Principes de fonctionnement des machines binaires

2019/2020

Pierluigi Crescenzi
Université de Paris, IRIF

Université de Paris

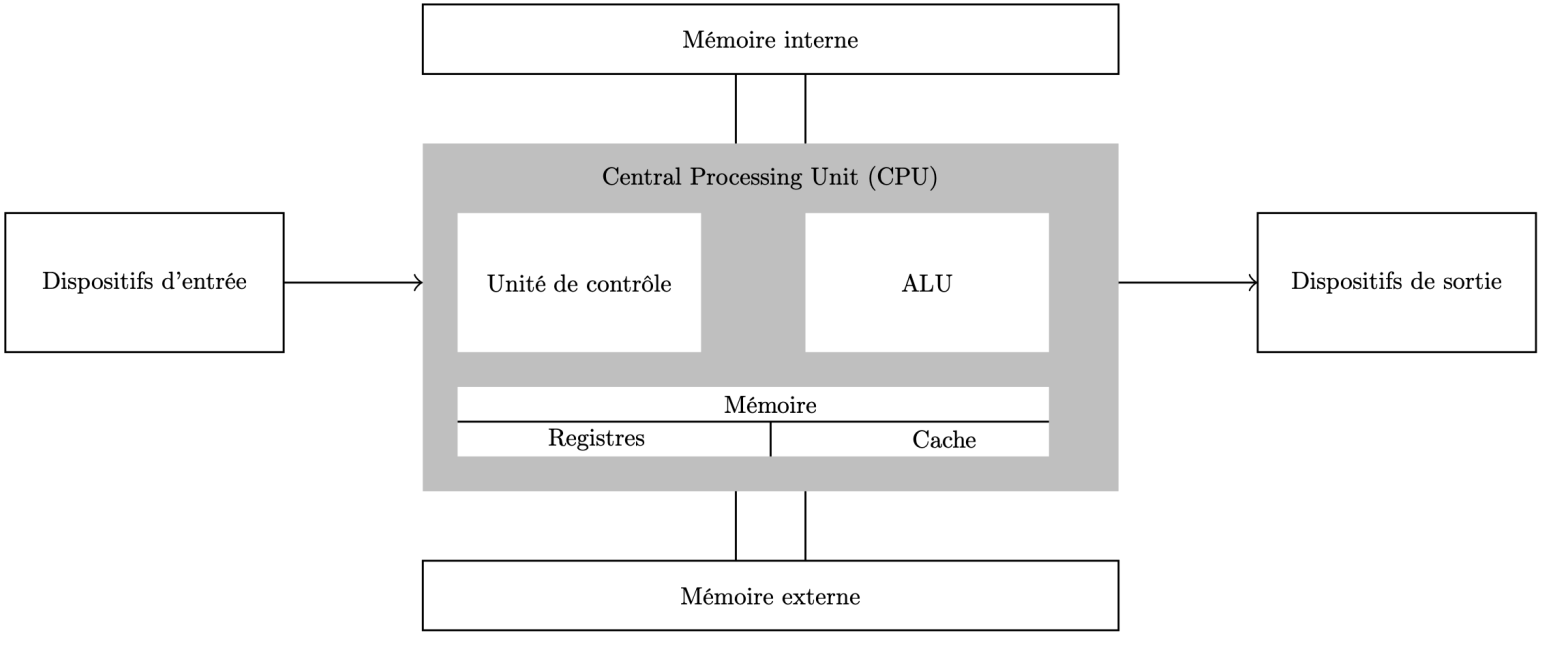
INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

cnrs dépasser les frontières

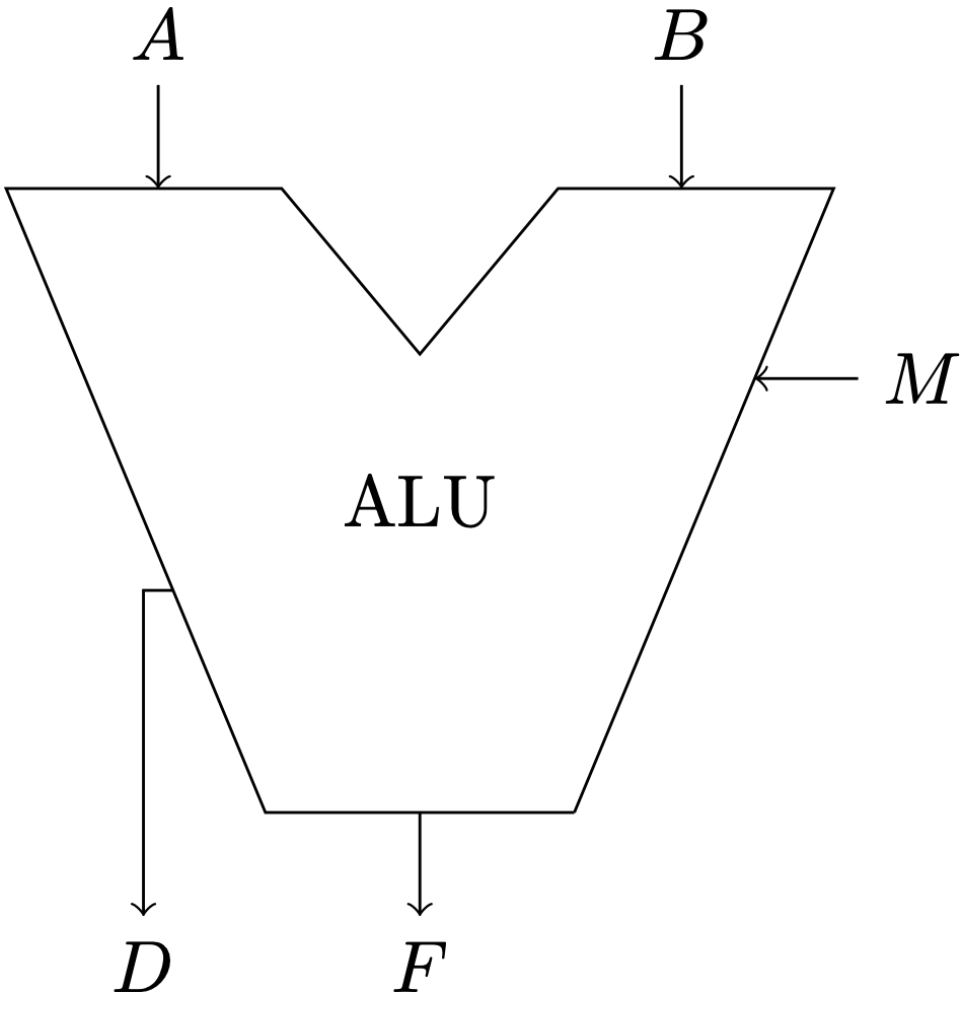
- Tests et examens
  - CC : résultat des tests en TD / TP (semaine 4 et **semaine 10 ou 11**)
  - E0 : partiel (samedi 26 octobre)
  - E1 : examen (19 décembre de 8h30 à 11h:30)
    - Examen écrit
  - E2 : examen fin juin
- Notes finales
  - Note session 1 : 25% CC + 25% E0 + 50% E1
  - Note session 2 : max(E2, 33% CC + 67% E2)
- Rappel
  - Pas de note $\Rightarrow$ pas de moyenne $\Rightarrow$ pas de semestre
- Site web
  - `moodlesupd.script.univ-paris-diderot.fr`

- Numération et arithmétique
- Numération et arithmétique en machine
- Numérisation et codage (texte, images)
- Compression, cryptographie, contrôle d'erreur
- Logique et calcul propositionnel
- **Circuits numériques**

