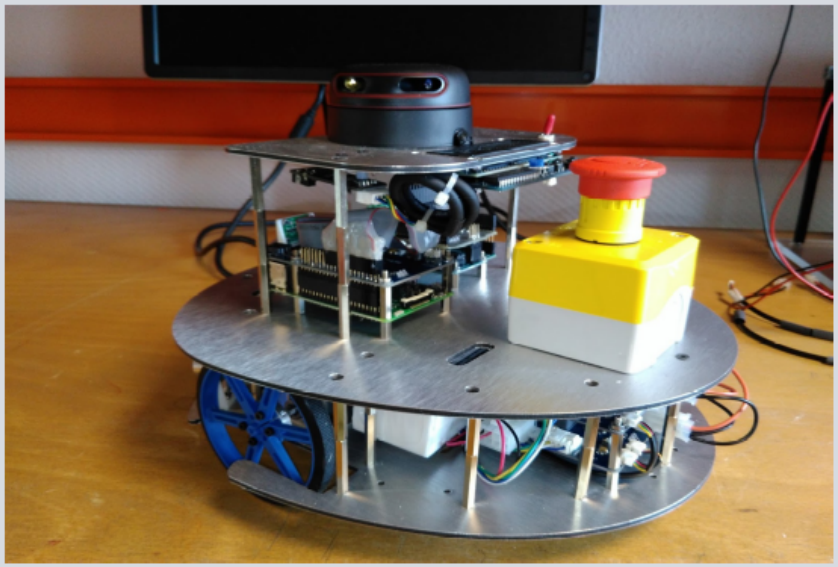
- Capteurs intelligents : mise en forme de signaux, conversions, ...
- Les distributeurs automatiques,
- Domotique, Électroménager, Robotique, Modélisme, Objets Connectés ...

Cas particulier

La carte Arduino est une carte à base d'ATMega principalement utilité pour le prototypage.


ATMEGA32
16AI 0348





Contenu de cet enseignement

- Rappels d'électronique numérique,
- Entrées-Sorties standards,
- Principe de fonctionnement d'un processeur,
- Caractéristiques et architecture d'un microcontrôleur,
- Timer, interruption, ...

Le tout illustré via Arduino UNO (ATMega328)

Remarque

Les microcontrôleurs relèvent de l'électronique numérique (digital electronics).

- Ils contiennent un processeur.
- En interne, ils n'exploitent que des signaux binaires dont les niveaux sont notés 0 et 1.

Définition

L'électronique numérique exploite des transistors en commutation (interrupteurs commandés électriquement).

Avec des transistors, on peut

- Stocker des informations (mémoire)
- Faire des opérations arithmétiques et logiques sur des données en binaire

Systèmes élémentaires

- portes logiques (et, ou, complément) [Combinatoire]
- bascules (systèmes à mémoire) [Sequentiel]

Définition

En architecture informatique, un mot est une unité de base manipulée par un microprocesseur. On parle aussi de mot machine. La taille d'un mot s'exprime en bits.

Exemple

Sur une architecture 8 bits, 00010100 est un mot.

Remarque

Un même mot peut être interprété de différentes façons. En effet, le mots de 8 bits 0100 1110 correspond

- au caractère *N* s'il est interprété comme de l'ASCII,
- au nombre 78 s'il est interprété comme un nombre écrit en base 2. En effet, $0100\ 1110 = 0\times2^7+1\times2^6+0\times2^5+0\times2^4+1\times2^3+1\times2^2+1\times2^1+0\times2^0$.

Objectifs

- Comprendre la représentation des nombres (et des données).
- Savoir écrire leur manipulation en langage C.

Problème de numération

Tout est binaire dans le processeur, mais en tant que développeur ou utilisateur, on saisit et on affiche dans d'autres bases . . .

Exemple

```
// Saisie de la valeur de la variable a0
short int a0 =12;
// Dans le microcontroleur a0 est stocké sous la forme
// du mot de 16 bits 00000000 00001100

// Affichage de la valeur de la variable a0
printf("%x",a0);
// Affiche le caractère à l'écran C car
// (12)_10 = (1100)_2 = (C)_16
```

Définition

Un *système de numération* est un ensemble de règles qui régissent une, voire plusieurs numérations données. De façon plus explicite, c'est un ensemble de règles d'utilisation des signes, des mots permettant d'écrire les nombres.

Exemple

Le codage des romains : $I, II, III, IV, V, VI, VII, \ldots$

Exemple 2 : système unaire

- Le système unaire est un système de numération permettant l'écriture des entiers naturels en ne disposant que d'un unique symbole représentant l'unité.

