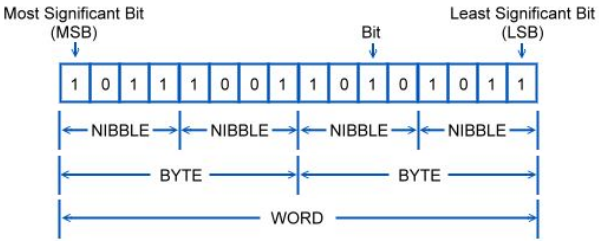
C5-160551-INFO
Objet Connectés : Programmation microcontrôleur
TD1

©B. Besserer

année universitaire 2021-2022

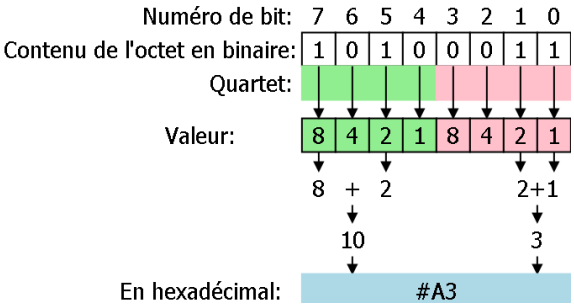
1 Rappels numération (binaire, hexadécimal)



A connaître par coeur : Les termes utilisés pour désigner les mots binaires en fonction de leur taille : Byte, Word et leur position (MSB, LSB)

A connaître par coeur : le code binaire pour les nombres de 0 à 15 et leur correspondance en hexadécimal.

$0_{16} = 0000_2 = 0_{10}$	$8_{16} = 1000_2 = 8_{10}$
$1_{16} = 0001_2 = 1_{10}$	$9_{16} = 1001_2 = 9_{10}$
$2_{16} = 0010_2 = 2_{10}$	$A_{16} = 1010_2 = 10_{10}$
$3_{16} = 0011_2 = 3_{10}$	$B_{16} = 1011_2 = 11_{10}$
$4_{16} = 0100_2 = 4_{10}$	$C_{16} = 1100_2 = 12_{10}$
$5_{16} = 0101_2 = 5_{10}$	$D_{16} = 1101_2 = 13_{10}$
$6_{16} = 0110_2 = 6_{10}$	$E_{16} = 1110_2 = 14_{10}$
$7_{16} = 0111_2 = 7_{10}$	$F_{16} = 1111_2 = 15_{10}$



Conversion : Utiliser des calculatrices, sur smartphone ou ordinateur. Certaines applications disposent d'un "mode programmeur" permettant de définir la taille du mot binaire, ce qui est utile pour les nombres signés.

Écriture des nombres hexadécimaux et binaire en langage C : La convention pour l'hexadécimal est de précéder le nombre par `0x`, par exemple `0x0020 = 32` en décimal. La notation pour le binaire `0b01010111` n'est pas un standard et n'est pas supportée par tous les compilateurs (GCC l'autorise, et la notation peut donx être utilisé dans le cadre du framework Arduino).

Codage binaire des nombres entiers signés Principe du codage "en complément à 2" :

- on exprime directement le nombre en binaire si il est positif,
- on calcule le complément à 2 d'un nombre négatif. Par définition, le complément à 2 sur n bits d'un nombre dont la valeur absolue est A est :

$$A' = 2^n - A$$

Si on choisit de calculer le complément à 2 après conversion en binaire, on remarquera que le complément à 2 est aussi le complément logique (appelé encore complément à 1) + 1. Le bit de poids fort (MSB) d'un nombre exprimé en complément à 2 correspond à un bit de signe : si MSB = 1, le nombre est négatif, si MSB = 0, le nombre est positif. Cela permet de déterminer l'opération à réaliser pour retrouver la valeur du nombre codé, qui s'obtient de la même manière puisque si $A' = 2^n - A$, on a $A = 2^n - A'$.

Sans utiliser de calculatrice ou application de conversion :

- 255 décimal à convertir en hexadecimal :
- AA00 hexadécimal à convertir en binaire et en decimal :

Donnez les codes binaires pour les nombres suivants (représentation binaire signée sur `8 bits`) :

- 127
- 1
- -128
- -1

· 100 (64 + 32 + 4) ☐ -100 ☐

```
+255   0xFF 1111 1111

0xAA00 1010 1010 0000 0000 = 10 x 4096 + 10 x 256 + 0 x 16 + 0 = 40960 + 2560 = 43520

+127   0x7F 0111 1111
....
 1     0x01 0000 0001
 0     0x00 0000 0000
-1     0xFF 1111 1111
...
-128   0x80 1000 0000
```

2 Opérateurs logiques

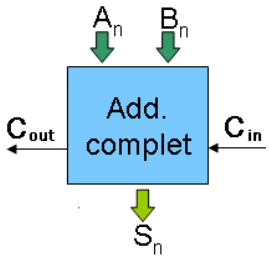
1. Un master dans une université propose des UE d'enseignement "majeures" (correspondant à la discipline) nommées X, Y et Z, ainsi que des UE "mineures" nommées A et B. L'étudiant obtient son diplôme (D) si il obtient toutes les UE majeures de X à Z ou s'il obtient 2 UE majeures de son choix et les 2 UE mineures. La variable logique correspondant à une UE réussie prends la valeur 1.

