ARCHITECTURE DES ORDINATEURS

EXERCICES

François SCHUMACKER, Ir

B1 – Bachelier en Développement d'Applications
Année académique 2025-2026

1 Histoire de l'informatique

1.1 Des nombres et des hommes

1. Donner le nom du système de numération et calculer la valeur décimale des nombres suivants :

a. 1010_2

b. 42031_5

c. 123_7

d. 777_8

e. 12345_{12}

2 Présentation générale d'un ordinateur

Pas d'exercices sur le chapitre 2

3 Codage des informations

3.1 Systèmes de numération

3.1.1 Systèmes de numération utilisés en informatique

1. **Compter de 0 à 15 en binaire.**

2. **Compter de 0 à 15 en base 5.**

3. **Compter de 0 à 15 en octal.**

3.1.2 Formule générale de définition d'un nombre

On peut définir de manière générale un nombre **N** exprimé en base **B** de la manière suivante :

$N_B = a_n B^n + a_{n-1} B^{n-1} + \dots + a_0 B^0 + a_{-1} B^{-1} + \dots + a_{-m} B^{-m} = \sum_{i=-m}^{n} a_i B^i$ **Forme expansée**

$N_B = a_n a_{n-1} \dots a_2 a_1 a_0 , a_{-1} \dots a_{-m}$ (ex: 1234,56) **Forme normale**

Où a_i est un des symboles de la base B.

Les exposants positifs (ou nul) correspondent à la partie entière du nombre et les exposants négatifs à sa partie fractionnaire.

Voir section 3.2.1 pour des exercices utilisant la formule générale de représentation des nombres entiers et réels.

3.1.3 Opérations arithmétiques

3.1.3.1 Addition et multiplication en décimal (base 10)

1. **Calculez à la main la somme : 12.345.678 + 23.456.789.**

2. **Calculez à la main le produit : 1234,56 x 678,9.**

3.1.3.2 Addition et multiplication en base quelconque

1. **Calculez à la main la somme binaire : $10110011_2 + 10111010_2$.**

2. **Calculez à la main la somme en base 7 : $456321_7 + 126154_7$.**

3. **Calculez à la main la somme en base 16 : $1A2B3C4D_{16}$ + $A012D789_{16}$.**

4. **Considérons l'addition suivante : 17731 + 81847 = 99078.**
   Quelle doit être la base de numération utilisée pour que cette addition soit correcte ?
   Dans cette base, quel sera le résultat de l'addition suivante 73287 + 47478 ?

5. **Considérons l'addition suivante : 33825 + 64613 = 98238.**
   Quelle doit être la base de numération utilisée pour que cette addition soit correcte ?
   Dans cette base, quel sera le résultat de l'addition suivante 27544 + 44772 ?

3.2 Conversions entre bases

3.2.1 Nombres entiers et réels : base B vers décimal

1. **Quel est le plus grand nombre de 3 chiffres en base 7 ?**

2. **Quel est le plus grand nombre de 4 chiffres en base 16 ?**

3. Calculez la valeur décimale de nombres entiers suivants :

Il suffit d'appliquer la formule générale et d'additionner les puissances successives de la base B.

a. 110011_2

b. 1000110011101110_2

c. 210012_3

d. 3210_4

e. 5632_7

f. 007_8

g. 428_8

h. 2014_8

i. 2014_{16}

j. ADA_{16}

k. $BEBE_{16}$

l. CAT_{16}

m. $BAFFE_{16}$

4. Calculez la valeur décimale de nombres réels suivants :

Pour les nombres réels, on applique également la formule générale, mais on additionne les puissances positives et négatives de la base B.

a. $10111,1001_2$

b. $123,456_8$

c. CA,FE_{16}

3.2.2 Nombres entiers : décimal vers base B

3.2.2.1 Division par la base

La technique de *division par la base* consiste à répéter les opérations suivantes :

1. Effectuer la division entière de N par la base B ($\frac{N}{B}$) et déterminer le quotient entier Q.
2. Le reste $R = N - Q \times B$ est le coefficient a_0 du terme $a_0 B^0$.
3. Recommencer au point 1 avec $N' = Q$ afin de déterminer les coefficients suivants ($a_1, a_2, \ldots$) et cela tant que $Q \neq 0$.

1. Convertir 51529 en binaire.

2. Convertir 816 en binaire.

3. Convertir 65536 en octal.

4. Convertir 47802 en hexadécimal.

5. Convertir 12345 en base 7.

3.2.2.2 Conversion en binaire par soustraction des puissances de 2

Soustraire les puissances décroissantes successives de 2 du nombre N à convertir, en commençant par la grande puissance de 2 qui est inférieure ou égale à N. On indique 1 à la position *n* du résultat lorsqu'il est possible de soustraire 2^n, 0 sinon

1. **Convertir 227 en binaire.**

2. **Convertir 176 en binaire.**

3. **Convertir 2876 en binaire.**

3.2.3 Nombres réels : décimal vers base B

Soit N = X,Y un nombre réel. On souhaite convertir la partie fractionnaire Y vers une base B.
La ***méthode de multiplication par la base*** consiste à répéter les opérations suivantes :

1. On multiplie Y par la base B.
2. On obtient un nombre réel N' = B x Y = X',Y'.
3. X' est le premier chiffre de la partie fractionnaire en base B.
4. On répète le processus à l'étape 1 en utilisant Y', jusqu'à obtenir Y'=0 ou avoir atteint la précision maximale disponible.

Attention, une partie fractionnaire finie dans une base donnée peut conduire à une partie fractionnaire qui est illimitée dans une autre base !