Définition

Un *système de numération* est un ensemble de règles qui régissent une, voire plusieurs numérations données. De façon plus explicite, c'est un ensemble de règles d'utilisation des signes, des mots permettant d'écrire les nombres.

Exemple

Le codage des romains : $I, II, III, IV, V, VI, VII, \ldots$

Exemple 2 : système unaire

- Le système unaire est un système de numération permettant l'écriture des entiers naturels en ne disposant que d'un unique symbole représentant l'unité.
- Un entier s'écrit par juxtaposition de la quantité correspondante de copies du symbole.

Définition

Un *système de numération* est un ensemble de règles qui régissent une, voire plusieurs numérations données. De façon plus explicite, c'est un ensemble de règles d'utilisation des signes, des mots permettant d'écrire les nombres.

Exemple

Le codage des romains : $I, II, III, IV, V, VI, VII, \ldots$

Exemple 2 : système unaire

- Le système unaire est un système de numération permettant l'écriture des entiers naturels en ne disposant que d'un unique symbole représentant l'unité.
- Un entier s'écrit par juxtaposition de la quantité correspondante de copies du symbole.
- Ainsi 6 s'écrit $||||||$ où $|$ est le symbole utilisé pour représenter l'unité.

Exemple 3 : notation positionnelle

La notation positionnelle est un procédé d'écriture des nombres, dans lequel chaque position d'un chiffre est reliée à la position voisine par un multiplicateur, appelé base du système de numération.

Exemple 3 : notation positionnelle

La notation positionnelle est un procédé d'écriture des nombres, dans lequel chaque position d'un chiffre est reliée à la position voisine par un multiplicateur, appelé base du système de numération.
Avec b un entier qui servira de base, un nombre entier x se décompose de façon unique via :

$$x = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0 \text{ avec } a_i < b$$

On écrira

$$x = (a_n a_{n-1} \ldots a_1 a_0)_b$$

Exemple 3 : notation positionnelle

La notation positionnelle est un procédé d'écriture des nombres, dans lequel chaque position d'un chiffre est reliée à la position voisine par un multiplicateur, appelé base du système de numération.
Avec b un entier qui servira de base, un nombre entier x se décompose de façon unique via :

$$x = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0 \text{ avec } a_i < b$$

On écrira

$$x = (a_n a_{n-1} \ldots a_1 a_0)_b$$

Les coefficients a_i sont appelés les digits de x en base b.

Exemples

- $(437)_{10} = 4 \times 102 + 3 \times 10 + 7 \times 1$
- $(437)_8 = 4 \times 82 + 3 \times 8 + 7 \times 1 = 4 \times 64 + 24 + 7 = (287)_{10}$
- $(00001001)_2 = 2^3 + 2^0 = 9$

Remarque

- En base 2, un Digit Binaire est un BIT pour BInary digiT.
- Il ne faut pas le confondre avec un BYTE qui est un octect, i.e. un mot de 8 bits.

Mots de taille finie, et alors ?

- En informatique, les données sont stockées sur un nombre fini de bits.
- Avec n digits dans une base B on peut décrire B^n valeurs différentes. En particulier,
  - Un octet binaire : 256 valeurs différentes,
  - Un mot 16 bits : 65536 valeurs différentes,
  - Un mot 32 bits : plus de 4 milliards de valeurs différentes.

Ordres de grandeur

On utilise souvent l'approximation

- $2^{10} = 1024 \simeq 1000 = 1K$,
- $2^{20} \simeq 1000000 = 1M$
- $2^{30} \simeq 1000000000 = 1G$
- Avec 32 bits, 4 Giga valeurs différentes

Changements de base, pourquoi ?

- Nous comptons en base 10,
- E.T. compte en base 8,
- Les ordinateurs s'appuient sur la base 2.

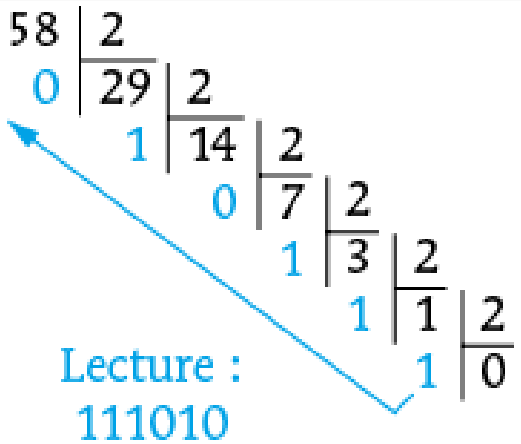
Conversion : base b vers base 10

On développe et additionne les puissances de b.
Exemple : $(134)_7 = 1 \times 7^2 + 3 \times 7^1 + 4 \times 7^0 = 49 + 21 + 4$.

