15. Réalisation des composants

Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d'Informatique

Contenu du cours

Partie I: Introduction

Partie II: Codage de l'information

Partie III: Circuits logiques

Partie IV: Architecture des ordinateurs

1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. Codage des nombres entiers naturels
5. Codage des nombres entiers relatifs
6. Codage des nombres réels
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
11. Circuits combinatoires importants
12. Principes de logique séquentielle
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. **Réalisation des composants**
16. Code machine et langage assembleur
17. Architecture d'un processeur
18. Performance et micro-architecture
19. Du processeur au système

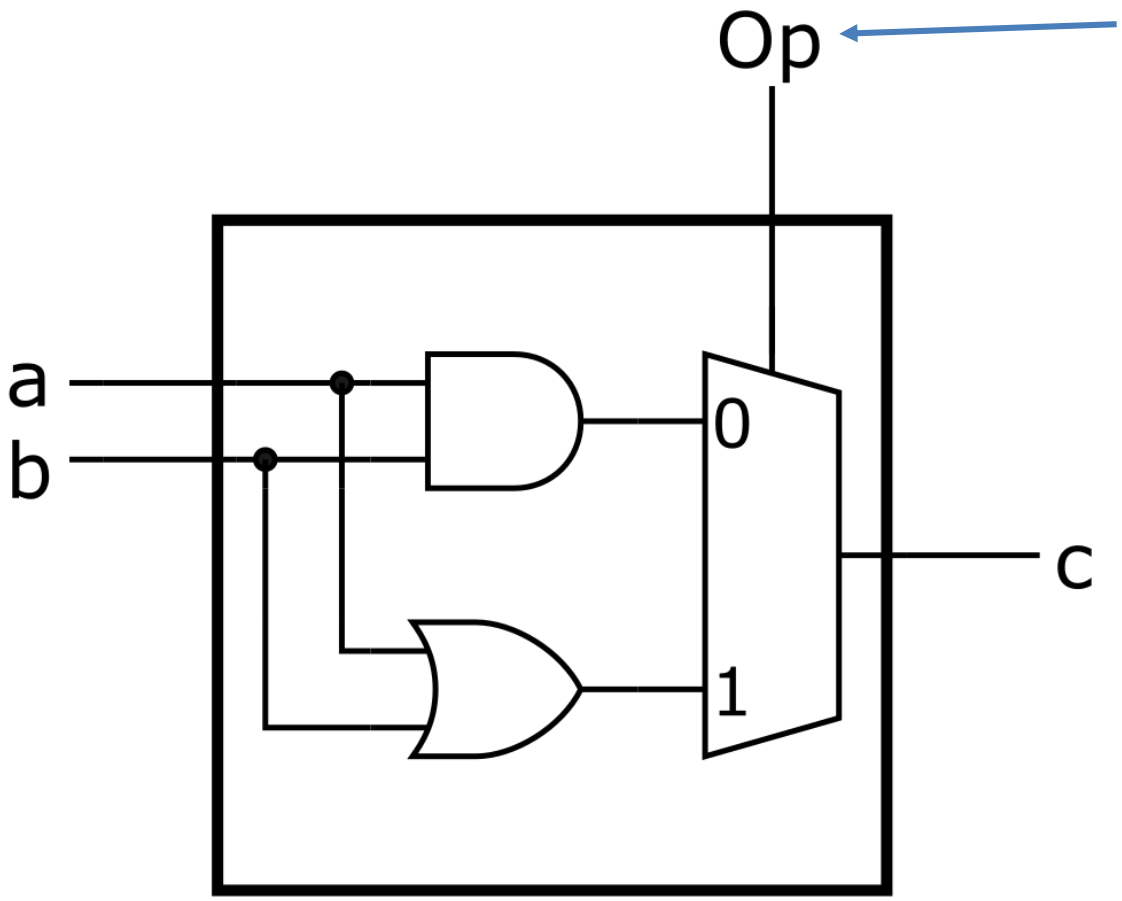
L’Unité Arithmétique et Logique

Jonas Lätt

Responsabilités de l’ALU

- L'**unité arithmétique et logique** (en anglais arithmetic-logic unit, **ALU**), est l'organe de l'ordinateur chargé d'effectuer les calculs.
- Une ALU élémentaire agit sur des nombres entiers, et effectue des opérations communes, que l'on sépare en quatre groupes :
  1. **Les opérations arithmétiques**: addition, soustraction, changement de signe, …
  2. **Les opérations logiques**: Le AND, OR, NOT, XOR, etc.
  3. **Les comparaisons**: test d'égalité, supérieur, inférieur, et leurs équivalents «supérieur ou égal», etc.
  4. **Les décalages et rotations** de bits (bit-shifts), comme dans le barrel shifter.
- Nous allons construire une ALU simplifiée qui comprend deux opérations logiques (AND et OR), deux opérations arithmétiques (addition et soustraction), et une comparaison (test d’égalité).

Comment effectuer différentes opérations en un circuit?