1. **Convertir $5{,}796875_{10}$ en binaire.**

2. **Convertir $59{,}85546875_{10}$ en octal.**

3. **Convertir $15{,}0859375_{10}$ en hexadécimal.**

4. **Convertir $12{,}34_{10}$ en binaire.**

3.2.4 Bases qui sont des puissances de 2

Pour convertir **d'une base B = 2^n vers le binaire**, on transforme chaque chiffre de la base B en sa représentation en *n* chiffres binaires ou **bits** (= *binary digits*) et on concatène les différents morceaux.

1. **Convertir $ABCD_{16}$ en binaire.**

2. **Convertir 123456_8 en binaire.**

Pour convertir **du binaire vers une base B = 2^n**, on découpe le nombre binaire en tronçons de *n* bits en allant de la droite vers la gauche. Chaque tronçon correspond à un chiffre en base B. Il faut éventuellement compléter le dernier tronçon de gauche avec des 0.

3. **Convertir 1011111010111110_2 en hexadécimal.**

4. **Convertir 10111000110101_2 en octal.**

Pour convertir **entre deux bases B et B'** qui sont des puissances de 2, on procède en deux étapes :
1. Conversion de la base B vers le binaire.
2. Conversion du binaire vers la base B'.

5. Convertir $4F16_{16}$ en octal.

6. Convertir 777_8 en hexadécimal.

3.3 Représentation interne des informations

Pas d'exercices

3.4 Représentation des nombres entiers

3.4.1 Nombres naturels

1. Combien de nombres naturels peut-on représenter sur 32 bits ?

2. Quelle est la valeur du plus grand nombre naturel représentable sur 16 bits ?

3. Si je souhaite représenter les nombres naturels de 0 à 63, de combien de bits ai-je besoin ?

4. Calculez la représentation binaire sur 16 bits des nombres décimaux suivants :

a. 11111_{10}

b. 54321_{10}

c. 66666_{10}

5. Calculez la représentation binaire sur 16 bits des nombres hexadécimaux suivants :

a. 1111_{16}

b. ABC_{16}

c. $R2D2_{16}$

d. $A9F2_{16}$

6. **Quelle est la caractéristique de la représentation interne binaire de tous les nombres naturels qui sont des multiples de 8 (0, 8, 16, 24, …, 800, …, 8000, …, 36504, …) ?**

7. **En utilisant la représentation binaire, déterminez le plus grand multiple de 32 représentable sur 16 bits ?**

8. **Représentez le nombre 102_{10} en binaire sur 8 bits.**

    a. Que se passe-t-il si on décale tous les bits de cette représentation d'une position vers la droite ?

    b. Que se passe-t-il si on décale tous les bits de cette représentation d'une position vers la gauche ?

9. **Si on dispose de la représentation binaire d'un nombre décimal, comment peut-on obtenir facilement le quotient et le reste (modulo) de la division de ce nombre par 2, 4, 8, 16, … ?**

10. **Considérons un circuit mémoire de 64K (64x1024 octets). Pour que chaque octet soit accessible au moyen d'une adresse mémoire unique, de combien d'adresses avons-nous besoin ? Quelle sera la taille (minimale) en bits d'une adresse ?**

11. On décide de diviser cette mémoire en 64 blocs de 1024 octets qui se suivent en mémoire. Les blocs sont numérotés de 0 à 63, les octets au sein de chaque bloc de 0 à 1023. Le bloc 0 commence à l'adresse 0, le bloc 1 à l'adresse 1024, le bloc 2 à l'adresse 2048, etc.

a. Quelle est l'adresse de l'octet 723 du bloc 45 ?

b. En partant de la représentation binaire d'une adresse, comment peut-on facilement obtenir le numéro du bloc et le numéro de l'octet au sein du bloc (offset) ?

12. Dans le système RGB (Red/Green/Blue), une couleur est représentée par un triplet de nombres compris entre 0 et 255 (8 bits) qui indiquent l'intensité de chaque couleur de base pour obtenir la couleur composée. Ainsi, le triplet (153, 51, 255) représente une nuance de mauve. On peut également représenter le code couleur par un nombre hexadécimal de 6 chiffres (2 chiffres hexa par composante). Pour notre nuance de mauve, cela donne 0x9933FF (=$9933FF_{16}$).

a. Calculer le code hexadécimal de la couleur RGB (57, 122, 206).

b. Calculer les composantes RGB décimales de la couleur 0x12B022.

c. Calculer le code hexadécimal et les composantes RGB décimales de la couleur représentée par le nombre décimal 15971855_{10}.

3.4.2 Stockage en mémoire

3.4.2.1 Alignement des mots mémoires

Pas d'exercices

3.4.2.2 Représentations petit-boutiste et gros-boutiste

1. Considérons le nombre entier sur 32 bits : 2544829550.
Calculez les représentations hexadécimales gros-boutiste et petit-boutiste de ce nombre.

Valeur hexadécimale

Représentation en mémoire

Adresse mémoire →	N	N+1	N+2	N+3
Représentation gros-boutiste				
Représentation petit-boutiste				

2. Considérons le nombre entier sur 32 bits : 935693578.
Calculez les représentations hexadécimales gros-boutiste et petit-boutiste de ce nombre.

Valeur hexadécimale

Représentation en mémoire

Adresse mémoire →	N	N+1	N+2	N+3
Représentation gros-boutiste				
Représentation petit-boutiste				

3.4.3 Nombres entiers relatifs

3.4.3.1 Signe et valeur absolue

Le bit le plus significatif (c.-à-d. le plus à gauche) représente le *signe*. Les autres bits sont utilisés pour représenter la *valeur absolue* du nombre. Par convention, le signe + est représenté par la valeur 0 et le signe - par la valeur 1.