Conversion : base 10 vers base b

Algorithme des divisions successives par b

Exemple : 58 s'écrit 11 1010 en base 2.



Théorème

Si $b' = b^n$ alors un digit en base b' correspond à n digits en base b

Exemple

- $(312)_4 = (11\ 01\ 10)_2$
- $(572)_9 = (12\ 21\ 02)_3$

Remarque

Les informaticiens aiment la base 16 car $2^4 = 16$ et donc 1 digit en hexa code 4 digits en base 2 (i.e. 4 bits).

- $(2B)_{16} = (0010\ 1011)_2$

Remarque

Ne jamais oublier qu'un processeur ne manipule concrètement que des mots binaires.

Remarque importante

La notation hexadécimale fournit souvent implicitement la taille utilisée en mémoire.

Exemple

- $(001A00F2)h$ est un mot de 32 bits (4 octets).
- $(04)h$ est un mot de 8 bits (4 octets).

En c

```
int x = 0x01001E00; // (en hexadécimale)
```

Théorème

Les techniques d'additions et de multiplications apprises, en base 10, à l'école élémentaire fonctionnent pour toutes les bases.

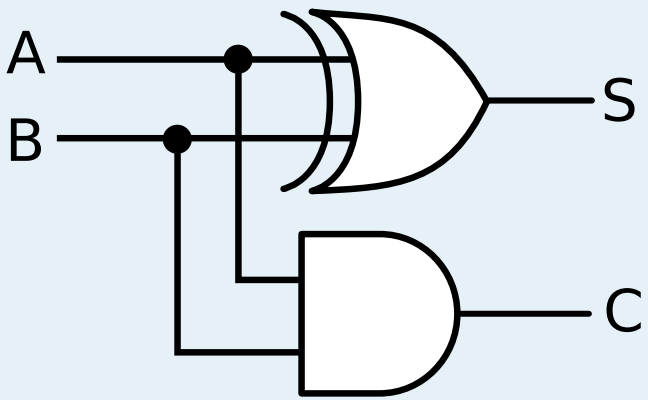
Exemple en base 2

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 1 \\ + & & 0 & 1 & 1 & 0 \\ \hline & . & . & . & . & . \end{array}$$

Table d'addition en base 2

+	0	1
0	0	1
1	1	10

Câblage qui réalise l'addition en base 2



Remarque

Les processeurs manipulent des octets (mot 8 bits), parfois des mots de 16 bits, 32 bits, 64 bits ...

Théorème

L'ensemble des entiers représentables avec n bits est fini (de cardinal 2^n).

- Sur 8 bits, on a 256 entiers différents.
- Sur 16 bits, on a 65536 entiers différents.
- Sur 32 bits, on a 4 294 967 296 entiers différents.

Parfois cela déborde

- En base 10 sur 2 digits, i.e. 00 à 99
  $63 + 54 = ??$
- Sur 8 bits (autrement dit en base 2 avec 8 digits)
  $130 + 130 = ??$
  Le nombre 130 est représentable sur 8 bits mais pas le résultat de 130+130.

Dans ce cas, on parle d'integer overflow.

Codages pour les nombres entiers

Il existe différents codages pour les nombres entiers :

- les entiers signés,
- les entiers non signés,
- les entiers de Delanoue ...

Définition

Avec la convention des entiers non signés, pour coder un entier x de $\mathbb{N}$, on utilise sa représentation binaire.

Codages pour les nombres entiers

Il existe différents codages pour les nombres entiers :

- les entiers signés,
- les entiers non signés,
- les entiers de Delanoue . . .

Définition

Avec la convention des entiers non signés, pour coder un entier x de $\mathbb{N}$, on utilise sa représentation binaire.

Exemple de codage avec la convention “entiers non signés”

- Par exemple, le mot de 8 bits 0000 1101 code le nombre 13.
- Par exemple, le mot de 16 bits 0000 0000 0000 1101 code aussi le nombre 13.

Définition

Avec la convention des entiers signés sur n bits, pour coder un entier x de $\mathbb{Z}$ (potentiellement négatif), on utilise le codage suivante :

- Si x est positif, alors on utilise le codage des entiers non signés,
- sinon, on calcule $\tilde{x} = 2^n - |x|$, on utilise le code (entiers non signés) de $\tilde{x}$ pour coder x.

