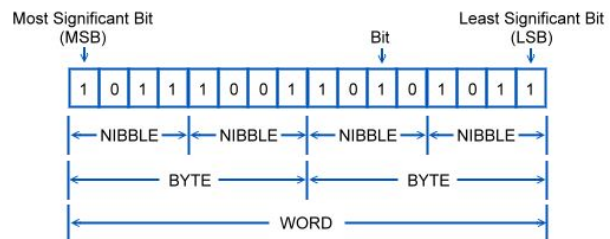


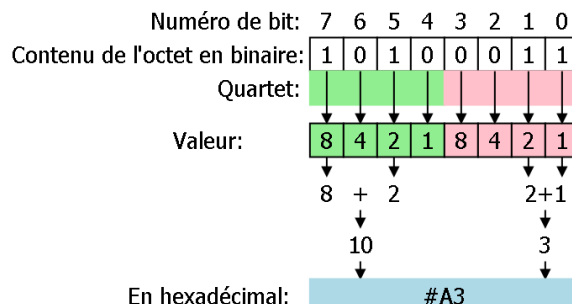
1 Rappels numération (binaire, hexadécimal)

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 donc être utilisée 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 hexadécimal : **FF**

· AA00 hexadécimal à convertir en binaire et en decimal : **1010 1010 0000 0000 AA00**

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

· 127 **0111 1111**

-128 **1000 0000**

· 1 **0000 0001**

-1 **1111 1111**

· 100 (64 + 32 + 4) **0110 0100**

-100 **1001 1100**

2 Opérateurs logiques

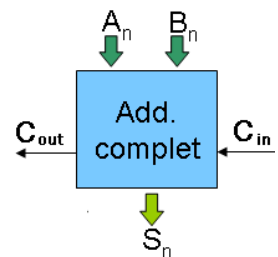
- Un master dans une université propose des UE d'enseignement "majeures" (correspondant à la discipline) nommées X, Y et Z, ainsi que des UE "mineures" nommées A et B. L'étudiant obtient son diplôme (D) si il obtient toutes les UE majeures de X à Z ou s'il obtient 2 UE majeures de son choix et les 2 UE mineures. La variable logique correspondant à une UE réussie prends la valeur 1.
 - Écrire D sous la forme d'une équation logique : $D = (X \text{ AND } Y \text{ AND } Z) \text{ OR } ((X \text{ AND } Y) \text{ OR } (Y \text{ AND } Z) \text{ OR } (X \text{ AND } Z)) \text{ AND } (A \text{ AND } B)$
 - 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 non obtenu");
```

- Effectuer une addition binaire correspond à résoudre un problème de logique combinatoire. En base 10, on apprend 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 a 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 peut prendre que la valeur 0 ou 1, c'est un problème de logique on peut donc déterminer la table de vérité d'un additionneur.

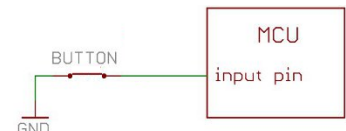
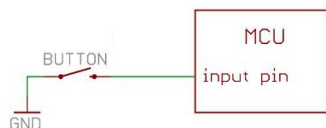
Soit A et B les deux nombres binaires à additionner (sur 1 seul bit, au rang n), C_{in} la retenue du rang inférieur, S le résultat (somme au rang n) et C_{out} la retenue à propager au rang supérieur :



- Combien d'équation(s) faut-il déterminer ? 2
- Combien de variables logiques interviennent dans chaque équation ? 3
- Combien de combinaisons possibles pour chaque équation (donc de lignes dans la table de vérité) ? 8
- Établir les tables de vérité (pas de réponse à donner sur Moodle, la correction sera faite en TD)

3 Entrées / Sorties

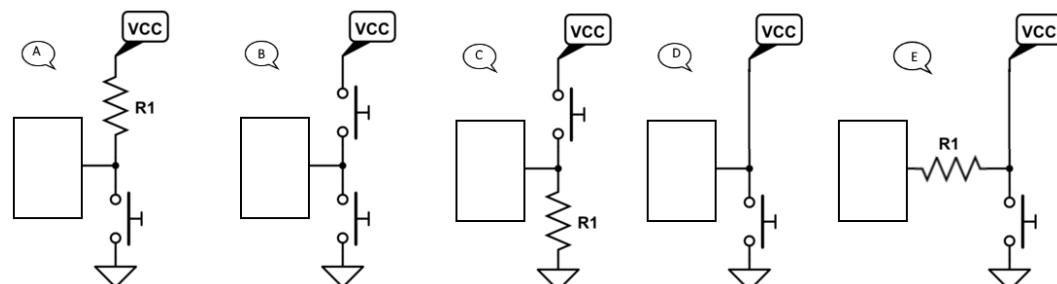
1.



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)



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

4 Programmation

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

0

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

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

Hello World

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

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 ?

48

(Il vous faudra sans doute chercher sur internet...)

4. Parmi les 3 codes ci-dessous, cochez celui ou ceux qui sont opérationnels : ☒ A ☒ B ☐ C
(pas de réponse 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);
}
```

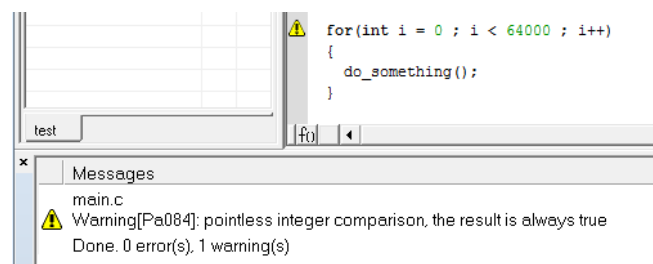
C

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

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

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 ?

Le maximum pour un int (16 bits) est 32768 - 1 donc 32767. Donc 32767 < 64000. Donc le int(16 bits) sera toujours inférieur à 64000. Donc toujours vrai.

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

short 16 bits (non trouvé dans la doc)
int 16 bits

float 4 bits
double 64 bits