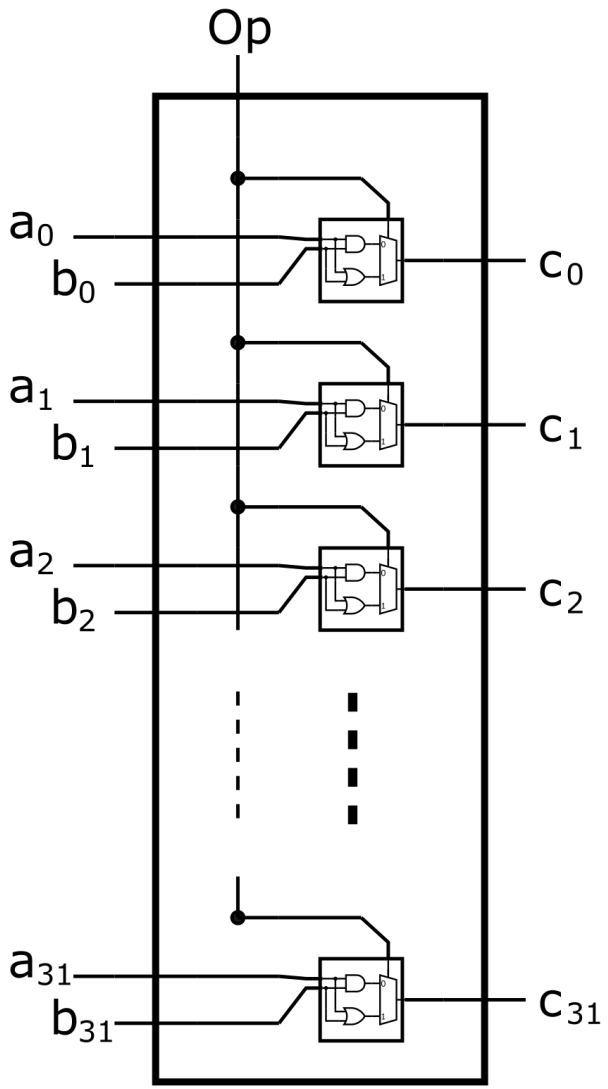


La ligne de contrôle sélectionne l'opération à exécuter.

Réponse: en utilisant un multiplexeur!

- Entrée: deux bits a et b.
- Sortie: résultat de AND ou OR.
- Le AND et OR travaillent en parallèle. Puis, le multiplexeur choisit le résultat désiré.