Définition

Avec la convention des entiers signés sur n bits, pour coder un entier x de $\mathbb{Z}$ (potentiellement négatif), on utilise le codage suivante :

- Si x est positif, alors on utilise le codage des entiers non signés,
- sinon, on calcule $\tilde{x} = 2^n - |x|$, on utilise le code (entiers non signés) de $\tilde{x}$ pour coder x.

Exemple

- Avec 8 bits, le nombre 13 est codé par le mot 0000 1101

Définition

Avec la convention des entiers signés sur n bits, pour coder un entier x de $\mathbb{Z}$ (potentiellement négatif), on utilise le codage suivante :

- Si x est positif, alors on utilise le codage des entiers non signés,
- sinon, on calcule $\tilde{x} = 2^n - |x|$, on utilise le code (entiers non signés) de $\tilde{x}$ pour coder x.

Exemple

- Avec 8 bits, le nombre 13 est codé par le mot 0000 1101
- Avec 8 bits, le nombre $x = -7$ est codé par le mot 1111 1001. En effet, $\tilde{x} = 256 - 7 = 249 = (11111001)_2$

Remarque

$\tilde{x}$ est appelé le complément vrai de x.

Exemples

Avec le codage des entiers non signés, on a

- $+23$ est représenté en binaire par le mot binaire 00010111 ou bien le mot hexa 17
- -23 est représenté en binaire par le mot binaire 11101000 ou bien encore le mot hexa $E9$

En pratique, on pourra indifféremment les codes c suivant

```
char x=23;
char y=-23;
```

```
char x=0x17;
char y=0xE9;
```

Regardons le code c suivant :

```c
int main()
{
char a='n';
printf("a=%d \n",a);
a+=20;
printf("a=%d \n",a);
}
```

Son exécution

```
a=110
a=-126
```

Étendue

Avec ce codage des nombres signés,

- sur 8 bits, on peut coder les nombres de -128 à +127.
- sur 16 bits, on peut coder les nombres de -32768 à +32767.

Proposition

Avec ce codage des nombres signés, le bit de poids fort du mot indique le signe du nombre codé.

Exemple

- Le mot 01010111 code un entier positif,
- Le mot 11010111 code un entier négatif.

Remarque

Les débordements sont aussi possibles.

Exemple

$$100 + 100 = (64)_{16} + (64)_{16} = (C8)_{16} = 200.$$

- Le résultat ne peut pas être écrit avec le codage entier signé sur 8 bits.
- Un microprocesseur 8 bits obtiendra comme résultat -56.

Exemple de nombres à virgule

- $(10,03)_{10} = 1 \times 10^1 + 0 \times 10^0 + 0 \times 10^{-1} + 3 \times 10^{-2}$
- $(4,3)_5 = 4 \times 5^0 + 3 \times 5^{-1} = 4,2$
- $(10,101)_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$
  $= 2 + \frac{1}{2} + \frac{1}{8} = 2.625$

Changement de bases avec $b' = b$

$(3,4)_{16} = (0011\ 0100)_2 = (11,01)_2 = 3,25.$

- $(0.1)_2 = 0.5$
- $(0.1)_3 = 0.333333333\ldots$

Nombre à virgules flottantes

La virgule flottante est une méthode d'écriture de nombres fréquemment utilisée dans les ordinateurs, équivalente à la notation scientifique en numération binaire. Elle consiste à représenter un nombre par :

- un signe (égal à -1 ou 1) ;
- une mantisse ;
- et un exposant (entier relatif, généralement borné).

Un tel triplet représente le nombre suivant

$$signe \times mantisse \times base^{exposant}$$

Exemple

En base 10 avec une mantisse sur 2 digits, un exposant sur 1 digit signé, le nombre -120 s'écrit $-1 \times 12 \times 10^1$. Par conséquent, il est représenté par le triplet :

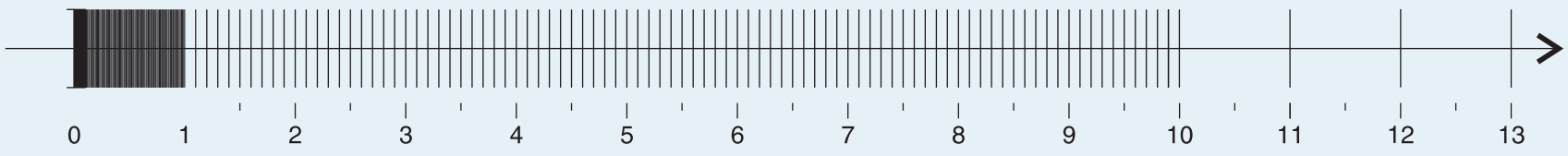
- signe = -1,
- mantisse = 12 ;
- exposant = 1.