1. Calculer la représentation binaire sur 8 bits en signe et valeur absolue des nombres suivants :

a. 108

b. -108

c. -127

d. -1

e. -128

2. Calculer la valeur décimale des nombres binaires suivants, sachant qu'ils sont représentés en signe et valeur absolue :

a. 01010101

b. 11010101

c. 11100011

3.4.3.2 Complément à 2

Le *complément à 2*, ou *complément arithmétique*, s'obtient en ajoutant +1 à la valeur du complément à 1. **Méthode rapide** : on recopie tous les bits du nombre positif en commençant par la droite jusqu'au premier 1 inclus, après quoi on inverse tous les bits qui suivent.

1. Calculer la représentation binaire sur 8 bits en complément à 2 des nombres suivants :

a. 108

b. -108

c. -127

d. -1

e. -128

2. Calculer la valeur décimale des nombres binaires suivants, sachant qu'ils sont représentés en complément à 2 :

a. 01010101

b. 11010101

c. 11100011

3. Exercice récapitulatif 1 – calculer la représentation binaire sur 8 bits des nombres suivants :

a. 91

b. -91 en signe et valeur absolue

c. -91 en complément à 2

4. Exercice récapitulatif 2 – calculer la valeur décimale du nombre représenté par `10101010 si` :

a. entier non signé

b. signe et valeur absolue

c. complément à 2

Pour soustraire un nombre, il suffit d'additionner son complément. On additionne tous les chiffres et on laisse tomber le report final éventuel. Un *dépassement de capacité* est détecté par le fait que la retenue générée sur le bit de signe est différente de celle générée sur le bit juste avant.

5. Réaliser les opérations suivantes en arithmétique binaire sur 8 bits en complément à 2. Vérifier si le résultat binaire obtenu correspond bien à la valeur attendue.

a. 5 + 122 = 127

b. 122 + 64 = 186

c. 64 - 96 = -32

d. -96 - 32 = -128

e. -96 - 64 = -160

3.4.3.3 Extension de signe

Extension de signe : pour « agrandir » un nombre entier signé codé en complément à 2 (ou à 1) sur 8 bits vers son équivalent sur 16 bits (32 bits ou 64 bits), il convient de *recopier le bit de signe* dans les octets ajoutés à gauche.

1. Étendre les nombres binaires suivants de 8 à 16 bits :

a. 01110111

b. 10011100

3.5 Représentation des nombres réels

3.5.1 Virgule fixe

Pas d'exercices.

3.5.2 Virgule flottante

La représentation des nombres réels en virgule flottante telle que définie par le standard IEEE 754 consiste à stocker en mémoire la forme normalisée d'un nombre réel binaire de la manière suivante :

SM	EXPOSANT	MANTISSE

où

- SM est le signe de la mantisse. Par convention, 0 signifie + (positif) et 1 signifie – (négatif).
- L'exposant utilise la notation biaisée.
- Seule la partie fractionnaire de la mantisse normalisée est stockée (la partie entière vaut 1).

Rappel : la *forme normalisée d'un nombre réel binaire* définie par le standard IEEE 754 s'obtient en ajustant l'exposant de telle sorte que la partie entière de la mantisse soit égale à 1. La seule exception est le nombre 0 qui est représenté par un mot mémoire dont tous les bits sont à 0.

Précision	SM	EXPOSANT / BIAIS		MANTISSE
Simple (32 bits)	1 bit	8 bits / +127	[-127, +128]	23 bits
Double (64 bits)	1 bit	11 bits / +1023	[-1023, +1024]	52 bits
Étendue (80 bits)	1 bit	15 bits / +16383	[-16383, +16384]	64 bits

3.5.3 Conversion vers le format IEEE 754

Nous pouvons résumer le processus de conversion comme suit :

1. Identifier le *signe* du nombre à convertir.
2. Convertir la *partie entière* du nombre réel décimal vers son équivalent binaire.
3. Convertir la *partie fractionnaire* du nombre réel décimal vers son équivalent binaire.
4. *Normaliser* le nombre réel binaire obtenu aux étapes 2 et 3, ce qui nous donne la valeur de l'*exposant*.
5. Calculer la valeur décimale biaisée de l'exposant en lui ajoutant 127 (simple précision).
6. Convertir la valeur décimale biaisée de l'exposant vers son équivalent binaire sur 8 bits.
7. Assembler les différents morceaux pour obtenir la représentation finale en virgule flottante. Pour rappel, on ne stocke pas le bit qui correspond à la partie entière de la mantisse. Si la partie fractionnaire de la mantisse comporte moins de 23 bits (simple précision), on complète à droite avec des zéros.

1. **Convertir $456{,}8125_{10}$ en float IEEE 754 sur 32 bits.**

2. **Convertir $-123{,}321_{10}$ en float IEEE 754 sur 32 bits.**

3.5.4 Conversion depuis le format IEEE 754

Nous pouvons résumer le processus de conversion comme suit :

1. Isoler les différentes portions du code binaire : S, E, M.
2. Le bit de signe S donne le signe du nombre réel (0=positif, 1=négatif).
3. Convertir la portion « E » du code binaire vers le décimal et soustraire le biais (127 ou 1023) pour obtenir l'exposant du nombre réel normalisé.
4. Exprimer le nombre réel sous sa forme binaire normalisée : $1,_{<M>} \times 2^{<exposant>}$
5. Dénormaliser le nombre, c'est-à-dire ajuster la position de la virgule pour annuler l'exposant.
   Exemples : $1,110110 \times 2^4 = 11101,10$ $1,011011 \times 2^{-3} = 0,001011011$
6. Convertir le nombre binaire dénormalisé vers le décimal et l'affecter du signe correct.

1. **Quelle est la valeur décimale du float IEEE 754 suivant ?**
   11000010111010010010000000000000

3.5.5 Différences entre nombres réels et nombres en virgule flottante

Pas d'exercices.

3.5.6 Opérations arithmétiques en virgule flottante

Pas d'exercices.

3.6 Représentation des caractères

Pas d'exercices.

3.7 Représentation d'autres données

Pas d'exercices.