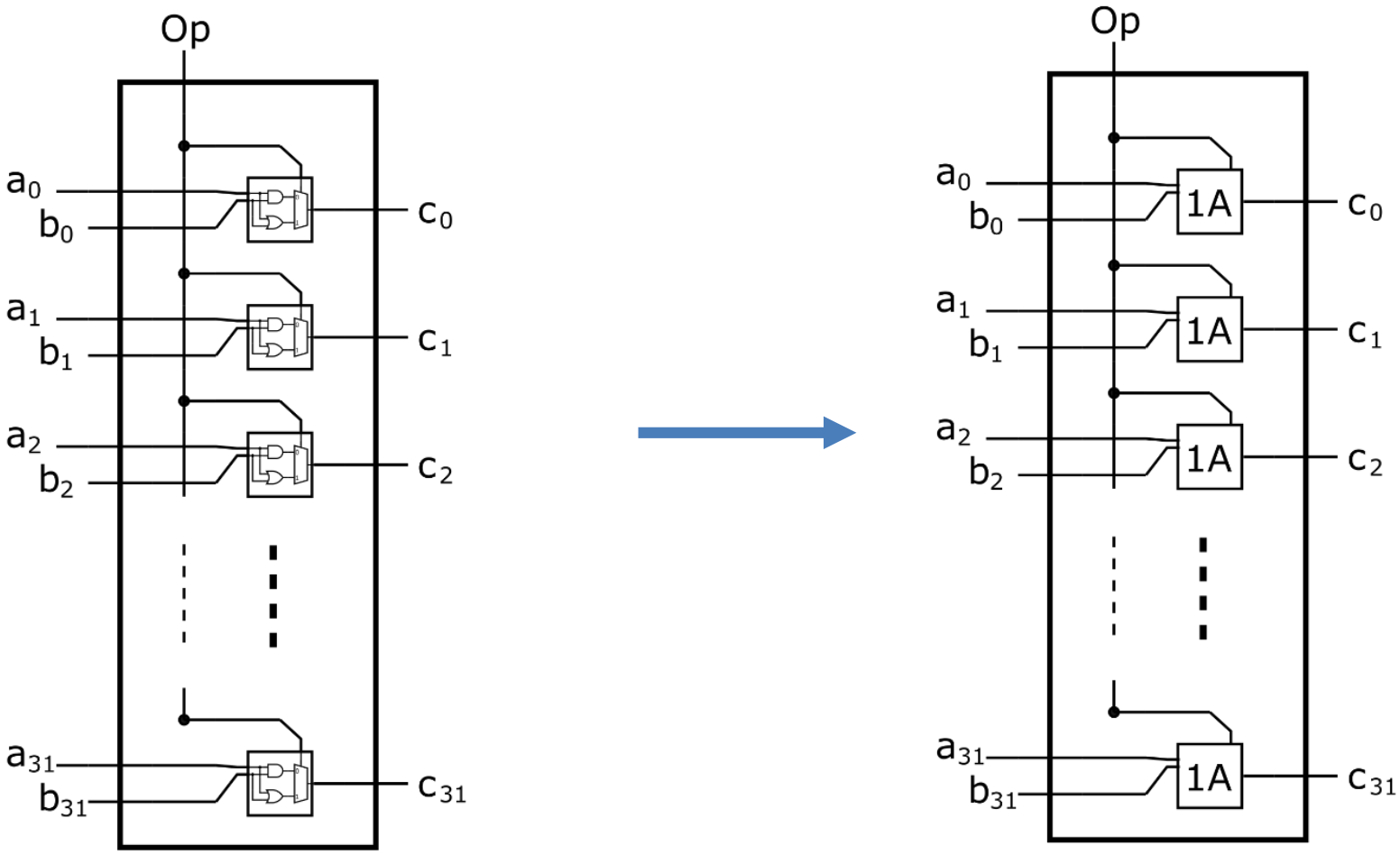
Notre première ALU: AND et OR



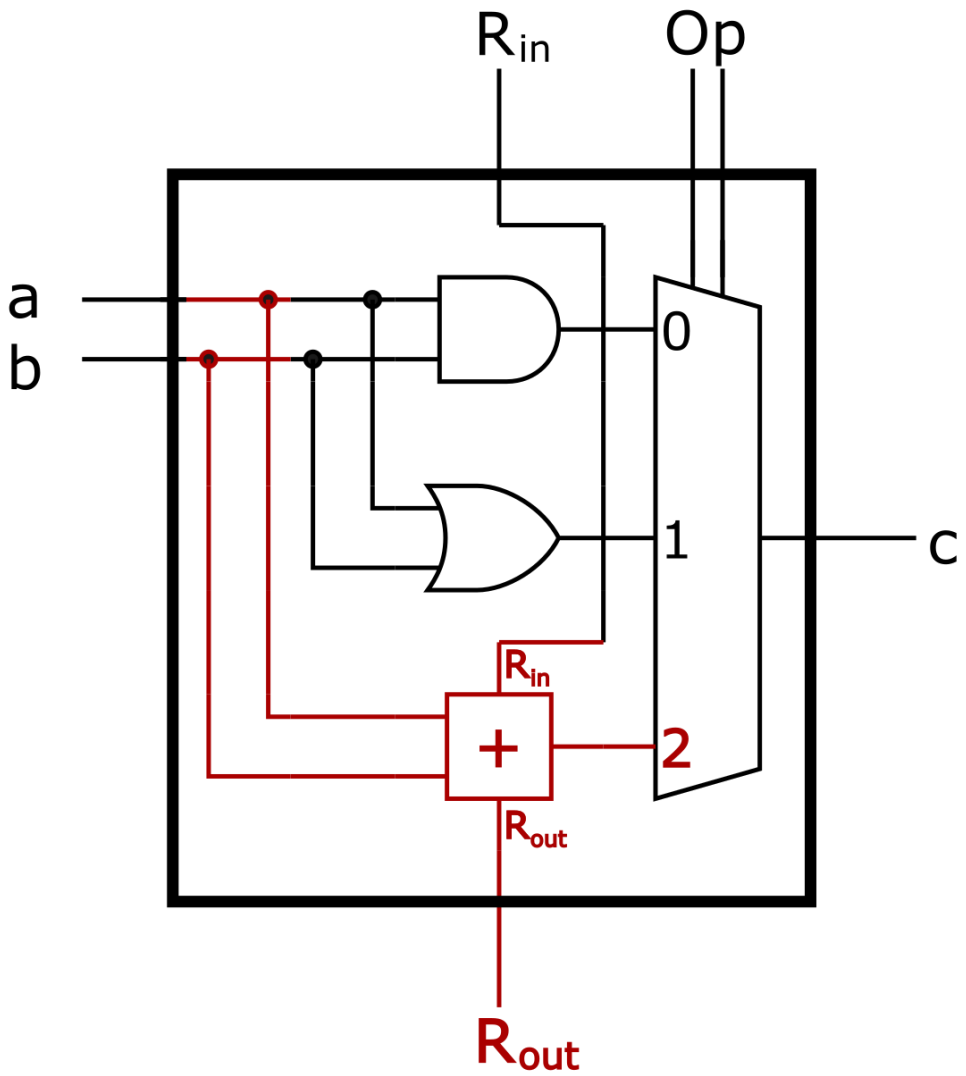
- Entrée: Des mots à 4 octets (32 bits), A et B
- Sortie: Un mot à 32 bits, C.
- Opérations: AND et OR, sélectionnés par la ligne de contrôle à 1 bit **Op**.
- Réalisation: on copie le circuit 1-bit 32 fois.
- Pour compléter cette ALU, il suffit de
  - Augmenter la capacité des multiplexeurs et rajouter des lignes de contrôle.
  - Rajouter d'autres opérations dans l'ALU à 1-bit.