   - Écrire D sous la forme d'une équation logique : D = ☐
   - On doit déterminer si D est vrai en écrivant un programme informatique en langage C. On suppose que 5 variables logiques X, Y, Z, A et B sont déclarées et initialisées selon les résultats de l'étudiant dont on examine le cas. Le programme doit afficher "diplôme obtenu" ou "diplôme non obtenu".
   - Écrivez l'équation logique en respectant la syntaxe du langage C. La structure du programme serait donc :

```
D = ....
if(D) printf("diplôme obtenu");
else printf("diplôme obtenu")
```

2. Effectuer une addition binaire correspond à résoudre un problème de logique combinatoire. En base 10, on apprends à faire une addition au niveau des unités, puis de continuer avec les dizaine en reportant éventuellement la retenue (si l'addition sur les unités à dépassé 10), etc...

   En binaire, c'est pareil : on effectue l'addition sur un "rang" donné, et il faut tenir compte de la retenue en provenance du rang inférieur et propager le cas échéant la retenue vers le rang supérieur. Comme chaque opérande ne peux prendre que la valeur 0 ou 1, c'est un problème de logique on peux donc déterminer la table de vérité d'un additionneur.
   Soit A et B les deux nombres binaires à additionner (sur 1 seul bit, au rang n), C_{in} la retenue du rang inférieur, S le résultat (somme au rang n) et C_{out} la retenue à propager au rang supérieur :

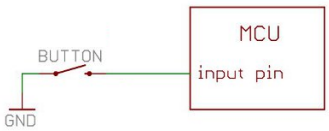


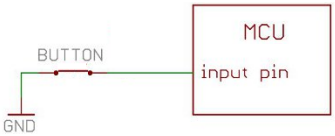
   - Combien d'équation(s) faut-il déterminer ? ☐
   - Combien de variables logiques interviennent dans chaque équation ? ☐
   - Combien de combinaisons possibles pour chaque équation (donc de lignes dans la table de vérité) ? ☐
   - Établir les tables de vérité (pas de reponse à donner sur Moodle, la correction sera faite en TD)

   2 sorties (somme, retenue sortante) donc deux équations à déterminer
   3 entrées (A_n, B_n et retenue entrante)
   Nombre de combinaisons binaires possibles : $2^3 = 8$, donc 8 lignes dans la table de vérité. Correction : `https://fr.wikipedia.org/wiki/Additionneur` voir additionneur complet.

3 Entrées / Sorties

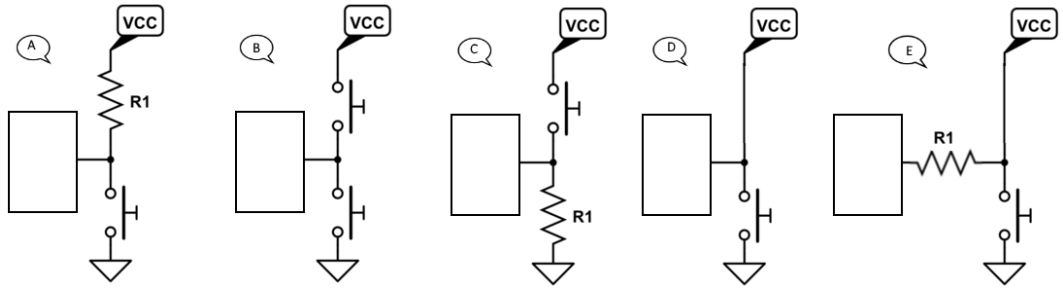
1.





Soit le montage ci-dessus. La broche (GPIO du microcontrôleur) est configurée comme une **entrée**.
A gauche, l'utilisateur n'appuie **pas** sur le bouton. Quel est alors le niveau logique qui sera lu sur la broche (input pin) : ☐ Niveau Haut (1 logique) ☐ Je ne sais pas, c'est aléatoire ☐ Niveau Bas (0 logique)
A droite, on considère que l'utilisateur appuie sur le bouton poussoir. Quel est le niveau logique qui sera lu sur la broche (input pin) : ☐ Niveau Bas (0 logique) ☐ Je ne sais pas ☐ Niveau Haut (1 logique)

A gauche : état indéterminé (fil "en l'air" = antenne = état logique aléatoire compte tenu de la technologie CMOS des microcontrôleurs)
A droite : niveau bas (0)

2. 