- De quoi sont constitués (principalement pour ce qui nous concerne) les ordinateurs ?
  - D'un processeur effectuant des opérations élémentaires (CPU)
  - D'une mémoire interne
    - Les deux s'échangeant des informations continument
  - D'une mémoire externe et de dispositifs d'entrée et de sortie



- L'**unité arithmétique et logique** (en anglais Arithmetic–Logic Unit, **ALU**) est l'organe de l'ordinateur chargé d'effectuer les calculs
- Les ALU élémentaires calculent sur des nombres entiers, et peuvent effectuer les opérations communes, que l'on peut séparer en quatre groupes
  - Arithmétiques : addition, soustraction, changement de signe, ...
  - Logiques : compléments à un, à deux, et, ou, ou-exclusif, non, non-et, ...
  - Comparaisons : test d'égalité, supérieur, inférieur, et leur équivalents « ou égal »
  - Éventuellement des décalages et rotations (mais parfois ces opérations sont externalisées)
- Certaines ALU sont spécialisées dans la manipulation des nombres à virgule flottante
- Certaines ALU, le plus souvent de la classe des FPUs, sont susceptibles d'offrir des fonctions avancées

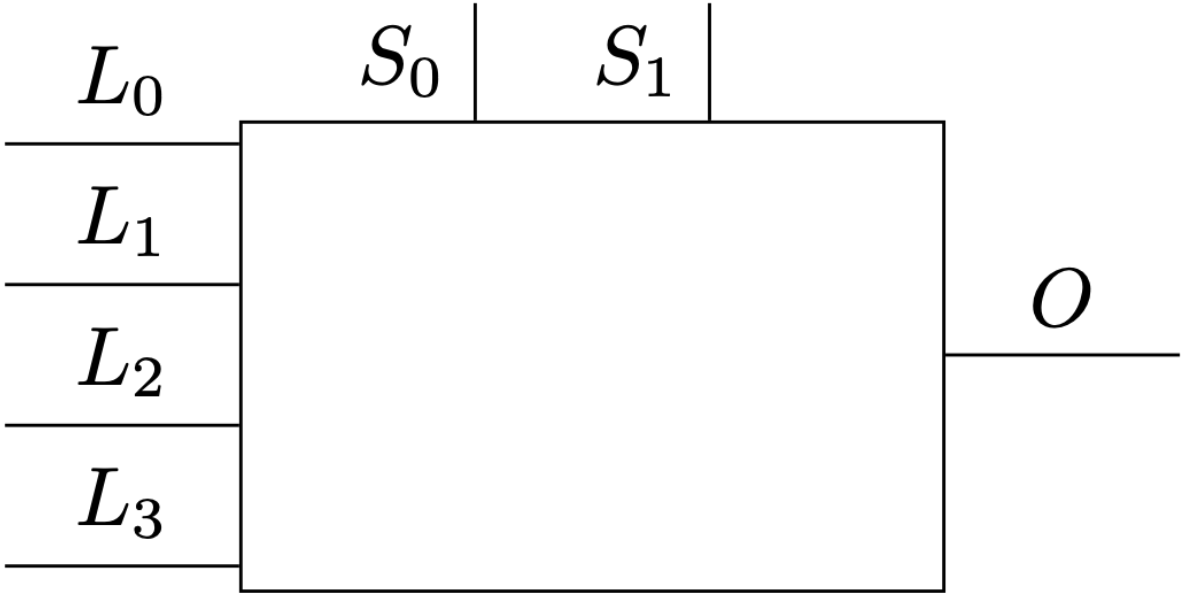


- Deux entrées A et B sur lesquelles on présente les données à traiter
- L'entrée M désigne l'opération à effectuer
- La sortie R est le résultat de l'opération
- La sortie D les éventuels drapeaux
  - Carry-out
  - Zéro
  - Négatif
  - Overflow

- Exemple : ALU 74181

Fonction				M = 1	M = 0	
S3	S2	S1	S0	Opération logique	Cn = 0	Cn = 1
0	0	0	0	F = non A	F = A	F = A + 1
0	0	0	1	F = non (A ou B)	F = A ou B	F = (A ou B) + 1
0	0	1	0	F = (non A) et B	F = A ou (non B)	F = (A ou (non B)) + 1
0	0	1	1	F = 0	F = - 1	F = 0
0	1	0	0	F = non (A et B)	F = A + (A et (non B))	F = A + (A et (non B)) + 1
0	1	0	1	F = non B	F = (A ou B) + (A et (non B))	F = (A ou B) + (A et(non B)) + 1
0	1	1	0	F = A xor B	F = A - B - 1	F = A - B
0	1	1	1	F = A et (non B)	F = (A et (non B)) - 1	F = A et (non B)
1	0	0	0	F = (non A) ou B	F = A + (A et B)	F = (A + (A et B)) + 1
1	0	0	1	F = non (A xor B)	F = A + B	F = A + B + 1
1	0	1	0	F = B	F = (A ou (non B)) + (A et B)	F = A ou (non B) + (A et B) + 1
1	0	1	1	F = A et B	F = (A et B) - 1	F = A et B
1	1	0	0	F = 1	F = A + (A << 1)	F = A + A + 1
1	1	0	1	F = A ou (non B)	F = (A ou B) + A	F = (A ou B) + A + 1
1	1	1	0	F = A ou B	F = (A ou (non B)) + A	F = A (non B) plus A plus 1
1	1	1	1	F = A	F = A - 1	F = A

- Multiplexeur
  - Permet de sélectionner une entrée par son numéro et de la reproduire en sortie
  - Fonction $\{0,1\}^m \times \{0,1\}^n \to \{0,1\}$ ou $m = 2^n$



  - $O = L_{S_1 S_0}$

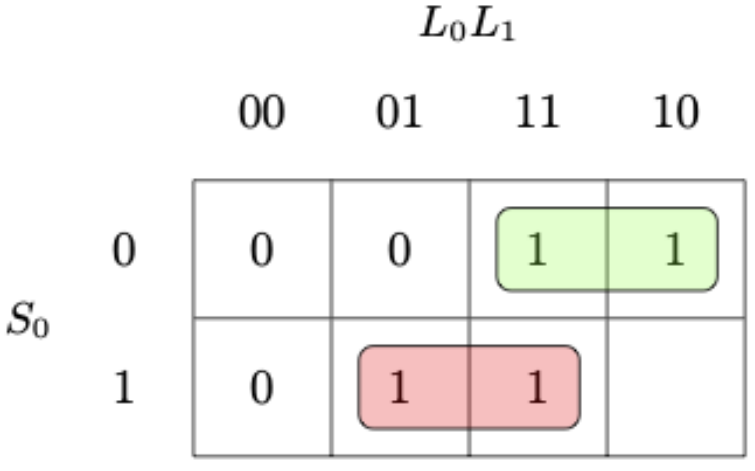
- Multiplexeur : $m = 2$ et $n = 1$
  - Table de vérité

L_0	L_1	S_0	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Multiplexeur : $m = 2$ et $n = 1$
  - Table de vérité