Exemples

En base 10 avec une mantisse sur 2 digits, un exposant sur 1 digit

- Peut-on représenter 100 ? 35 ? 10,5 ? 0,0025 ? 5200 ? 101 ?

Représentation graphiques des flottants



Théorème

Seul une partie finie des nombres réels est représentable avec des flottants.

Définition

Sous b un entier (qui servira de base), n et p deux entiers. L'ensemble des nombres suivants est appelé l'ensemble des nombres flottants en base b avec n digits de mantisses et q digits pour l'exposant :

$$
\begin{aligned}
F_{n,q}^{b} = \{ \quad & x \in \mathbb{R}, \ \ x = \pm m \cdot b^{p} \\
& m \text{ un décimal sous la forme } 0, a_0 a_1 \dots a_n, \\
& \text{où } a_i \in \{0, \dots, b\} \text{ et } b^{-1} \leq m < 1 \\
& \text{et } p \text{ un entier à } q \text{ chiffres}\}
\end{aligned}
$$

La norme IEEE 754

Format standard :

$$\pm 1, \mathit{mantisse} \cdot 2^{\mathit{exposant}}$$

La norme IEEE754 imposent

- pour les architecture 32 bits
  - un bit de signe,
  - 8 bits pour l'exposant,
  - 23 bits pour la mantisse.
- pour les architecture 64 bits
  - 1 bit de signe,
  - 11 bits pour l'exposant,
  - 52 bits pour la mantisse.

Code c

```c
int main()
{
float f=12.75;
unsigned char * pUC;
pUC=(unsigned char*) &f;
printf("%u %u ", pUC[3],pUC[2]);
printf("%u %u\n",pUC[1],pUC[0]);
}
```

Son exécution

```
65 76 0 0
```

Code c

```
int main()
{
float f=12.75;
unsigned char * pUC;
pUC=(unsigned char*) &f;
printf("%u %u ", pUC[3],pUC[2]);
printf("%u %u\n",pUC[1],pUC[0]);
}
```

Son exécution

```
65 76 0 0
```

Explications

$12.75 = (1.10011)_2 \times 2^3$

Avec IEEE flottant 32 bits : 12.75 est représenté par le mot hexa 41 4C 00 00 et $(41)_{16} = 65$, $(4C)_{16} = 76$

Exemple

12.75 codé $(41\ 4C\ 00\ 00)_h$
$6.375 = 12.75/2 = (1.10011)_b \times 2^2$
Seul l'exposant change, sa représentation IEEE flottant est
$40\ CC\ 00\ 00$

```
int main()
{
unsigned char tab[4]={0,0,204,64};
float * pF=(float *)tab;
printf("%f\n",*pF);
}
```

Son exécution

```
6.375
```

Résumé

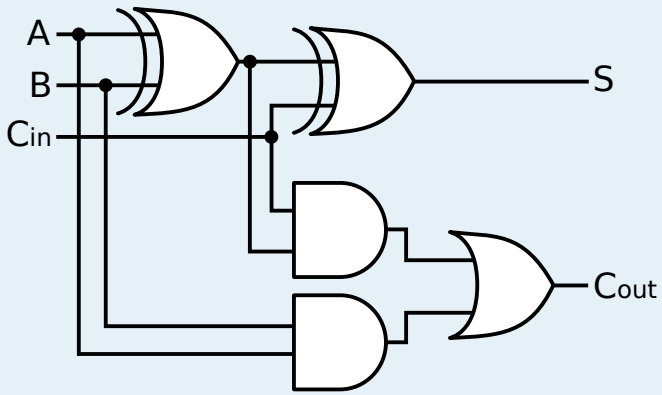
On peut coder et stocker dans des mots binaires :

- des entiers naturels,
- des entiers relatifs,
- des nombres à virgule flottante,
- des codes de caractères (table ASCII),
- tout ce qu'on veut, à partir du moment où on sait comment on l'interprète ...

Remarque

- Les processeurs savent faire des opérations arithmétiques et logiques sur ces quantités en binaire.
- la somme d'entiers et de flottants est toutefois réalisée différemment.

Câblage qui réalise l'addition en base 2 avec une potentielle retenue (Full adder)



Câblage qui réalise l'addition de deux mots de 4 bits