Principes d'une ALU

Représentation symbolique de l'ALU à 1-bit:



Extension de l'ALU 1-bit: Somme

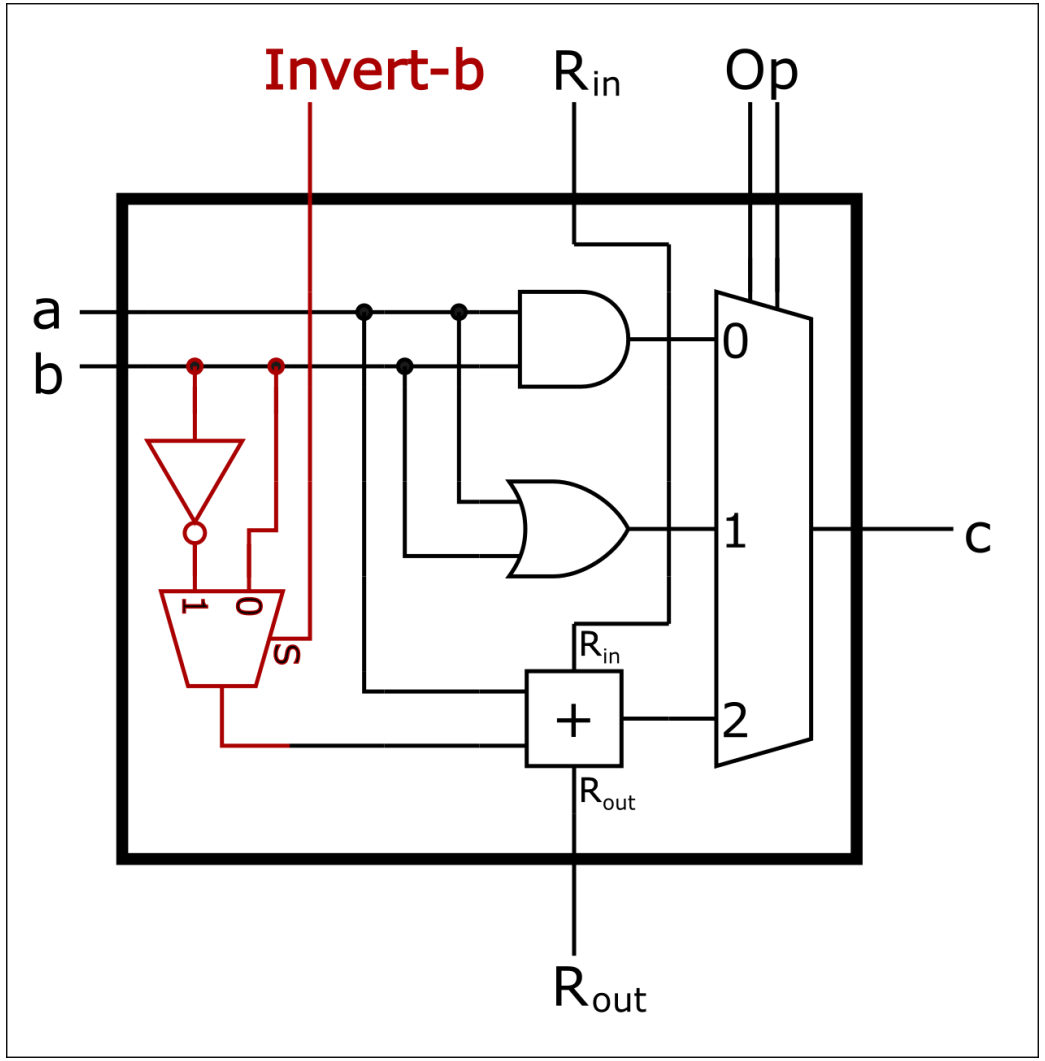


Rajoutons l'opération somme:

- On rajoute une ligne au multiplexeur pour passer à potentiellement 4 opérations (on n'en utilisera que 3).
- On rajoute un additionneur complet, qui s'exécute en parallèle avec le OR et le AND.

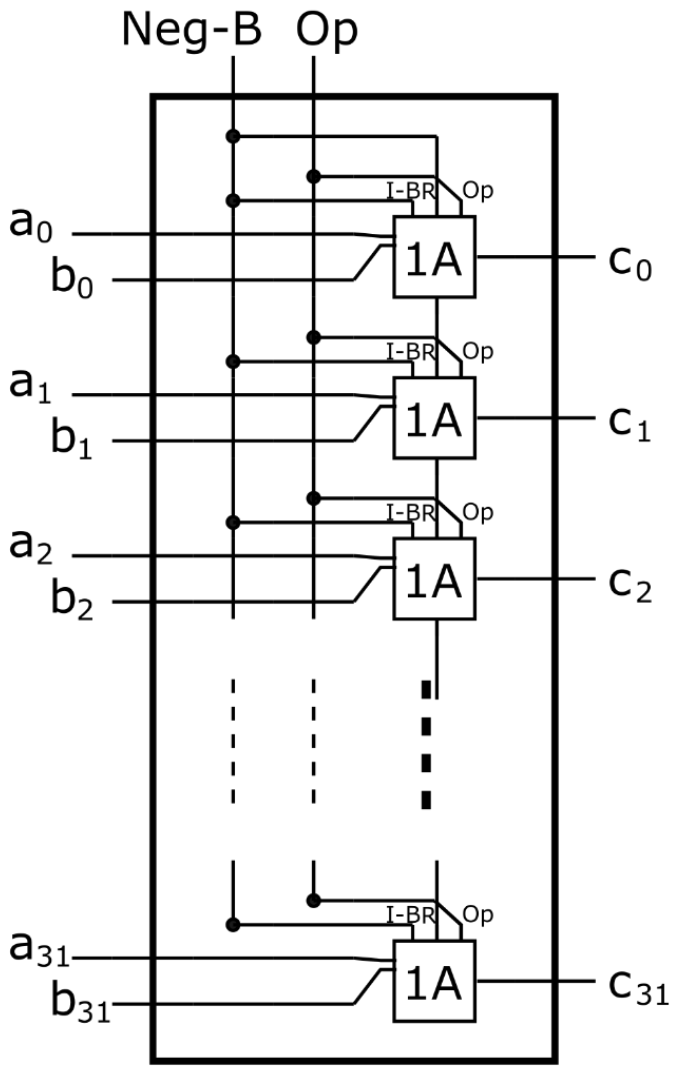
Ce circuit a maintenant, en plus, une retenue en entrée et en sortie.

