# Définition

Un ALU est une unité arithmétique et logique, elle permet entre autres de faire des opérations sur des entrées données. Dans le cas de ce laboratoire, nous réalisons un ALU qui gère 2 entrées de 4 bits et une entrée de sélection (opcode).

L’opcode est une entrée de 6 bits qui nous permet de sélectionner l’opération voulue. Il est séparé en plusieurs partie et chaque partie opère sur une opération spécifique. Par exemple les deux premiers bits à gauche de l’opcode permettent de sélectionner quelle opération des 4 blocs que nous avons construits sera exécutée.

Une image contenant table

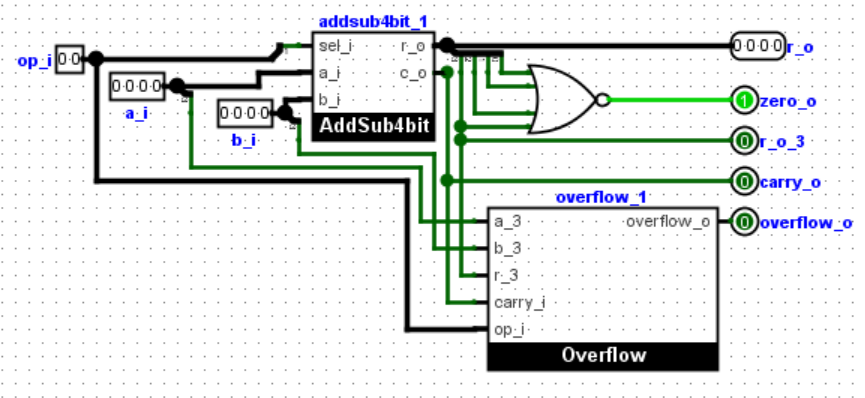
Description générée automatiquement

Comme on peut le voir ci-dessus notre ALU est composé de 4 blocs principaux. On va détailler chacun de ces blocs ci-dessous.

# Blocs

## Add/Sub

Ce premier bloc va nous permettre d’additionner ou de soustraire les deux valeurs d’entrées. On peut effectuer des opérations en signé ou non-signé. On réutilise le bloc AddSub4Bit de notre laboratoire sur le checksum.



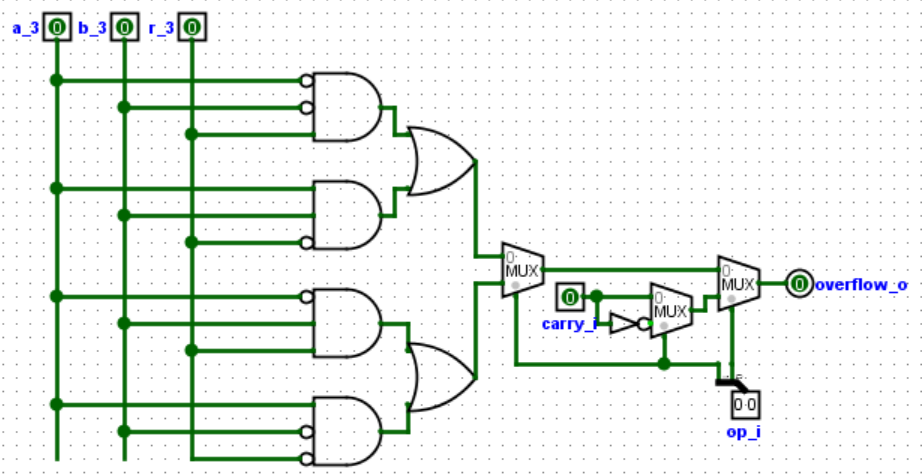
Pour détailler les opérations :