L_0	L_1	S_0	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

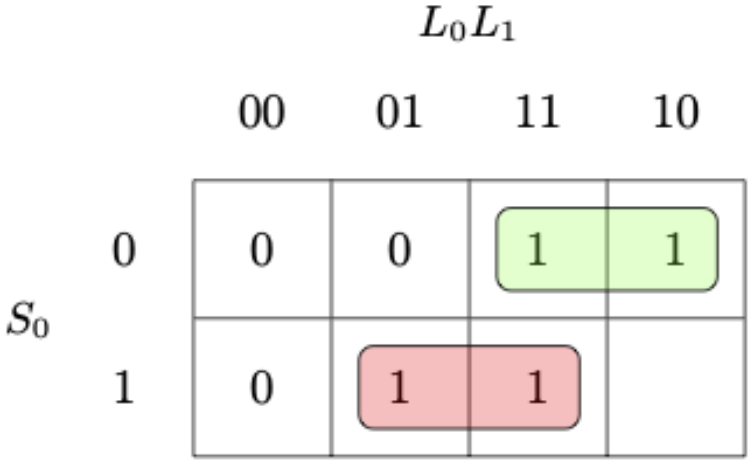
  - Table de Karnaugh



- Multiplexeur : $m = 2$ et $n = 1$
  - Table de vérité

L_0	L_1	S_0	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

  - Table de Karnaugh



  - Expression
    - $O = (\neg S_0 \wedge L_0) \vee (S_0 \wedge L_1)$

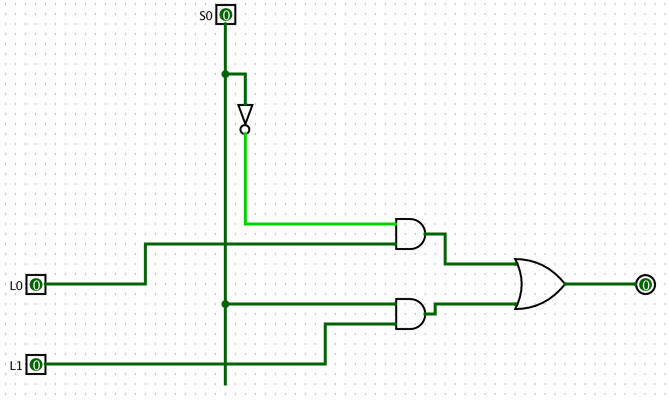
- Multiplexeur : $m = 2$ et $n = 1$
  - Table de vérité

L_0	L_1	S_0	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

  - Table de Karnaugh

S_0 \ L_0L_1	00	01	11	10
0	0	0	1	1
1	0	1	1	

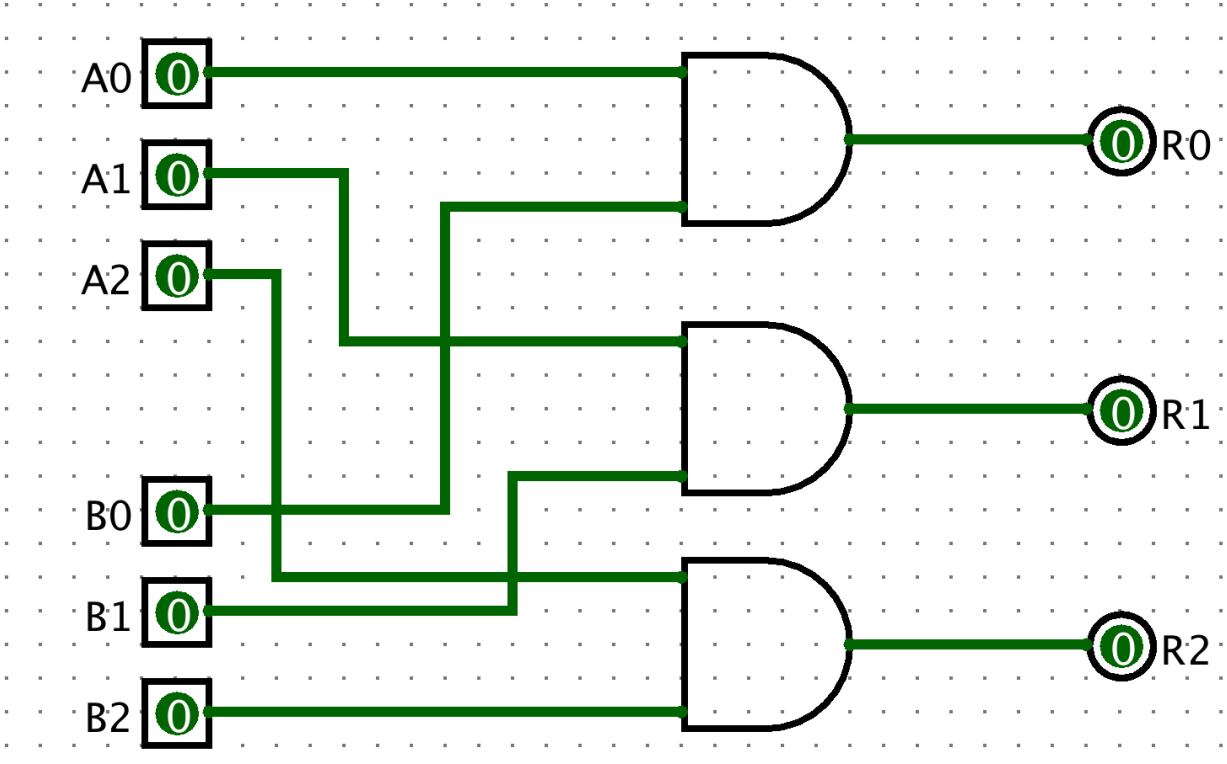
  - Circuit



  - Expression
    - $O = (\neg S_0 \wedge L_0) \vee (S_0 \wedge L_1)$

- 3 bits
  - 2 opérations
    - Et bit à bit
    - Addition

- 3 bits
  - 2 opérations
    - Et bit à bit
    - Addition
  - Et bit à bit : circuit



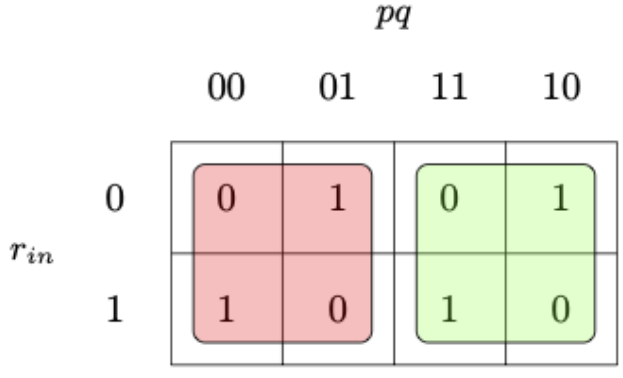
- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

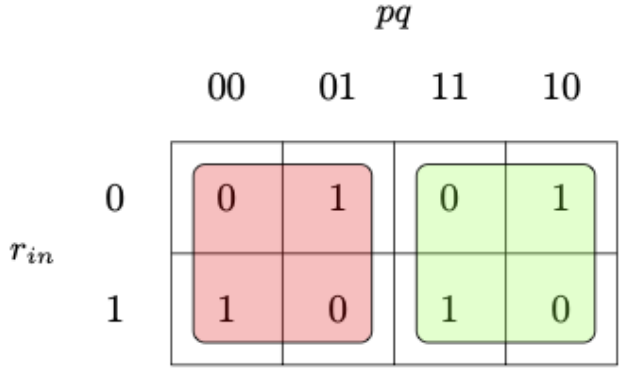
  - Table de Karnaugh de s



- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

  - Table de Karnaugh de s



  - Expression de s
    - $s = (\neg p \wedge (q \oplus r_{in})) \vee (p \wedge \neg(q \oplus r_{in})) = p \oplus q \oplus r_{in}$