3.8 Représentation des instructions

3.8.1 Code machine

Voir 3.8.4.

3.8.2 Registre du CPU

Pas d'exercices.

3.8.3 Langage d'assemblage

Pas d'exercices.

3.8.4 Assembleur

1. **Déterminer le code machine binaire et hexa des instructions suivantes :**

   (Note : les constantes sont représentées en complément à 2 sur 16 bits.)

   a. `addi $t0, $t1, 50`

   b. `lw $t0, 25($s0)`

c. `sll $t6, $s6, 16`

d. `beq $s0, $s1, -25`

2. Déterminer l'instruction assembleur représentée par le code machine hexadécimal suivant :

(Note : les constantes sont représentées en complément à 2 sur 16 bits.)

a. `0x2108FFFF`

b. `0x1A800007`

c. `0x912A0000`

3.9 Détection et correction d'erreurs

3.9.1 Introduction

Pas d'exercices.

3.9.2 Bit de parité

La technique du *contrôle de parité* consiste à ajouter un bit supplémentaire, appelé *bit de parité*, aux données à protéger. La valeur du bit de parité est calculée de telle sorte que le nombre total de bits à 1 soit pair (parité paire) ou impair (parité impaire).

1. **Calculer le bit de parité paire/impaire pour chaque lettre du mot « LOVE » codé en ASCII sur 7 bits.**

2. **Les codes suivants sont-ils corrects si on utilise une parité paire ?**

   a. 10110011110001

   b. 01110001110010

3.9.3 Double parité

1. **Compléter le tableau de double parité pour le mot « WORLD » en ASCII sur 7 bits. On suppose que les lignes utilisent une parité paire (PP) et les colonnes une parité impaire (PI).**

Car.	b6	b5	b4	b3	b2	b1	b0	PP
W								
O								
R								
L								
D								
PI								

2. Déterminer le mot codé en ASCII sur 7 bits qui correspond au tableau de double parité suivant. On suppose que les lignes utilisent une parité impaire (PI) et les colonnes une parité paire (PP).

Car.	Code	b6	b5	b4	b3	b2	b1	b0	PI
		1	0	1	1	0	0	0	1
		1	0	0	0	1	0	1	0
		1	0	1	1	1	0	0	0
		1	0	0	1	1	0	1	1
		1	1	1	1	1	1	1	1
	PP	1	1	0	0	0	1	1	

3.9.4 Détection d'erreurs groupées (CRC)

Pas d'exercices.

3.9.5 Mémoire à code correcteur d'erreurs (ECC)

Pas d'exercices.

4 Circuits logiques

4.1 Algèbre de Boole et portes logiques

1. Déterminer les 2 formes normales correspondant à la table de vérité de l'opérateur NAND.

a	b	NAND	Formes normales
			$FN1 =$
			$FN2 =$

2. Déterminer les 2 formes normales correspondant à la table de vérité de l'opérateur NOR.

a	b	NOR	Formes normales
			$FN1 =$
			$FN2 =$

3. Déterminez les 2 formes normales de la fonction F() donnée par la table de vérité suivante.

A	B	C	F()
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

FN1 =

FN2 =

4. Déterminez les 2 formes normales de la fonction F() donnée par la table de vérité suivante.

A	B	C	F()
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

FN1 =

FN2 =

5. Considérons la fonction de 3 variables « majorité », c'est-à-dire une fonction F(a,b,c) dont la valeur vaut 1 lorsqu'au moins deux des trois variables a, b, c valent 1.

a. Déterminer la table de vérité de cette fonction.
b. Déterminer ses deux formes normales.
c. Dessiner le circuit générique (=non optimisé) qui correspond à chaque forme normale.

a	b	c	maj	Formes normales
				FN1 =
				FN2 =

Schéma FN1 fonction majorité

Schéma FN2 fonction majorité

6. Considérons la fonction de 3 variables « parité paire », c'est-à-dire une fonction F(a,b,c) dont la valeur vaut 1 lorsqu'un nombre impair des trois variables a, b, c valent 1.

a. Déterminer la table de vérité de cette fonction.
b. Déterminer la <u>première</u> forme normale de cette fonction.
c. Dessiner le circuit générique (=non optimisé) qui correspond à la première forme normale.

a	b	c	PP	Première forme normale
				$FN1 =$

Schéma FN1 fonction parité paire

7. Soit l'expression algébrique suivante : $F(a, b) = (a + b)(\overline{a} + \overline{b})$

a. Dessiner le circuit logique non simplifié.
b. Déterminer la table de vérité de *F*.
c. À quoi sert cette fonction ?

a	b	$a + b$	$\overline{a} + \overline{b}$	F

Utilité ?

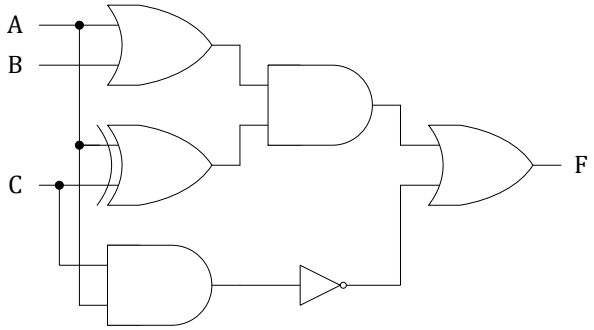
8. Soit l'expression algébrique suivante : $F(a, b) = \left(\left(a \oplus \overline{b}\right) + (\overline{a}b)\right) + \overline{b}$

a. Dessiner le circuit logique non simplifié.
b. Déterminer la table de vérité de *F*.
c. À quoi sert cette fonction ?

		T1	T2	T3	
a	b	$a \oplus \overline{b}$	$\overline{a}b$	T1+T2	F

Utilité ?