* **Addition signée (00) :** On additionne bit à bit les deux entrées. La sortie carry s’allume s’il y a une retenue en fin de calcul et la sortie overflow s’allume si le résultat n’est pas cohérent. C’est-à-dire si l’addition de deux nombres de même signe donne un nombre de signe opposé.
* **Soustraction signée (01) :** Dans le cas d’une soustraction, on fait le complément à 2 du second nombre grâce au bit de l’opcode qui est à 1. Puis on additionne les deux nombres. Encore une fois s’il y’a une retenue à la fin le carry sera allumé. Pour l’overflow, c’est pareil que précédemment si le résultat attendu n’a pas le bon signe c’est activé.
* **Addition non-signée (10) :** On additionne simplement bit à bit les deux entrées. En cas de dépassement de capacité (nombre > 15), la sorite **carry** et **overflow** s’allument. Car lors de l’addition non-signée, l’overflow c’est le carry.
* **Soustraction non-signée (11) :** Pareil que pour l’addition signé, on fait le complément à 2. Ici le carry s’allume avec les mêmes conditions que précédemment, mais l’overflow s’active lorsque le carry est désactivé. Car l’overflow c’est l’inverse du carry dans la soustraction non-signée.

## Overflow

Justement l’overflow est un bloc à part dans le circuit, il est séparé en deux partie. La première étant les opérations non-signées, où comme expliquer ci-dessus l’overflow s’active lorsque le carry est activé pour l’addition et lorsque le carry est désactivé pour la soustraction.

Pour les nombres signés, il faut suivre le tableau dans la consigne et le traduire sous format logique à l’aide de portes ET et d’inverseurs. Les portes OU permettant de sélectionner si l’on est en mode addition ou soustraction.



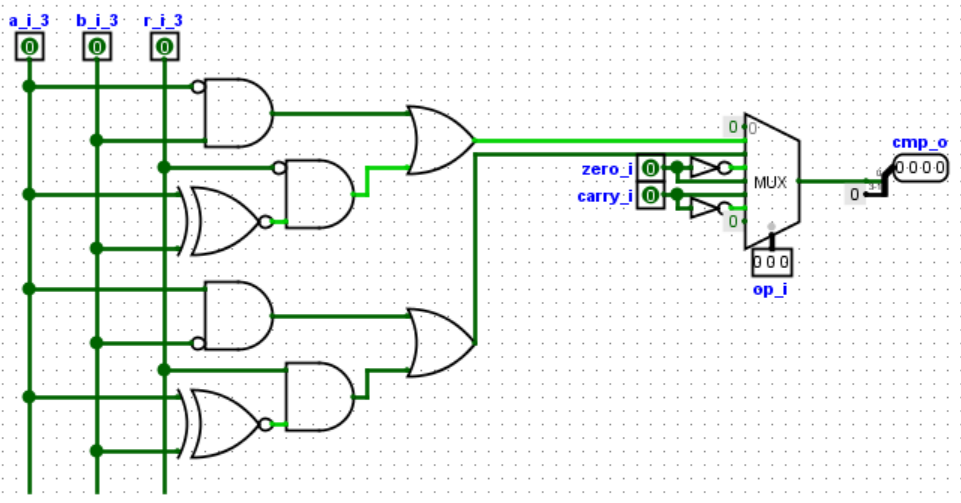
## Comparateur

Ce bloc est plutôt étant donné que tous détails des opérations à effectuer sont données sous forme d’expression logique. Il suffit donc de réaliser le circuit de portes logiques pour que cela fonctionne et évidemment d’intégrer les bonnes entrées (zéro et carry) du tableau ci-dessous.

Une image contenant table

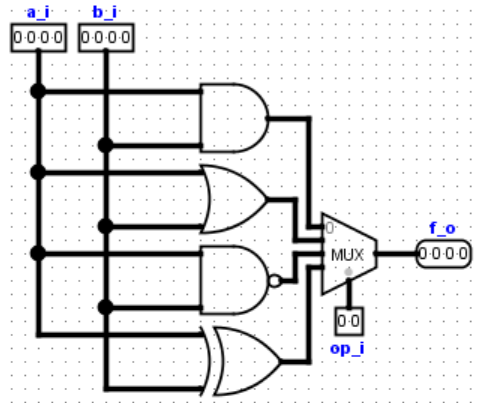
Description générée automatiquement

Le circuit correspondant à ce tableau étant le circuit représenté ci-dessous.