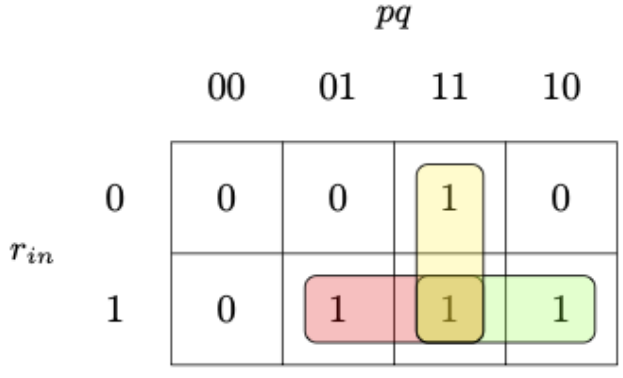
- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

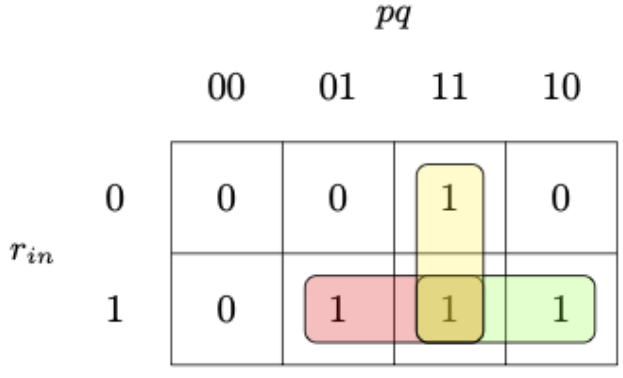
  - Table de Karnaugh de r



- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Table de vérité

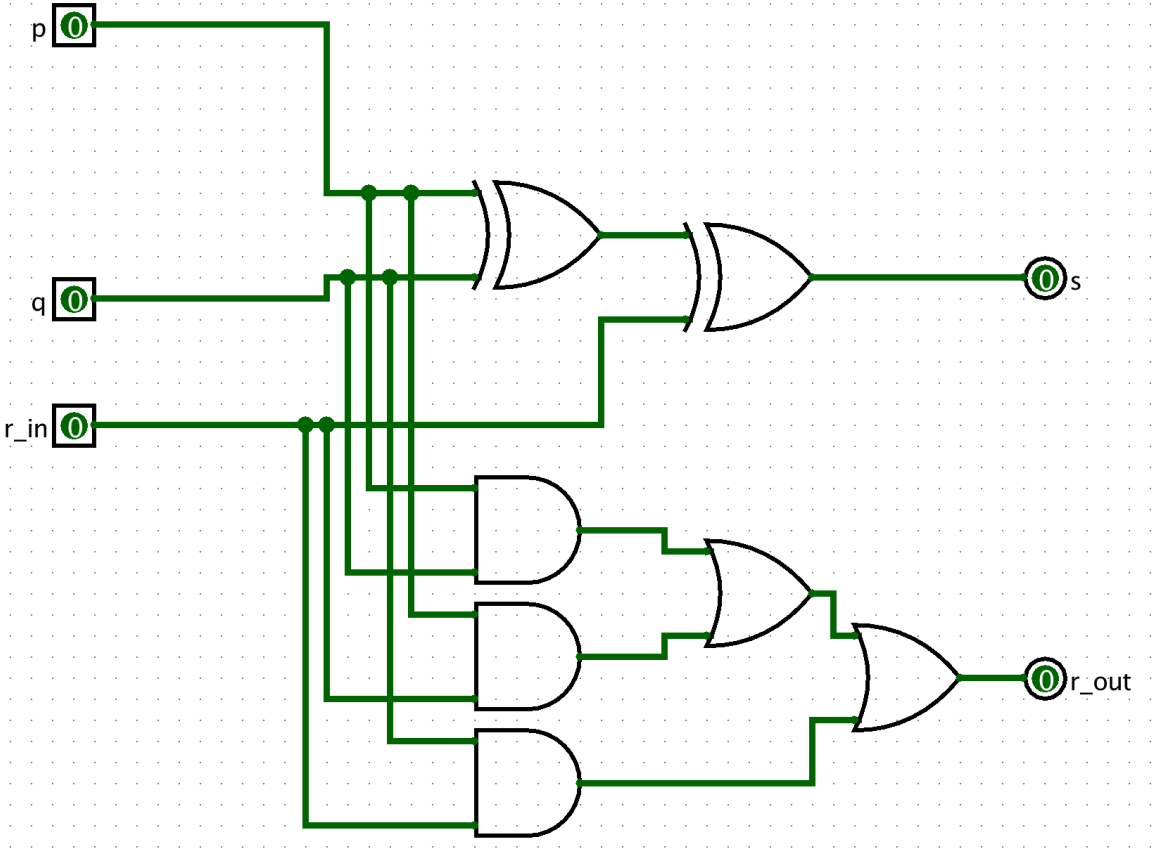
p	q	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

  - Table de Karnaugh de r

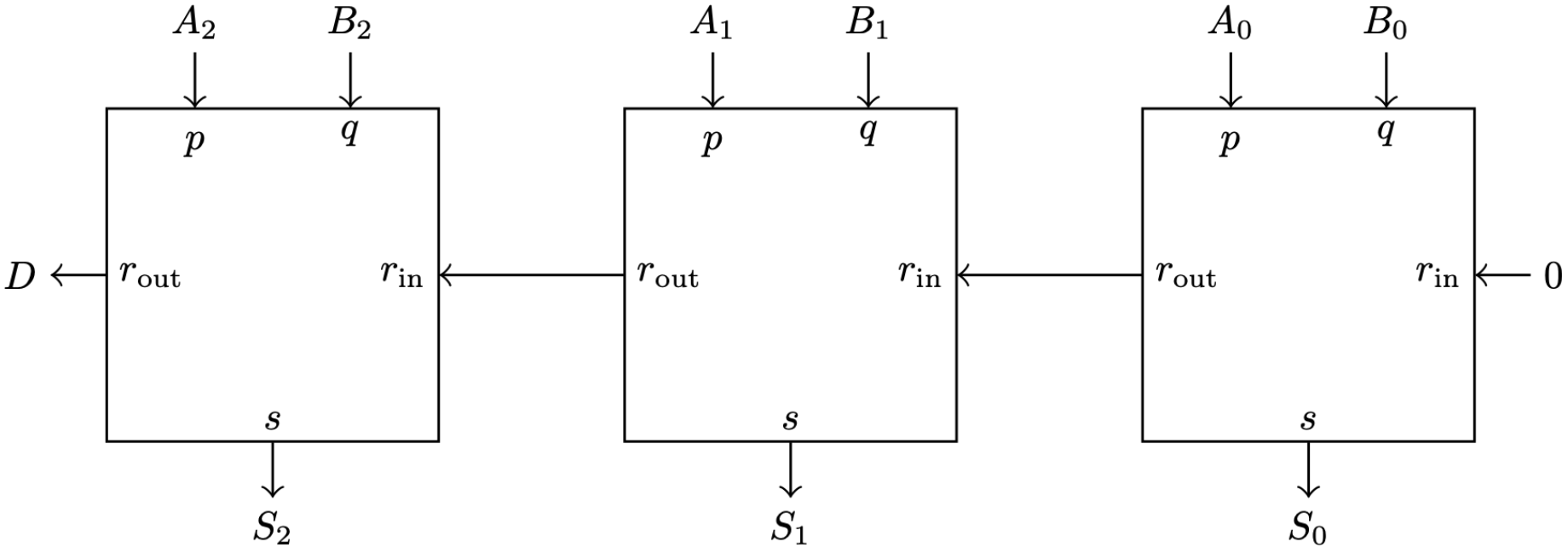


  - Expression de r
    - $r_{out} = (q \wedge r_{in}) \vee (p \wedge r_{in}) \vee (p \wedge q)$