Extension de l'ALU 1-bit: Soustraction



- Pour la soustraction, B est remplacé par –B.
- A cet effet, la ligne de contrôle **Invert-b** invertit le bit d'entrée b.

ALU améliorée: AND, OR, Somme, Soustraction

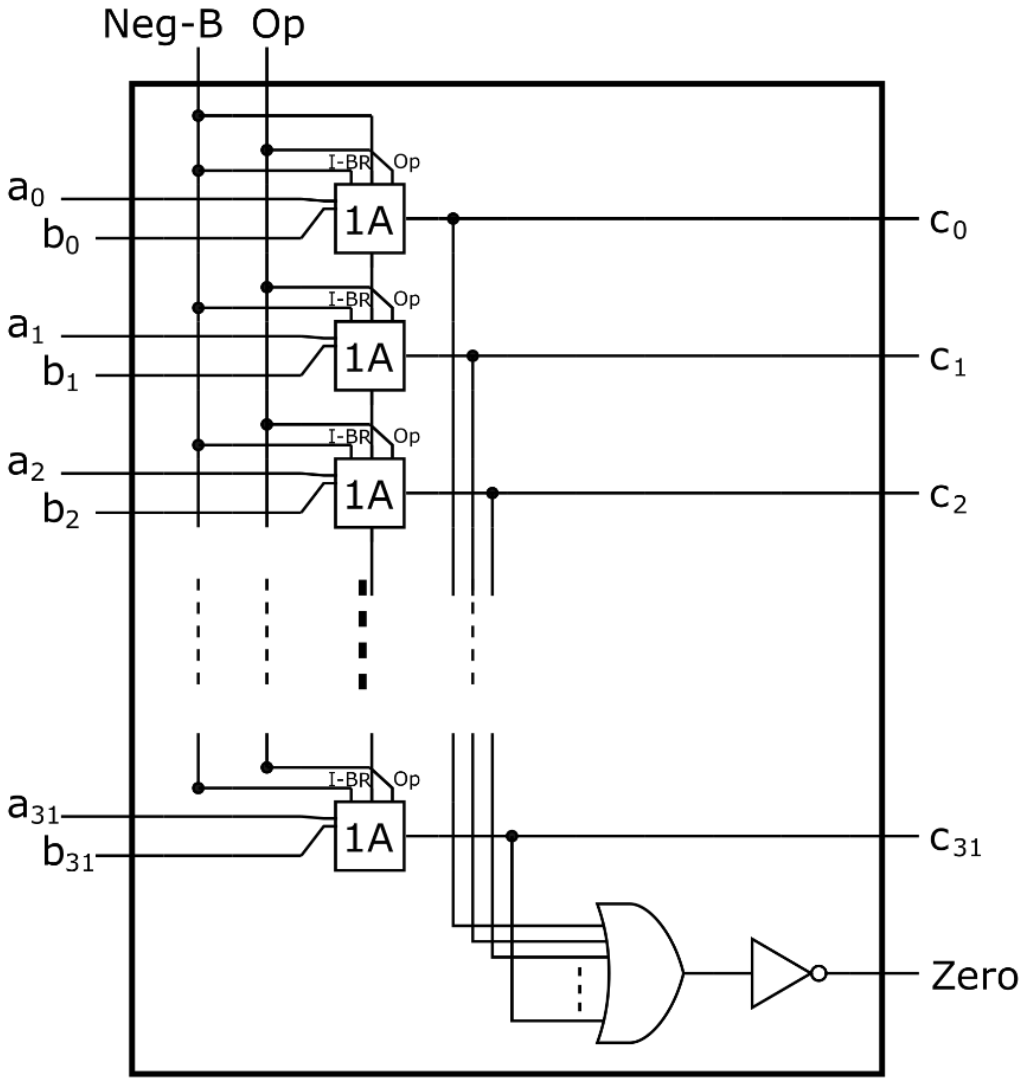


L'ALU améliorée possède

- Deux lignes de contrôle **Op** pour le choix de l'opération (AND, OR, SOMME)
- Une ligne de contrôle **Neg-B** pour
  - Invertir les bits de B
  - Fournir une retenue en entrée à la première ALU à 1-bit.

Les retenues d'entrée/sortie des 1A sont connectées pour effectuer une addition.

