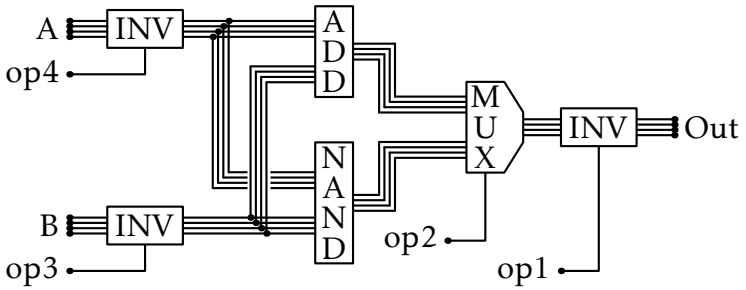
Unité Arithmétique et Logique

UAL

L'**Unité Arithmétique et Logique** (ou UAL) est la partie du processeur qui réalise les calculs et les opérations logiques élémentaires. Selon le processeur, cette unité peut réaliser des opérations plus ou moins complexes, sur des nombres plus ou moins longs.
Nous allons considérer un UAL simpliste pouvant travailler sur des nombres de 4 bits. Dans la suite, nous noterons $\overline{A}$ l'opposé de A bit à bit. L'architecture globale est la suivante :

- INV : renvoie A si op = 0 et $\overline{A}$ sinon.
- ADD : renvoie la somme des deux nombres.
- NAND : renvoie le résultat de l'opération logique.
- MUX : renvoie A si op = 0 et B sinon.



Les bits op4 à op1 correspondent au **code opération** (*opcode*). Ce code sert a déterminer le type d'opération effectuée par l'unité. Par exemple, avec le code 0000, l'unité réalise A + B, alors qu'avec 0010, elle réalise A nand B.
La plupart des UAL ont des bits supplémentaires en sortie permettant de savoir s'il reste une retenue à la fin de la somme (overflow), si le résultat est négatif (les entiers sont représentés en complément à 2), égal à zéro…

Exemples d'utilisation

Le tableau ci-contre présente quelques exemples d'utilisation. On remarque qu'il est possible de faire une soustraction. En effet, par définition du complément à 2, on a $-A = \overline{A} + 1$, donc $\overline{A} = -A - 1$.
Pour la 4e ligne, l'opération effectuée correspond à $\overline{\overline{A} + B}$. On a donc :

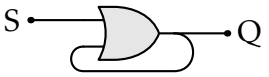
$$\overline{\overline{A} + B} = \overline{-A - 1 + B} = -(-A+B-1)-1 = A-B$$

opcode	A	B	Out	commentaire
0000	0011	0101	1000	A + B
0000	0001	1111	0000	overflow
0101	0011	0101	0010	B – A
1001	0011	0101	1110	A – B
0101	0011	0000	1101	–A
0001	1100	0000	0011	non A
0010	0011	0101	1110	A nand B
0011	0011	0101	0001	A et B
1110	0011	0101	0111	A ou B

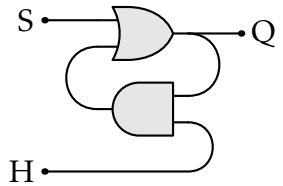
Exercice : Déterminer le résultat des opérations suivantes :

opcode	A	B	Out
0000	1011	0110	
0101	1011	0110	
1001	1011	0110	
0010	1011	0110	
0011	1011	0110	
1110	1011	0110	
0101	1011	0000	
0001	1011	0000	

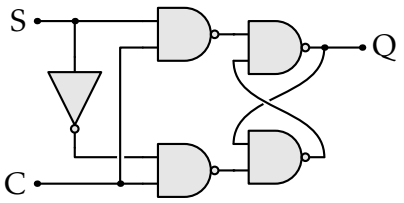
Mais où on va ranger tout ça ?



Une fois que les calculs sont fait, il faut pouvoir les stocker afin de les réutiliser ensuite. Pour cela on utilise des **circuits séquentiels**.
En effet, en branchant la sortie du circuit à l'entrée d'une des portes, on peut former des boucles qui conservent une valeur. C'est le cas du circuit ci-dessus. Tant que S vaut 0, alors Q vaut aussi 0. Par contre, une fois que S passe à 1, alors Q aussi. Après cela, peut importe la valeur de S, Q reste à 1. On dit que l'on a un **verrou** puisque la valeur de Q est verrouillée.



Le problème avec ce circuit, c'est qu'il n'est pas possible de déverrouiller Q. On peut alors rajouter une autre entrée, H, qui permet de mettre la sortie à 0. Tant que H est à 1, le circuit se comporte comme le précédent. Lorsque H passe à 0, la sortie est remise à 0. En améliorant ce circuit, on obtient un verrou SR avec une entrée S qui sauvegarde la valeur et R qui la réinitialise.



Les verrous introduisent une notion de temporalité dans les circuits. Comme dans un programme avec l'affectation d'une valeur à une variable déjà définie, il y a une valeur avant et après l'affectation. On leur associe souvent une horloge qui est une entrée qui change de valeur à intervalles réguliers.
Dès que l'horloge passe à 1, la sortie est modifiée en fonction des entrées. Cette valeur sera verrouillée pendant un cycle d'horloge. On parle alors de **bascule**, comme la bascule D présentée ci-dessus.

Tenir les registres à jour

Ces bascules permettent à leur tour de faire des compteurs qui s'incrémentent à chaque cycle d'horloge ou simplement des **registres** qui peuvent sauvegarder des données. La **mémoire vive**, ou RAM (Random Access Memory), d'un ordinateur est composée de milliards de registres. Chacun de ces registres peut stocker la valeur d'une variable pour des types simples, comme des booléens, des entiers ou des nombres à virgule. Pour des objets plus complexes, comme des textes, il peut être nécessaire d'utiliser plusieurs registres. Chaque registre est identifié par une adresse.
Il y a deux opérations possibles sur la mémoire : la lecture ou l'écriture. Pour une lecture, il faut juste indiquer l'adresse et pour une écriture il faut aussi donner la valeur à inscrire. Il est possible de passer de n'importe quelle adresse à n'importe quelle autre, sans passer par toutes celles au milieu. Ce qui explique le nom RAM, ou "mémoire à accès arbitraire".
Par contre, cette mémoire doit être alimentée électriquement pour perdurer. C'est une mémoire volatile, contrairement aux supports de stockages comme les disques durs ou les clefs usb où les données sont stockées physiquement.

Tout le monde dans le bus

Puisqu'un registre sauvegarde plusieurs bits, il faut des connexions permettant de transporter plusieurs bits en même temps. On appelle cela un **bus**. Il y a 3 grands types de bus :

- Le bus d'adresse, par exemple pour envoyer une adresse mémoire.
- Le bus de données, pour faire circuler les données.
- Le bus de contrôle pour spécifier le type d'action, par exemple écrire ou lire une donnée en mémoire.

Les différents composants d'un ordinateur communiquent entre eux grâce aux bus.