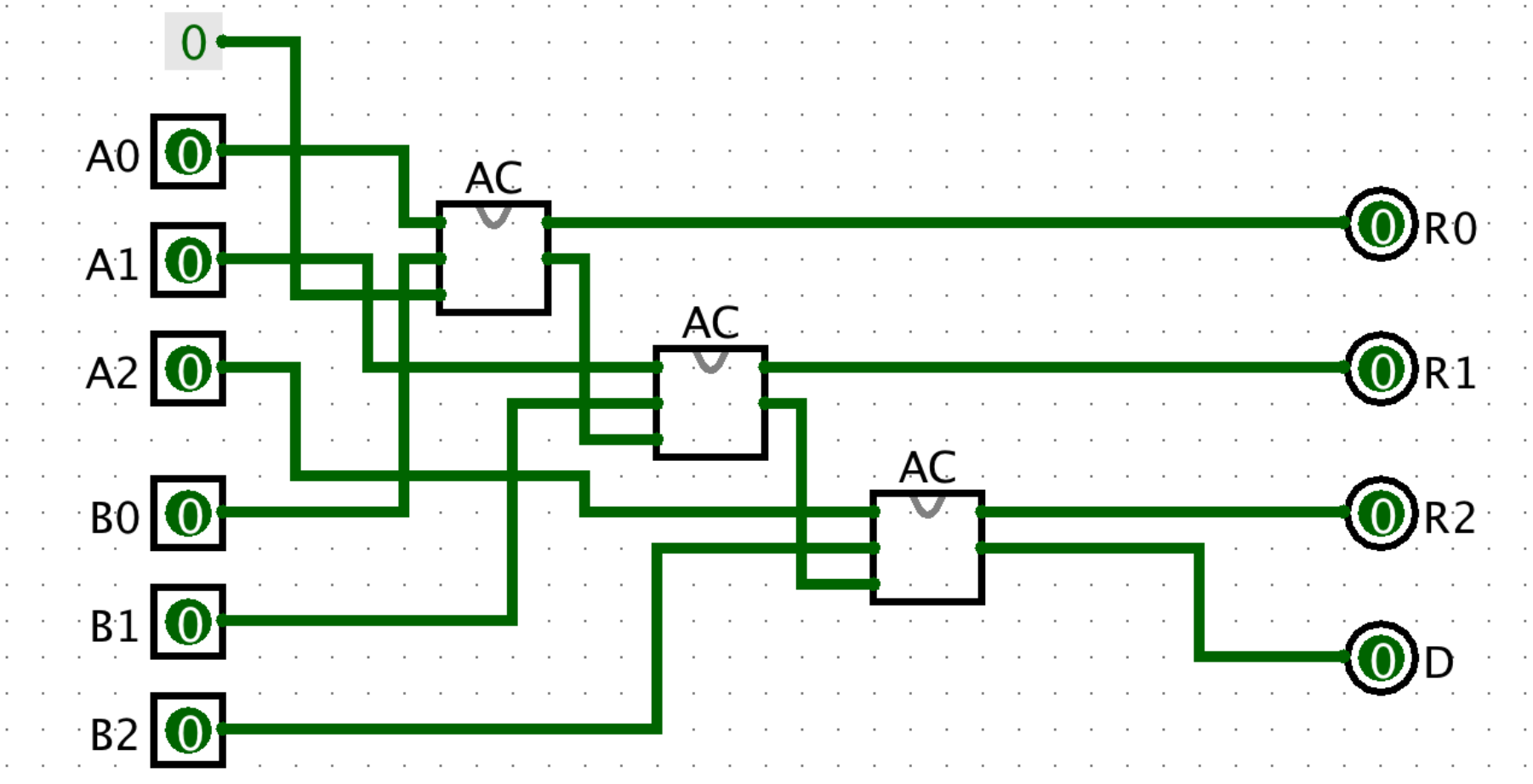
- Additionneur complet
  - Il permet de réaliser l'addition des deux bits p et q et la prise en compte d'une retenue éventuelle
  - Circuit



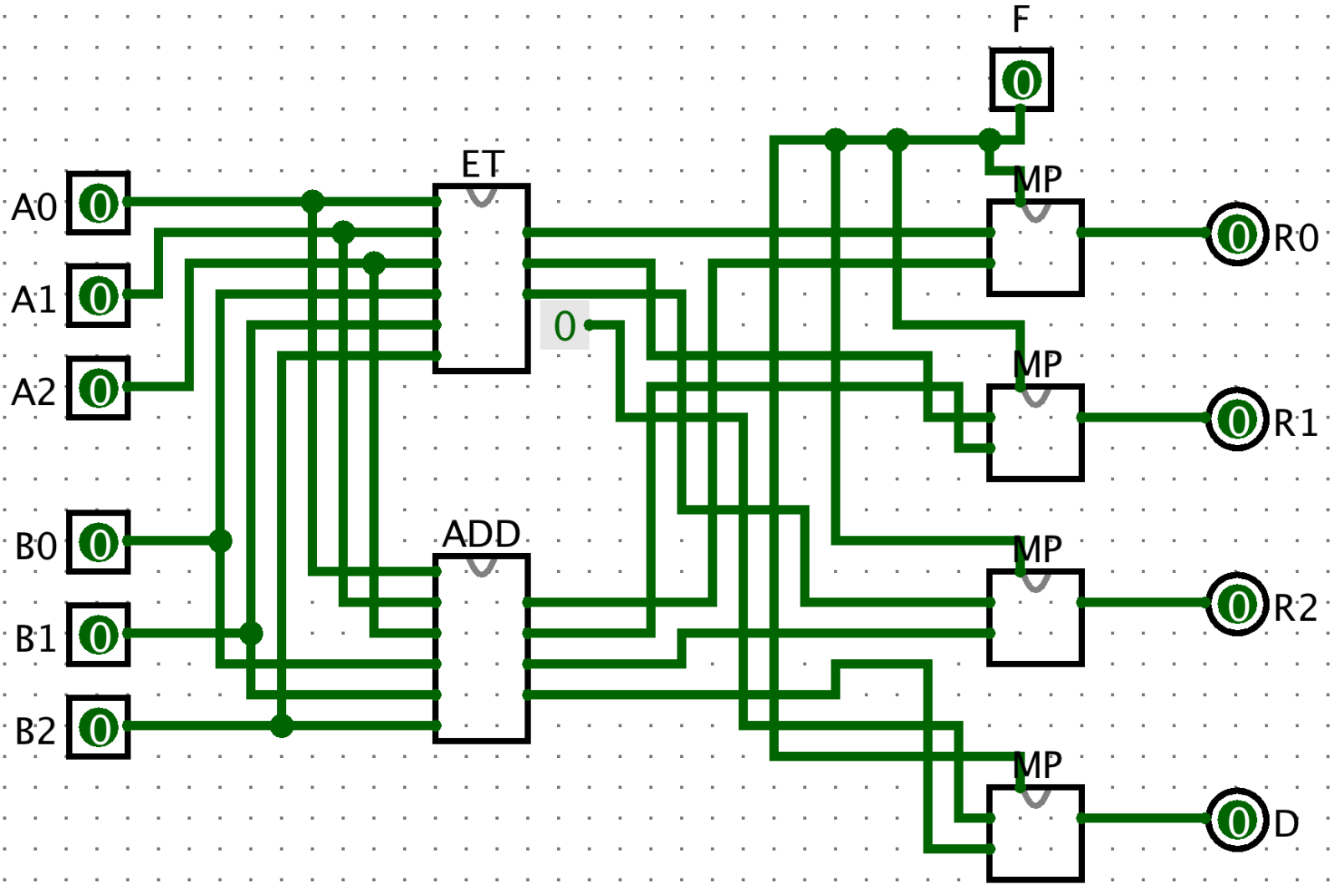
- Additionneur (3 bits)