## Logique

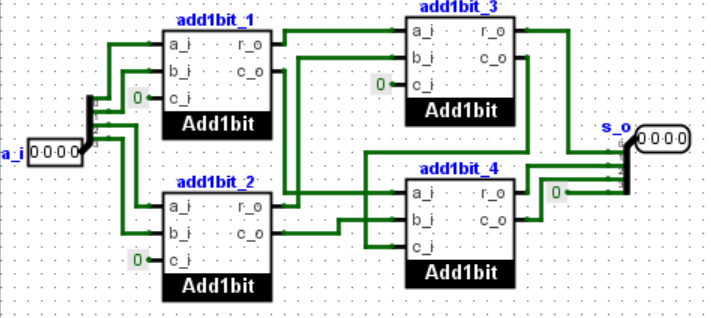
Cette unité est simplement un ensemble de portes logiques reliées au MUX de sélection. Elle réalise tout simplement les opérations logiques indiquées entre les deux nombres.



## Custom

Permet de compter le nombre de 1 dans un nombre binaire. Il y a pour ce bloc plusieurs manières de le réaliser. Que ce soit avec des additionneurs et des portes logiques, en faisant un additionneur 3 bits ou comme la méthode présentée ici, c’est-à-dire 2 additionneurs 1 bits et 1 additionneur 2 bits.

On n’aura jamais besoin de plus qu’une addition de 3 bits, étant donné que le nombre maximum de 1 est 4 et que 4 se représente sur 3 bits en binaire.



## Erreur

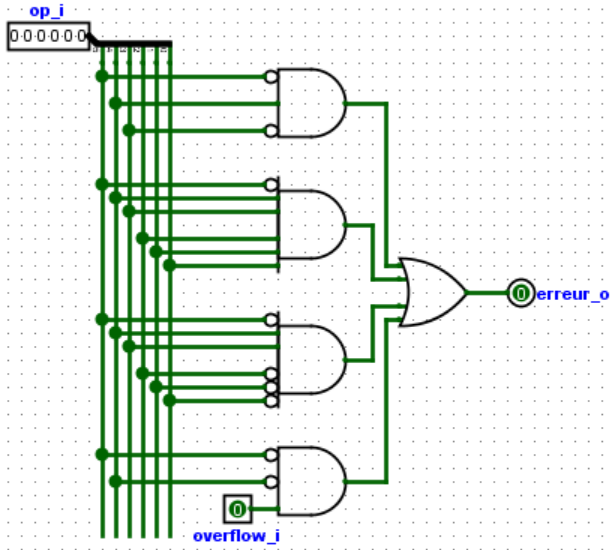
Ce dernier bloc permet de de détecter s’il y’a une erreur dans notre circuit. C’est-à-dire soit un opcode en entrée qui n’est pas pris en charge, soit un résultat incohérent.

Pour le premier cas, en regardant le tableau d’opcodes on remarque qu’il existe 10 opcodes qui ne sont pas pris en charge.

* 011000 (3ème porte ET)
* 011111 (2ème porte ET)
* 010XXX (1ère porte ET)

Pour le second cas de l’opération incohérente, le seul moment où l’on peut avoir un résultat incohérent est dans le bloc Add/Sub, c’est-à-dire lorsque les deux bits à gauche de l’opcode sont égaux à 00 et que l’overflow est à 1 (overflow qui indique un problème dans le résultat). (4ème porte ET).

Au format logique on peut représenter cela avec le circuit suivant.



## Assemblage

Voici la représentation finale de tous les circuits.