9. Considérons le circuit logique suivant :

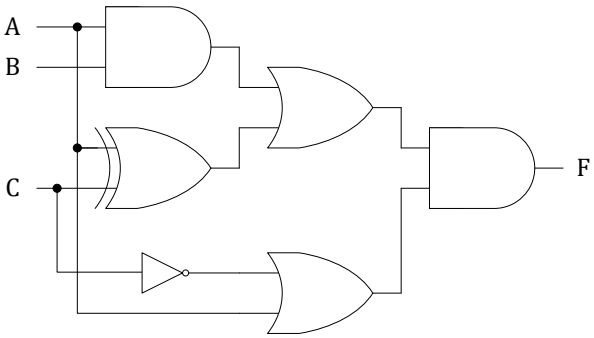


a. Déterminer l'expression algébrique de F.
b. Déterminer la table de vérité de F.

$F(A, B, C) =$

A	B	C						F()
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

10. Considérons le circuit logique suivant :

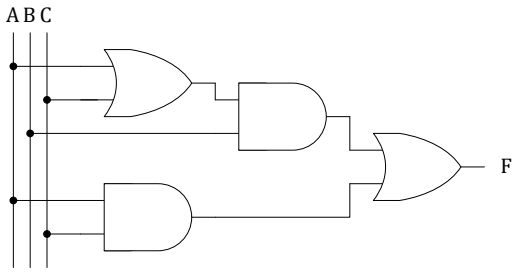


a. Déterminer l'expression algébrique de F.
b. Déterminer la table de vérité de F.

$F(A, B, C) =$

A	B	C						F()
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

11. Considérons le circuit logique suivant :



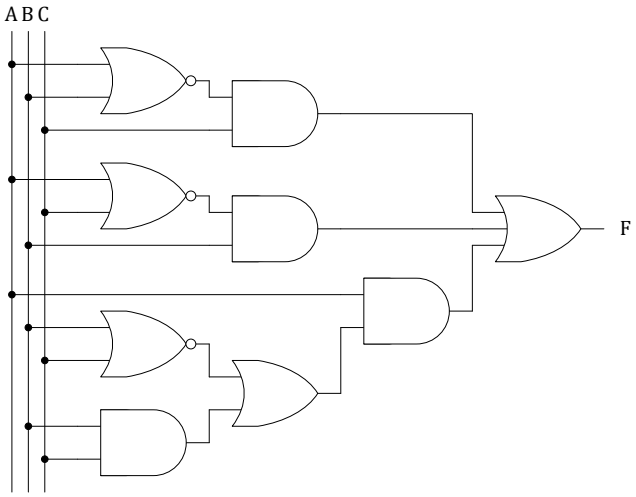
a. Déterminer l'expression algébrique de F.
b. Déterminer la table de vérité de F.
c. À quoi sert cette fonction ?

$F(A, B, C) =$

A	B	C				F()
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Utilité ?

12. Considérons le circuit logique suivant :



a. Déterminer l'expression algébrique de F.
b. Déterminer la table de vérité de F.
c. À quoi sert cette fonction ?

$F(A, B, C) =$

A	B	C									F()
0	0	0									
0	0	1									
0	1	0									
0	1	1									
1	0	0									
1	0	1									
1	1	0									
1	1	1									

Utilité ?

13. Implémenter l'opérateur ET en utilisant uniquement des portes OU et NON.

Suggestion : utiliser le théorème de De Morgan.

14. Implémenter l'opérateur NOR en utilisant uniquement des portes NAND.

Suggestion : utiliser le théorème de De Morgan.

15. Considérons une fonction logique de 4 variables dont la valeur vaut 1 lorsque (exactement) 2 des 4 variables d'entrée valent 1.

a. Construire la table de vérité de cette fonction.
b. Déterminer la forme normale la plus compacte.
c. Dessiner le circuit correspondant.
d. Combien faudrait-il de portes à 2 ou 1 entrée(s) pour implémenter cette forme normale ?

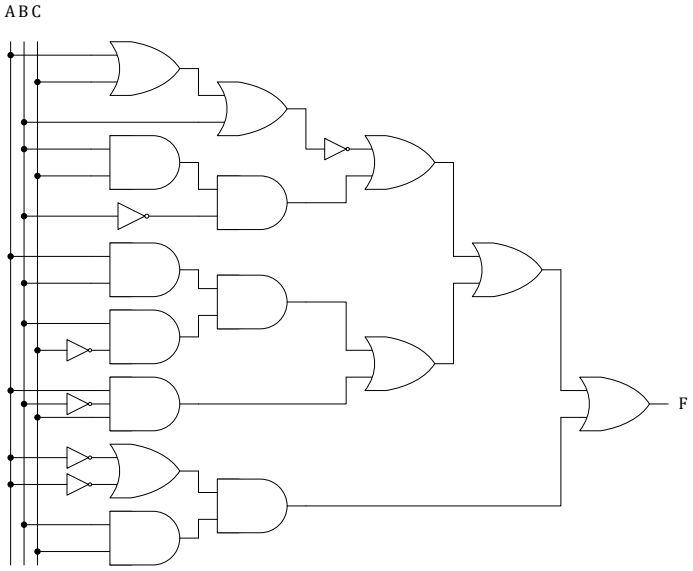
	A	B	C	D	F()
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

Forme normale la plus compacte :

Schéma logique de la forme normale

Nombre de portes à 2 ou 1 entrées ?

16. Considérons le circuit logique suivant :



a. Calculer l'expression algébrique qui correspond à ce circuit.
b. Calculer la table de vérité du circuit.
c. Quel est le rôle de ce circuit ?

F(A, B, C) =

A	B	C	F()
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Rôle du circuit ?

17. Déterminez la table de vérité des fonctions $F(A,B,C)$ et $G(A,B,C)$ dont on vous donne une des deux formes normales. En déduire, l'autre forme normale.