- Additionneur (3 bits)



- ALU

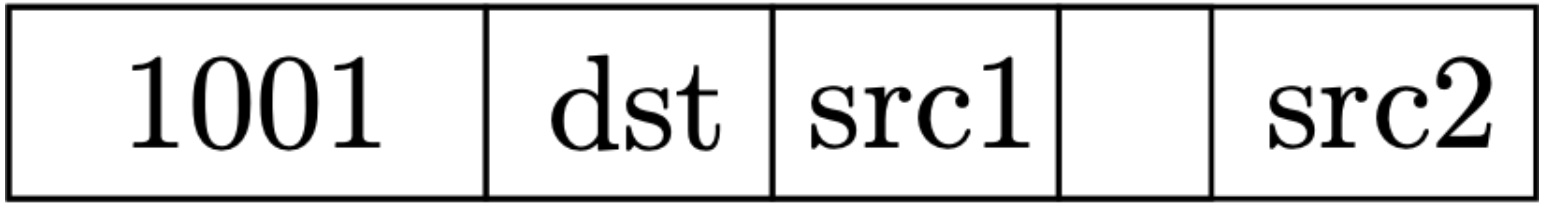


- Question : ajouter l'opération de soustraction

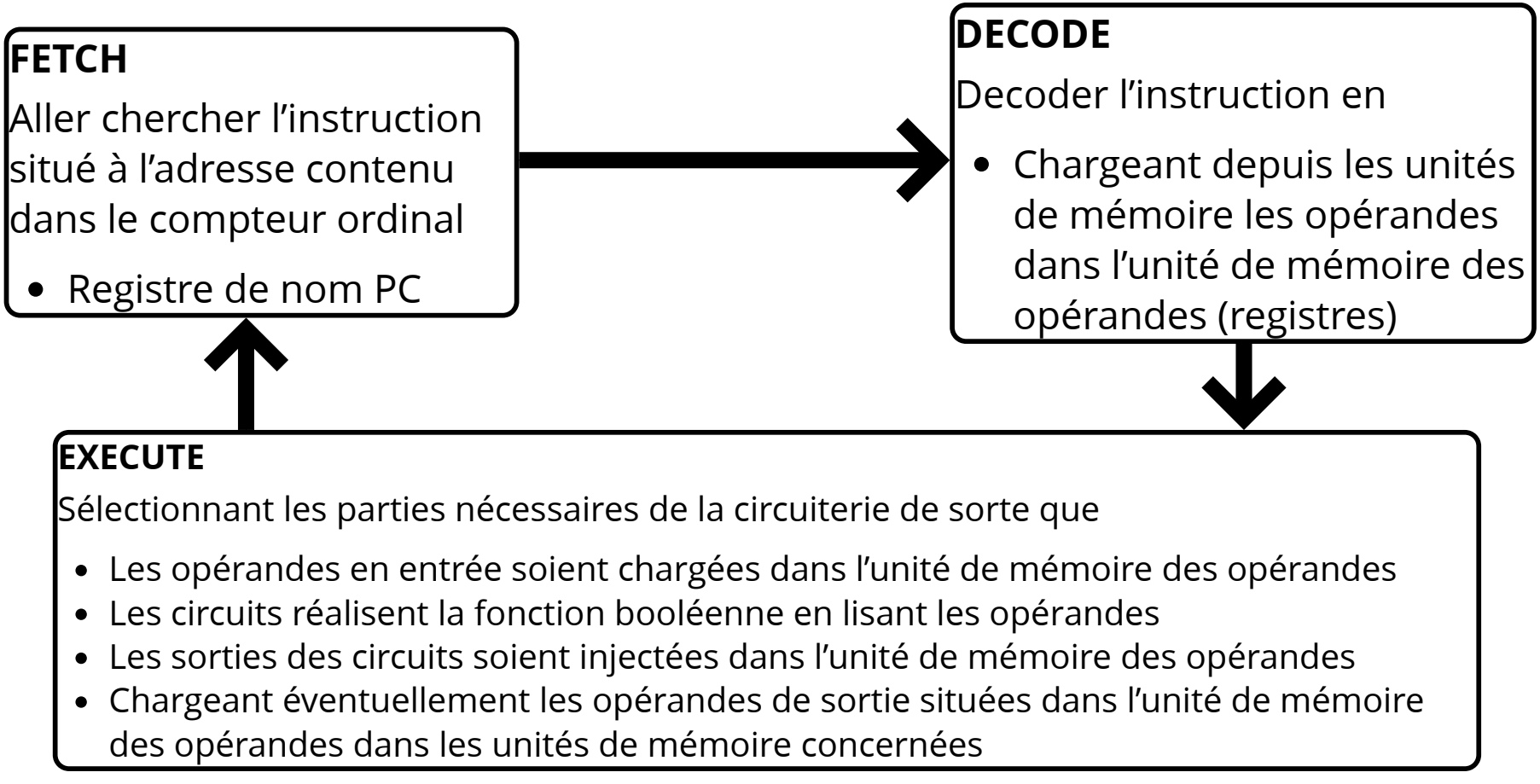
- Les ordinateurs sont des circuits complexes mais dont le principe de fonctionnement est assez simple
  - Le cœur est le processeur dont le rôle est d'exécuter des instructions en provenance de la mémoire
  - Une instruction est lue depuis la mémoire et interprétée en
    - Lisant des valeurs depuis la mémoire
    - Sélectionnant la partie du circuit permettant de réaliser la fonction booléenne correspondante
    - Écrivant le résultat en mémoire
  - Les instructions sont codées
    - Le code permet par l'intermédiaire d'un décodeur de sélectionner le circuit qui réalisera le calcul
    - Les codes sont des mots binaires
      - Par exemple pour un processeur de la famille Little Computer 3, la négation du registre R1 se code sur 16 bits par 1001001001111111

- Les ordinateurs actuels ont de nombreuses unités de mémorisation (vives) dont
  - Les **registres** : très rapides (1ns) mais en nombre ridicule (<100)
    - L'unité est le mot (couramment 64 bits)
    - Chaque registre a un **nom** qui lui est propre
  - Les **caches** (cache) : relativement rapide (10ns) mais de taille assez faible (Ko-Mo)
  - La **mémoire vive** (RAM) : relativement lente (50ns) mais plus volumineuse (Go)
    - L'unité est l'octet
    - Chaque octet a une **adresse** qui lui est propre

- Une instruction du langage machine est directement interprétable par le processeur
  - La collection des instructions disponibles est appelé **jeu d'instructions**
  - Une **instruction** contient plusieurs informations codées
    - Le **code de l'instruction** à réaliser (correspond essentiellement à la désignation des circuits nécessaires)
    - Les **adresses des opérandes** dans les unités de mémoire
  - Exemple : addition