ALU améliorée:
AND, OR, Somme, Soustraction, test d'égalité



Test d'égalité «A=B» ↔ «A-B=0»

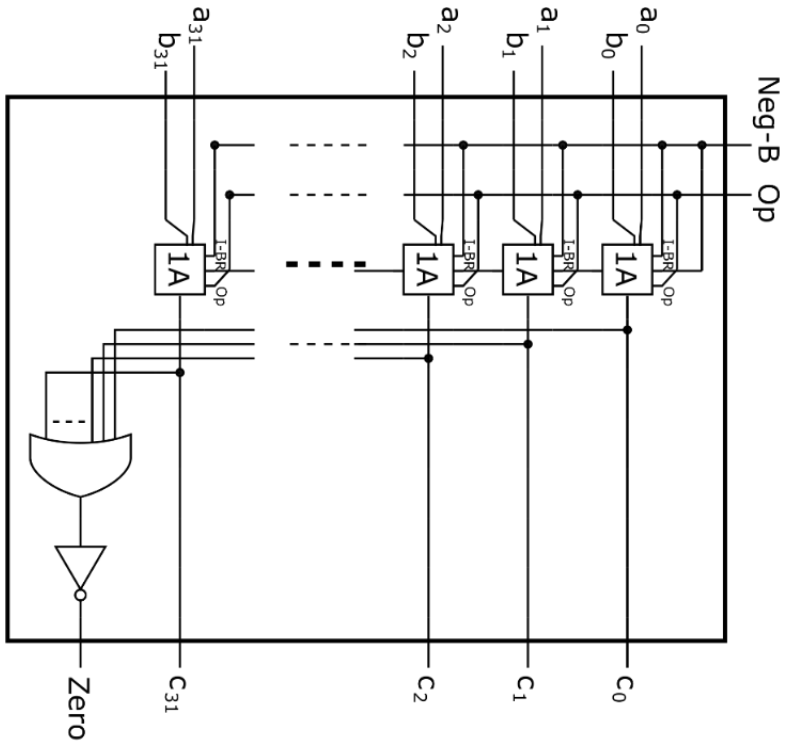
- Un bit de sortie **Zero** est rajouté.
- A chaque utilisation de l'ALU, ce bit indique si le résultat est zéro.

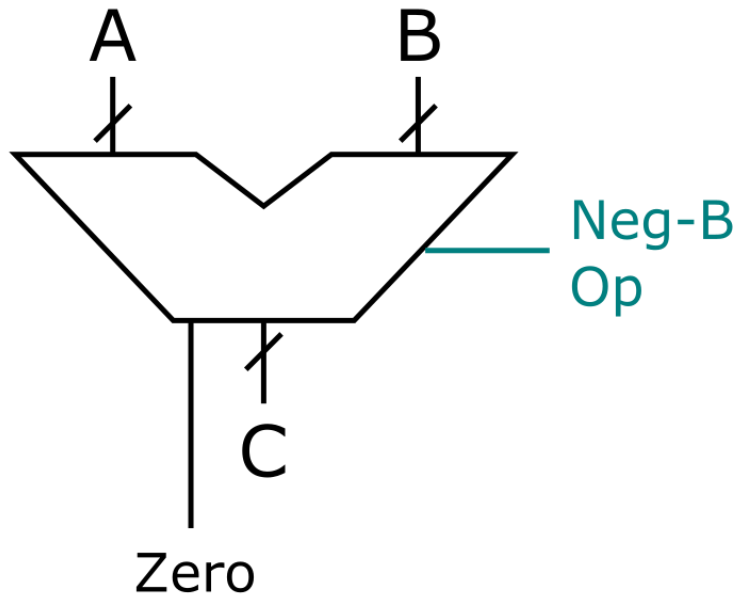
ALU: Résumé

Pas montrés ici (mais présent dans la plupart des ALU):

- Opérations de bit-shift (décalage de bits).
- Autres opérations logiques et de comparaison.

Symbole de diagramme:



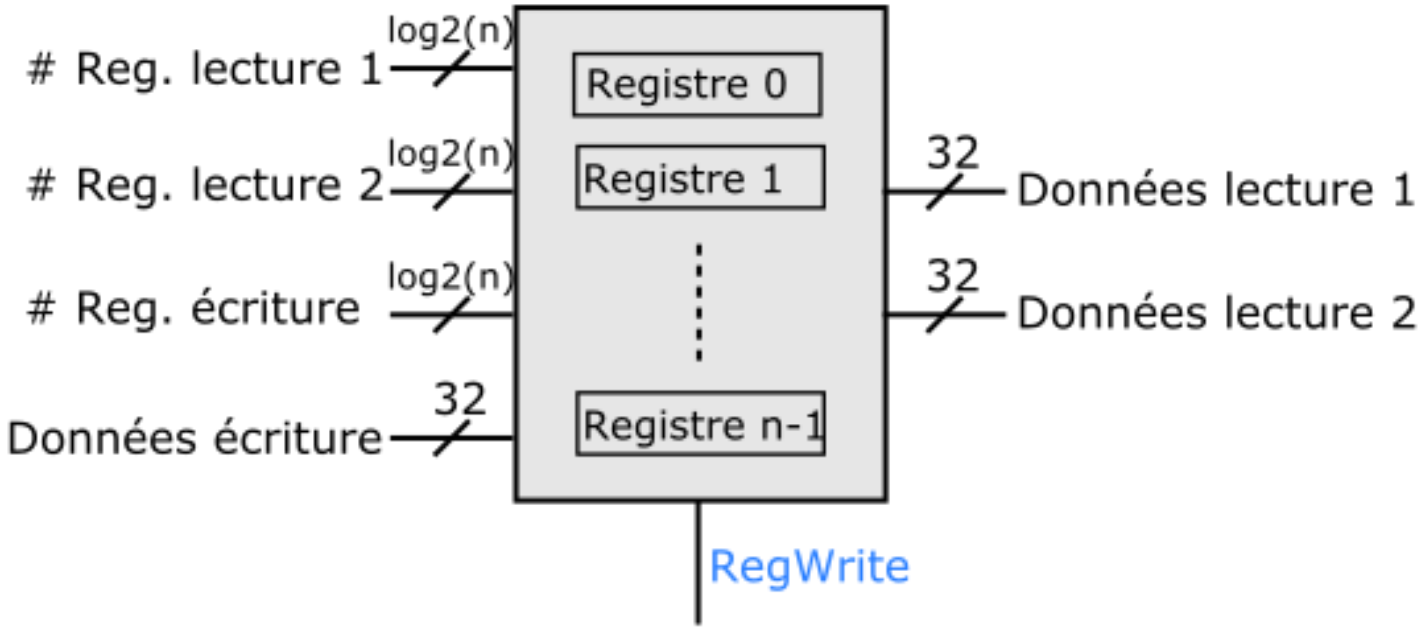


Le Banc de Registres

Jonas Lätt

Les bancs de registres

- Les registres du processeur sont arrangés dans des bancs de registres.
- Dans l'architecture RISC-V, le banc de registres permet d'écrire un registre, ou de lire deux registres simultanément.
- Les sorties de lecture reflètent à tout moment l'état des registres dont le numéro est indiqué en entrée (la sortie «Donnée lecture 1» donne l'état du registre dont le numéro est précisé dans l'entrée «# Reg. Lecture 1»).
- Un registre peut être écrit (le registre de numéro «# Reg. écriture» adopte alors la valeur «Données écriture») si le signal de contrôle «RegWrite» prend la valeur 1.

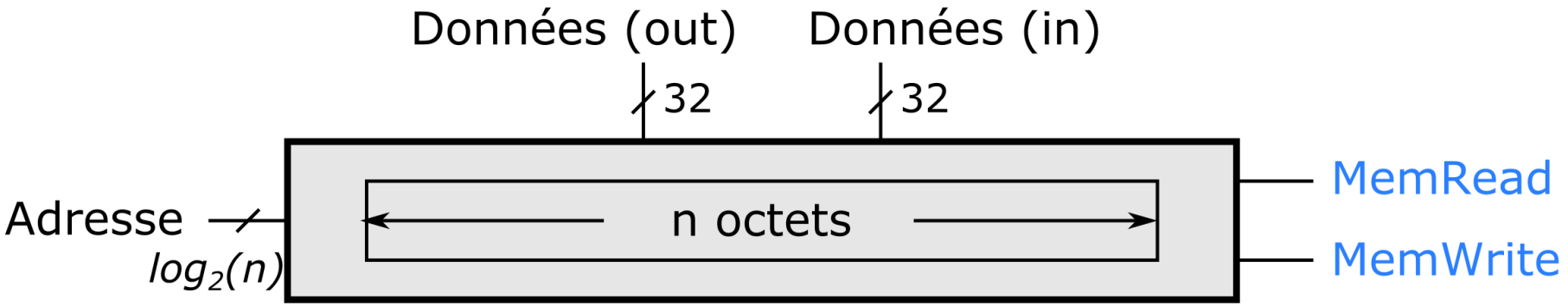


La mémoire centrale

Jonas Lätt

La mémoire centrale

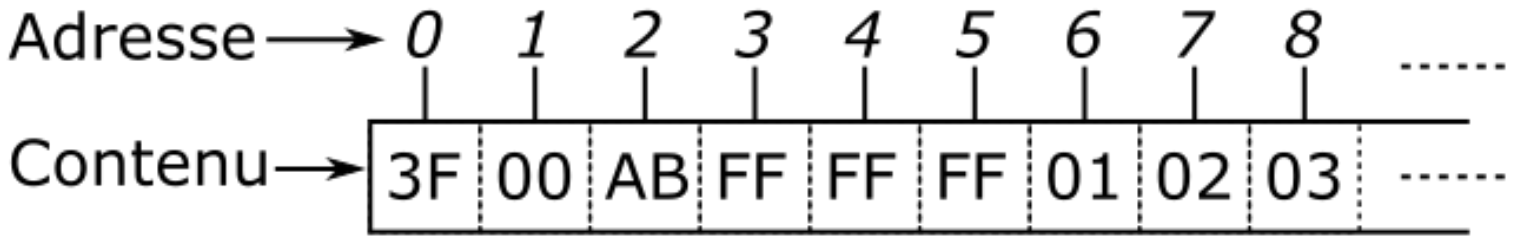
- La mémoire centrale accueille n octets, qui peuvent être lus / écrits mot par mot.
- Dans l'architecture RISC-V 64-bit, on peut lire ou écrire des blocs de 8 octets à la fois.
- Les signaux de contrôle «MemRead» / «MemWrite» enclenchent des lectures / écritures.