$F(A,B,C) = \bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + A.\bar{B}.\bar{C} + A.B.\bar{C}$ Forme normale : ……..

$G(A,B,C) = (\bar{A}+B+C).\ (\bar{A}+\bar{B}+C).\ (\bar{A}+B+\bar{C})$ Forme normale : ……..

A	B	C	F(A,B,C)	G(A,B,C)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Autre forme normale de chaque fonction :

$F(A,B,C) =$

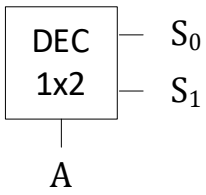
$G(A,B,C) =$

4.2 Circuits combinatoires

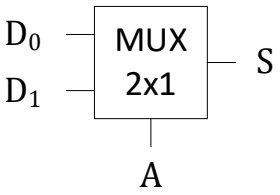
1. **Concevoir un circuit comparateur qui reçoit en entrée deux nombres A et B exprimé sur 1 bit possède trois sorties INF, EQU et SUP qui valent 1 respectivement si $A < B$, $A == B$ ou $A > B$.**

   Suggestion : commencer par établir la table de vérité du circuit.

2. **Réaliser un circuit décodeur 1×2.**



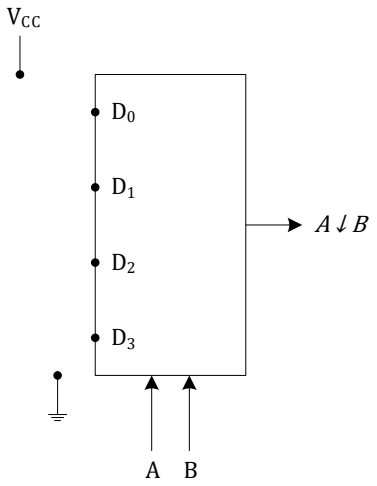
3. **Réaliser un circuit multiplexeur 2×1.**



4. Utiliser un multiplexeur pour implémenter les fonctions suivantes :

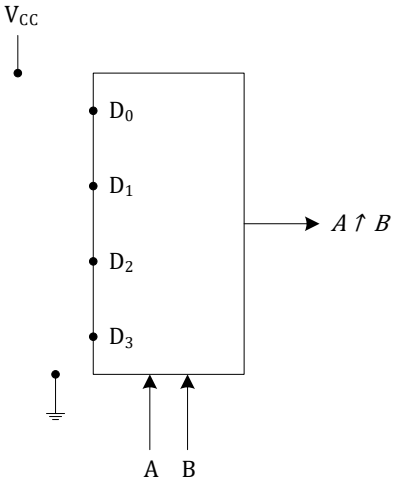
a. NOR

A	B	NOR
0	0	
0	1	
1	0	
1	1	



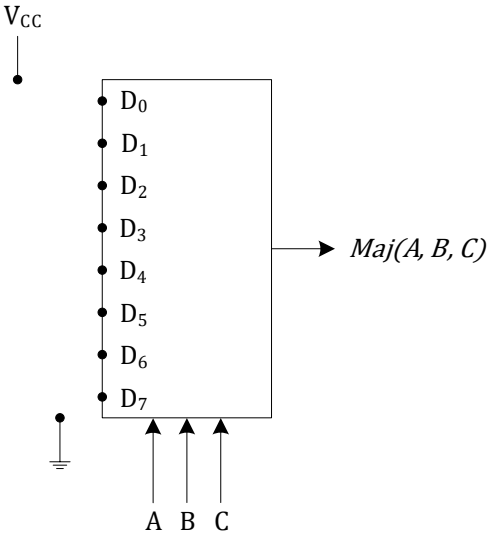
b. NAND

A	B	NAND
0	0	
0	1	
1	0	
1	1	



c. Majorité de 3 variables

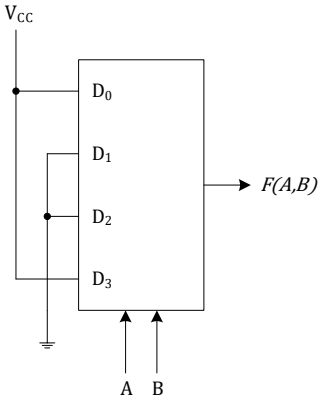
A	B	C	MAJ
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



5. Déterminer la table de vérité de la fonction $F(A, B)$ implémentée par le multiplexeur suivant :

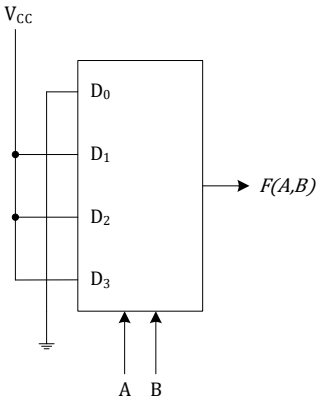
a. Fonction 1

A	B	???
0	0	
0	1	
1	0	
1	1	



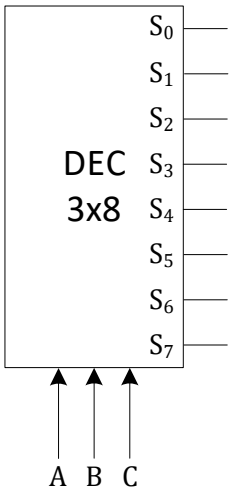
b. Fonction 2

A	B	???
0	0	
0	1	
1	0	
1	1	



6. En utilisant un décodeur 3×8 et des portes OU à 2 entrées, implémenter la fonction $F(A, B, C)$ dont la première forme normale est :

$$F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C + A.B.C$$



7. Au moyen de deux multiplexeurs et d'une porte NOR, implémenter un circuit qui détermine si une majorité, une égalité ou une minorité de ses 4 d'entrées valent 1.

Suggestion : commencer par établir la table de vérité du circuit.