Parmi les montages de la figure précédente, lequel(s) présente(ent) un risque électrique ?
☐ A ☐ B ☐ C ☐ D ☐ E
3. Parmi ces montages, lequel(s) permet(tent) au microcontrôleur de connaître l'état du bouton poussoir ?
☐ A ☐ B ☐ C ☐ D ☐ E
Les montages B, D et E présentent un risque électrique (court-circuit franc entre le +VCC et GND possible)
Les montages A et C permettent au microcontrôleurs de connaître la position des boutons (B éventuellement, si l'on appuie toujours sur un et un seul des deux boutons poussoirs)

4 Programmation

1. Quelle est la sortie console lorsqu'on exécute le code C suivant : ☐

```
main()
{
   int x = 1;
   float y = x>>2;
   printf( "%f", y );
}
```

Le résultat est 0.00000
En effet, l'opérateur >> 2 effectue une division par 4 et seule la partie entier est conservée, affichée comme un réel. On ne peux non plus pas transformer x en float car dans ce cas, l'opérateur >> ne peut être utilisé (signalé par le compilateur)
On a ici une syntaxe C classique. Sur microcontrôleur en utilisant le framework Arduino, on écrit sur le Terminal en passant par la liaison série, donc on utilise la methode Serial.print() qui ne connaît pas les construction de formatage %f

```
int x = 1;
float y = x>>2;
Serial.println( y );
```

mais le résultat est identique : 0.00

2. Quel affichage sur la console lorsqu'on exécute le code C suivant (attention, il y un piège...) : ☐

```
#include <stdio.h>
main()
{
   int i = 0;
   switch (i)
   {
      case '0': printf("Hello");
                break;
      case '1': printf("World");
                break;
      default: printf("Hello World");
   }
}
```

Affichage "Hello World" car i vaut 0, alors que les différents case correspondent aux valeurs des codes ASCII du caractère 0 et du caractère 1

3. En reprenant le code de la question précédente, quel doit être la valeur numérique de i (donner la valeur en décimal) pour afficher seulement"Hello" sur la console ? ☐
(Il vous faudra sans doute chercher sur internet...)
Code ASCII du caractère 0 : 48 ou 0x30

4. Parmi les 3 codes ci-dessous, cochez celui ou ceux qui sont opérationnels : ☐ A ☐ B ☐ C
(pas de reponse à donner dans le quiz car le framework arduino n'a pas encore été abordé)

A
```
const int LED_pin = 13;

void setup() {
  pinMode( LED_pin, OUTPUT );
}

void loop() {
  digitalWrite( LED_pin, HIGH );
  delay(1000);
  digitalWrite( LED_pin, LOW );
  delay(1000);
}
```

B
```
int LED_pin = 13;

void setup() {
  pinMode( LED_pin, OUTPUT );
}

void loop() {
  digitalWrite( LED_pin, HIGH );
  delay(1000);
  digitalWrite( LED_pin, LOW );
  delay(1000);
}
```

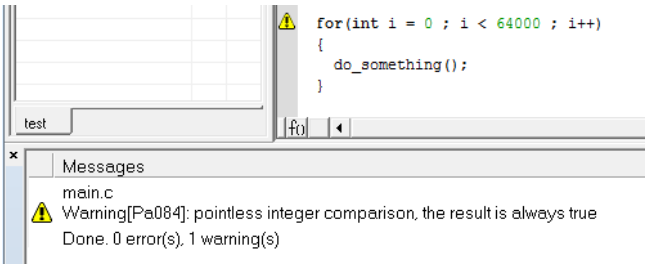
C
```
void setup() {
  int LED_pin = 13;
  pinMode( LED_pin, OUTPUT );
}

void loop() {
  digitalWrite( LED_pin, HIGH );
  delay(1000);
  digitalWrite( LED_pin, LOW );
  delay(1000);
}
```

A et B fonctionnement (la variable `LED_pin` n'est jamais réaffectée dans le programme, elle peut être déclarée comme `const`). Par contre, pour C, la portée de la variable `LED_pin` est limitée à la fonction setup() et sera inconnue dans loop().

5 Type des données

Les conventions ANSI C89 pour le langage C imposent que `char` et `byte` sont sur 8 bits, `short` et `word` sur 16 bits et `long` sur 32 bits. Mais le type de donnée `int` peut être sur 16bits ou sur 32bits, selon l'architecture du processeur cible.



1. Sur un microcontrôleur **16 bits**, pour la boucle `for` écrite ci-dessus, le compilateur me signale un avertissement. Pourquoi ?

2. Donnez, pour l'ESP32, la taille en octet des types de donnée suivants (recherche sur internet) :
short ☐ float ☐
int ☐ double ☐

1. *Dans la boucle, i est typé comme int (16 bits), et les données sont signés par défaut. Sur un processeur 16bits (ex. Texas Instrument MSP430) les valeurs possibles pour i vont donc de -32768 à +32767. C'est une boucle infinie car la condition d'arrêt n'est jamais rencontrée. Ce n'est pas une erreur évidente à détecter, à remarquer que le compilateur fait bien son travail en signalant cela par un Warning. Il faudrait écrire :*

```
for(unsigned int i = 0 ; i < 64000 ; i++)
{
  do_something();
}
```

2. *Pour l'ESP32 : short sur 16 bits, int sur 32 bits, float sur 32 bits et double sur 64 bits.*