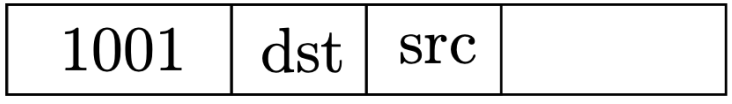
- L'exécution d'une instruction consiste en les opération

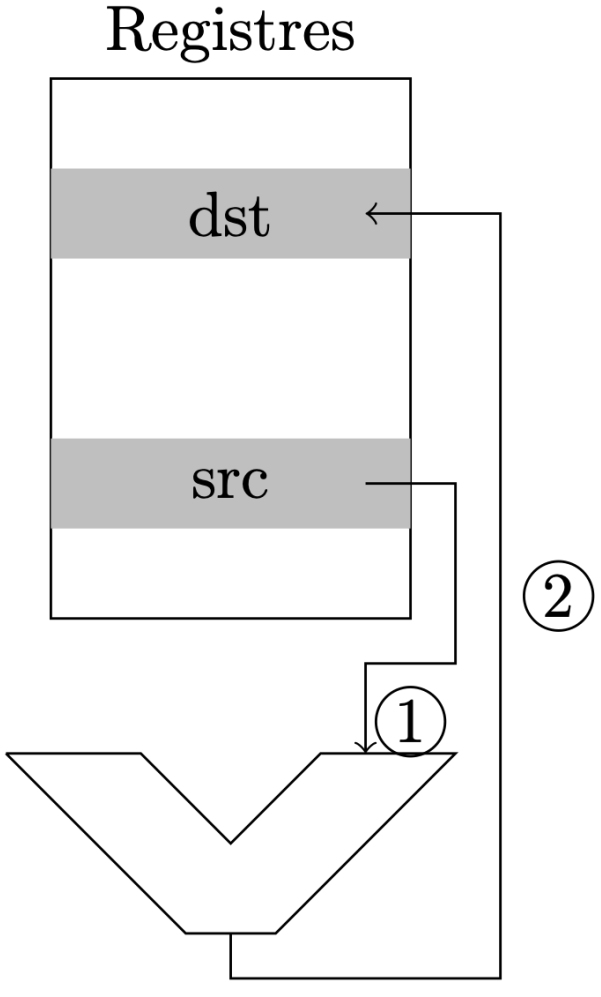


- Passer à l'instruction suivante (généralement passage en séquence, mais parfois saut "if-then")

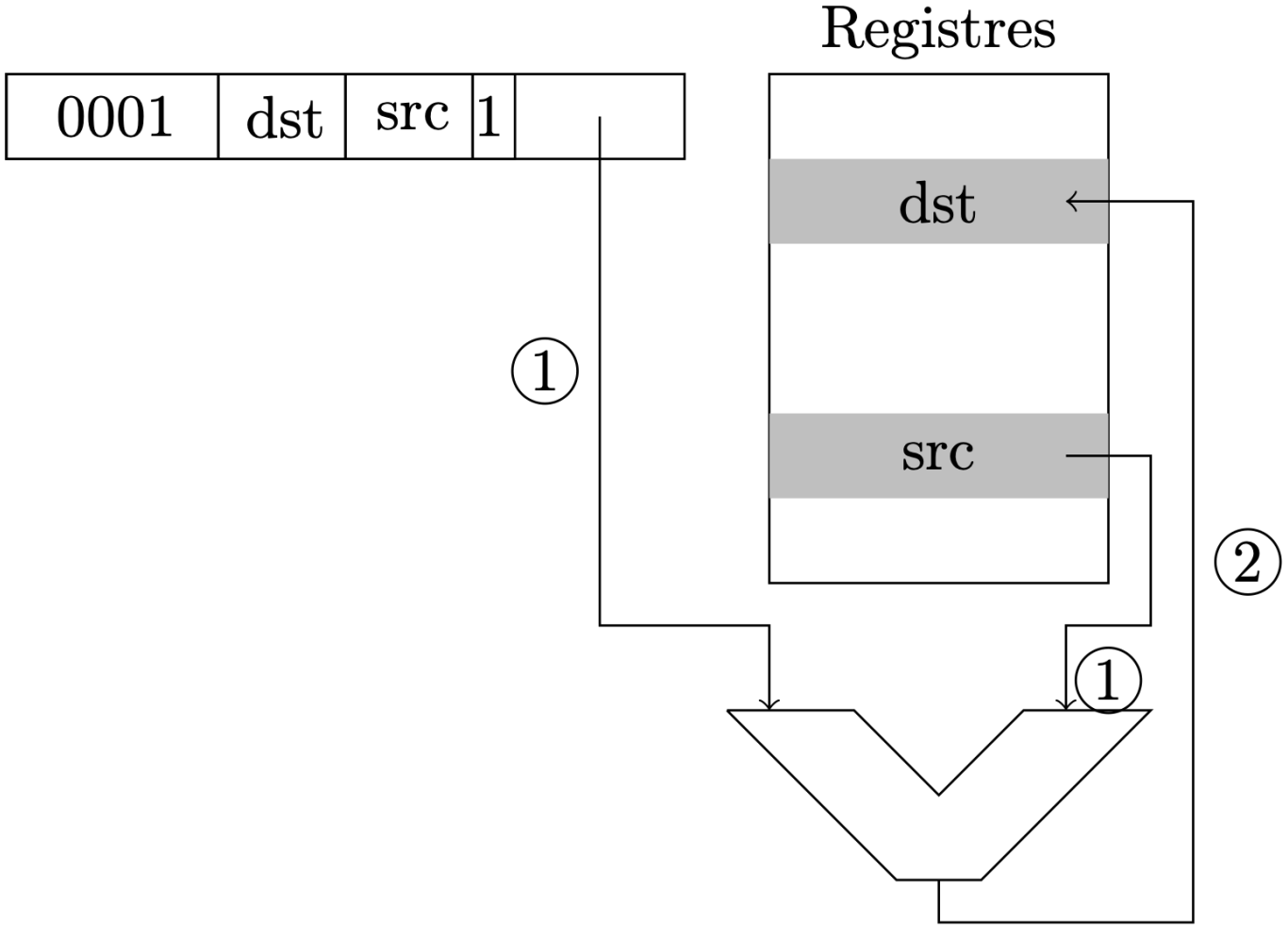
- Il existe différents modes d'adressages en mémoire (différentes façons de désigner une valeur)
  - Le mode **registre** : la valeur de l'opérande est le numéro d'un registre dans lequel la valeur est stockée
  - Le mode **immédiat** : la valeur de l'opérande est la valeur qui devra être utilisée
  - Le mode **direct** : la valeur de l'opérande est l'adresse d'un emplacement en mémoire dans lequel la valeur est stockée
  - Le mode **indexé** ou **relatif** : la valeur de l'opérande est l'adresse relative d'un emplacement mémoire dans lequel la valeur est stockée (le point d'origine est en général contenu dans un registre spécial)
  - Le mode **indirect** : la valeur de l'opérande est l'adresse d'un emplacement dans lequel est stockée l'adresse d'un emplacement dans lequel est stockée la valeur

- Mode registre
  - Négation (code 1001)