Questions : quel sera le rôle de chaque multiplexeur et de la porte NOR ? Quelle sera la taille des multiplexeurs ?

	A	B	C	D	MA	EG	MI
0	0	0	0	0			
1	0	0	0	1			
2	0	0	1	0			
3	0	0	1	1			
4	0	1	0	0			
5	0	1	0	1			
6	0	1	1	0			
7	0	1	1	1			
8	1	0	0	0			
9	1	0	0	1			
10	1	0	1	0			
11	1	0	1	1			
12	1	1	0	0			
13	1	1	0	1			
14	1	1	1	0			
15	1	1	1	1			

8. Réaliser un additionneur complet en utilisant un décodeur 3×8 et des portes OU à 2 entrées.

Suggestion : établir la table de vérité du circuit. Quelles sont les entrées et les sorties ?

4.3 Unité arithmétique et logique (ALU)

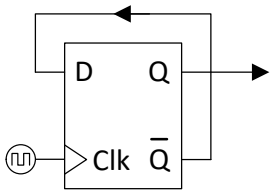
Pas d'exercices.

4.4 Circuits asynchrones et synchrones

Pas d'exercices.

4.5 Circuits séquentiels

1. Quelle est l'utilité du circuit suivant ?

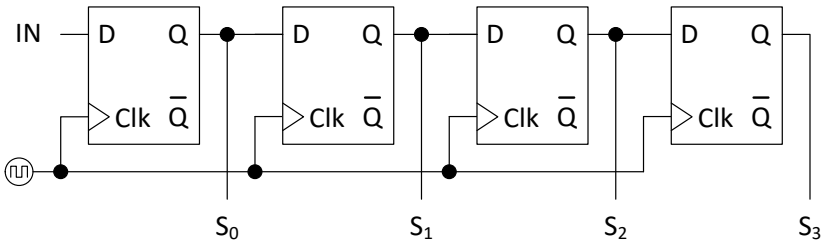


Suggestion : dessiner l'évolution dans le temps de la sortie Q en fonction du signal d'horloge.

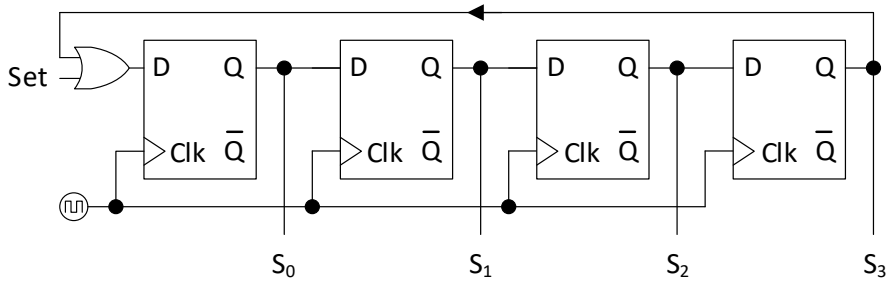
2. Construire un compteur d'impulsions sur 4 bits en utilisant des bascules D.

Suggestion : la solution est basée sur le circuit de l'exercice précédent.

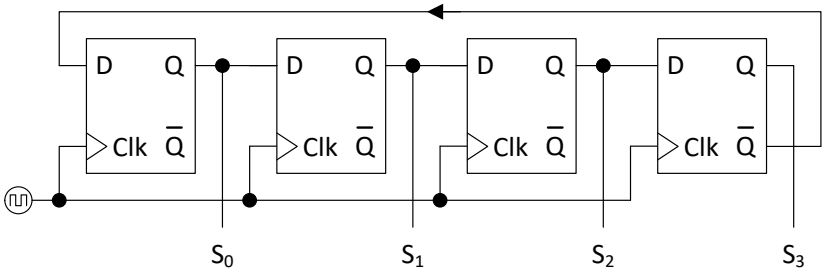
3. Quelle est l'utilité du circuit suivant ?



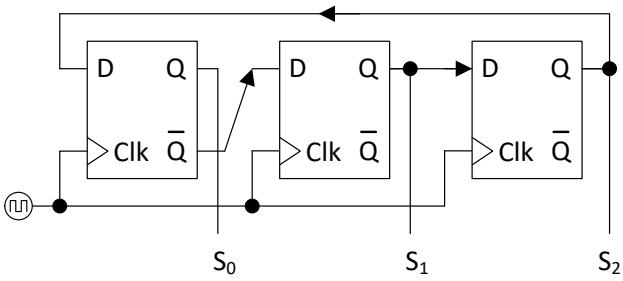
4. Quelle est l'utilité du circuit suivant ?



5. Quelle est l'utilité du circuit suivant ?

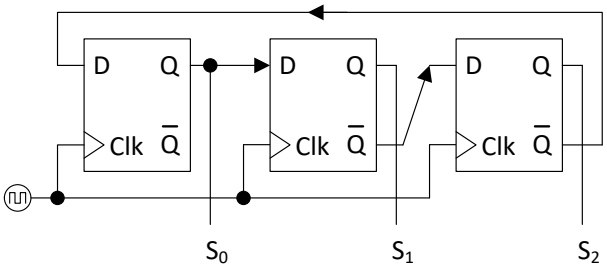


6. Si l'état initial (T_0) du circuit représenté ci-dessous est $S_0S_1S_2$ = 000, quel sera son état aux instants suivants T_1, T_2, T_3, … ?