Random Access Memory (RAM): On peut lire les données dans n'importe quel ordre

La mémoire centrale

- Chaque octet possède une **adresse en mémoire**.
- Ces adresses sont des valeurs entières, consécutives de 0 à n-1.
- Voici une configuration possible des 9 premiers octets de la mémoire:



Exercice instantané

Alignement des octets en mémoire

Imaginons-nous qu'un ordinateur représente le nombre entier 256 dans un mot à 16 bits, à l'adresse 1000 dans sa mémoire centrale.

Laquelle des deux affirmations est-elle juste?

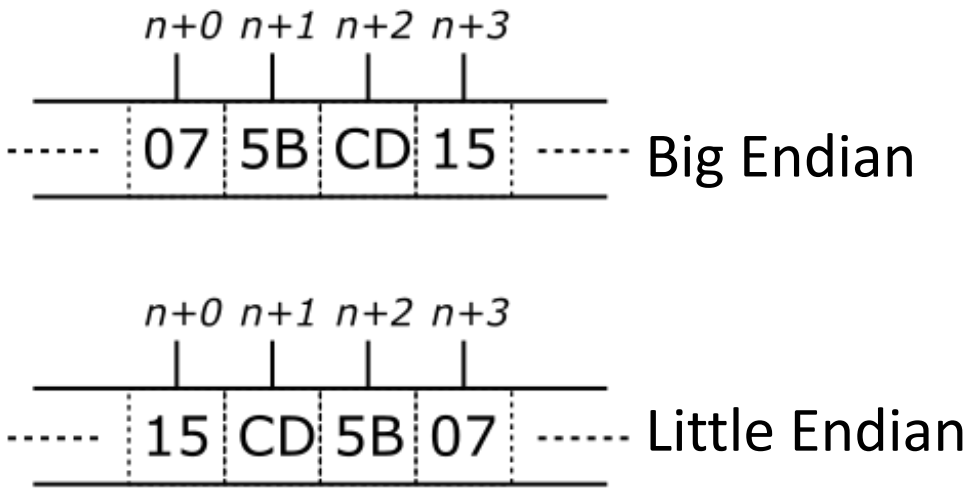
1. L'octet à l'adresse 1000 a la configuration binaire 0x00, et l'octet à l'adresse 1001 la configuration 0x01.
2. L'octet à l'adresse 1000 a la configuration binaire 0x01, et l'octet à l'adresse 1001 la configuration 0x00.

«Little-Endian» vs. «Big-Endian»

- Les valeurs entières sont fréquemment codées sur plus qu'un octet. Dans ce cas, la question se pose, dans quel ordre les octets de la valeur sont placés en mémoire.
- La convention «Little Endian» place l'octet de poids le plus faible en premier.
- La convention «Big Endian» place les octets de gauche à droite, dans le sens de la lecture.

Exemple:

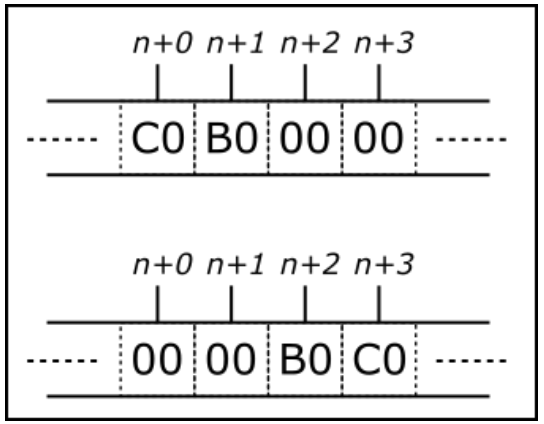
$123456789_{(décimal)}$ =
$07\ 5B\ CD\ 15_{(hexadécimal)}$



«Little-Endian» vs. «Big-Endian»: virgule flottante

- Pour les nombre réels à virgule flottante, les deux mêmes choix sont possibles.
- Exemple: représentation de la valeur -5.5 en simple précision IEEE 754:

- Dans ce cas, les quatre octets
  11000000 10110000 00000000 000000
  sont représentés par la séquence hexadécimale
  C0 B0 00 00



Big Endian

Little Endian

«Little-Endian» vs. «Big-Endian»: le choix des architectures existantes

La convention little-endian

- La plupart des architectures modernes adoptent la convention little-endian. En particulier, **les processeurs x86** (les processeurs Intel et AMD) fonctionnent en little-endian.
- Le p**rocesseur RISC-V** utilisé pour illustration dans les prochains chapitres est little-endian.
- **Les processeurs ARM** (utilisés sur une grande majorité des plateformes mobiles du type smartphone/tablette) sont bi-endian: ils peuvent fonctionner en mode little-endian ou big-endian. Ceci dit, sur iOS et Android ils sont utilisés en mode little-endian.

La convention big-endian

- Beaucoup de machines de grande valeur historique, et occasionnellement encore utilisées de nos jours (par exemple **PowerPC, Amiga, SPARC**) utilisent la convention big-endian.
- Le protocole IP (protocole standard de **communication réseau**) utilise une convention big-endian pour transférer les données sur le réseau.