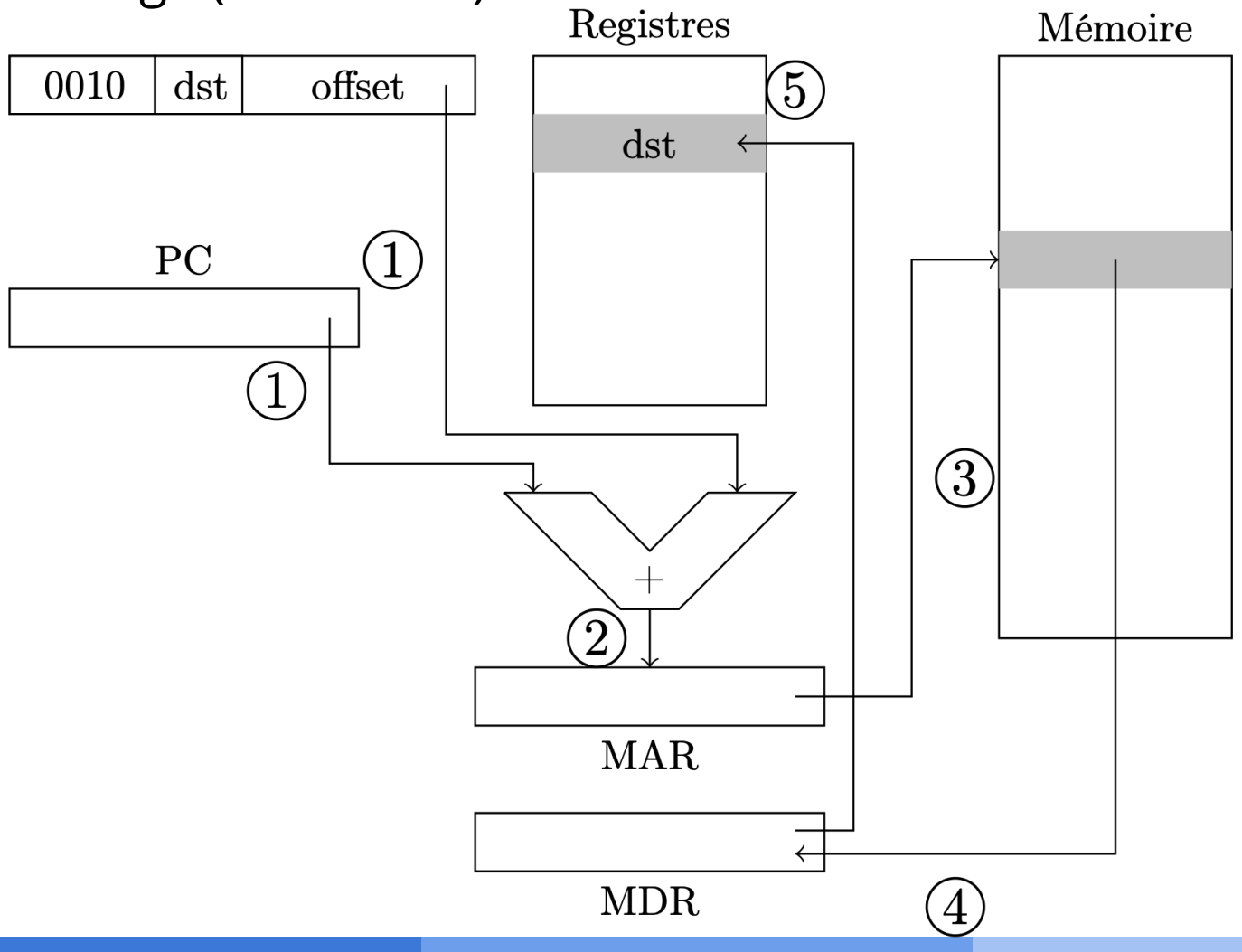




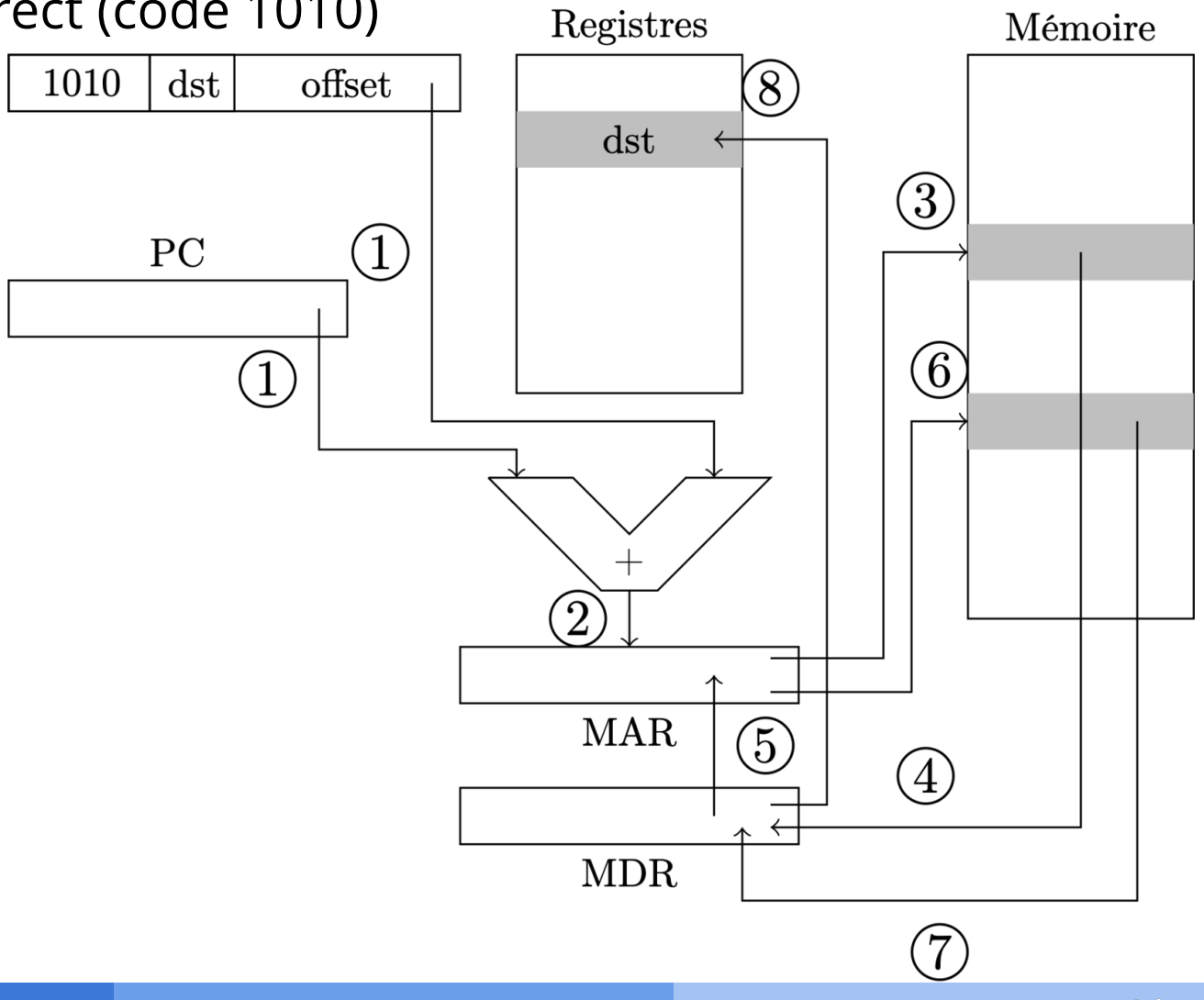
- Mode immédiat
  - Addition (code 0001)



- Mode relatif
  - Charge (code 0010)



- Mode indirect
  - Charge indirect (code 1010)



- Les instructions d'une machine peuvent être rangées dans les catégories suivantes
  - Instructions **arithmétiques**
  - Instructions **logiques**
  - Instructions de **lecture/écriture** en mémoire
  - Instructions de **contrôle** (déroutement du flux normal)