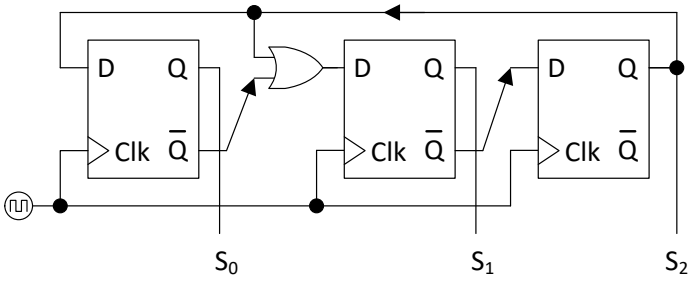
	Q_0	Q_1	Q_2
T+1 →			
T_0	**0**	**0**	**0**
T_1			
T_2			
T_3			
T_4			
T_5			
T_6			
T_7			

7. Si l'état initial (T_0) du circuit représenté ci-dessous est $S_0S_1S_2$ = 000, quel sera son état aux instants suivants T_1, T_2, T_3, … ?



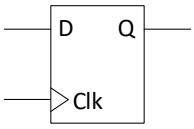
	Q_0	Q_1	Q_2
T+1 →			
T_0	**0**	**0**	**0**
T_1			
T_2			
T_3			
T_4			
T_5			
T_6			
T_7			

8. Si l'état initial (T_0) du circuit représenté ci-dessous est $S_0S_1S_2$ = 000, quel sera son état aux instants suivants T_1, T_2, T_3, … ?

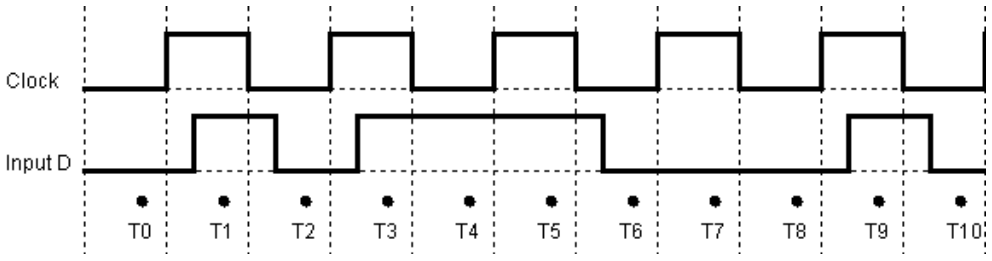


	Q_0	Q_1	Q_2
T+1 →			
T_0	**0**	**0**	**0**
T_1			
T_2			
T_3			
T_4			
T_5			
T_6			
T_7			

9. Considérons une bascule D dont le changement d'état est synchronisé sur une transition <u>montante</u> du signal d'horloge :



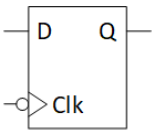
Le chronogramme ci-dessous indique les variations du signal d'horloge (Clock) et du signal à l'entrée D de la bascule au fil du temps. Un signal de niveau haut signifie 1 et un signal de niveau bas signifie 0.



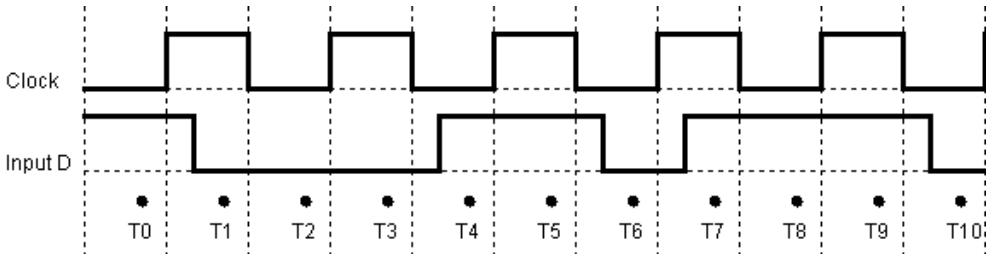
À l'instant T_0, la bascule se trouve dans l'état 0 ($Q = 0$). Déterminez l'état Q de la bascule (0 ou 1) aux instants $T_1, T_2, \dots, T_{10}$.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
0										

10. Considérons une bascule D dont le changement d'état est synchronisé sur une transition <u>descendante</u> du signal d'horloge :



Le chronogramme ci-dessous indique les variations du signal d'horloge (Clock) et du signal à l'entrée D de la bascule au fil du temps. Un signal de niveau haut signifie 1 et un signal de niveau bas signifie 0.



À l'instant T_0, la bascule se trouve dans l'état 1 ($Q = 1$). Déterminez l'état Q de la bascule (0 ou 1) aux instants $T_1, T_2, \dots, T_{10}$.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
1										

5 Mémoires

5.5 Mémoire cache

1. Soit une mémoire cache à <u>correspondance directe</u>.

- **La mémoire cache comporte 6 lignes.**
- **L'état initial de la mémoire est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		18	**27**	**3**	**11**	**22**	**29**	**15**	**25**	**13**	**1**
Ligne	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	30										
1	7										
2	-										
3	9										
4	-										
5	-										

2. Soit une mémoire cache à <u>complètement associative</u> qui utilise un algorithme de <u>remplacement circulaire</u>.

- **La mémoire cache comporte 6 lignes.**
- **L'état initial de la mémoire est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		11	**17**	**28**	**29**	**7**	**17**	**31**	**12**	**18**	**14**
Ligne	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	22										
1	12										
2	7										
3	55										
4	37										
5	-										

3. Soit une mémoire cache <u>associative par ensemble</u> qui utilise un algorithme de <u>remplacement circulaire</u>.

- **La mémoire cache comporte 4 ensembles (sets) de 2 lignes chacun.**
- **L'état initial de la mémoire cache est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		22	**23**	**20**	**21**	**17**	**16**	**20**	**25**	**29**	**17**
Set	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	4										
	-										
1	17										
	-										
2	18										
	-										
3	19										
	-										

6 Architectures avancées

Pas d'exercices sur le chapitre 6.

7 Programmation en assembleur

1. **Ecrire un programme qui détermine et affiche les N premiers multiples non nuls de l'entier M.**

   Les valeurs sur 32 bits de N et de M sont définies de manière statique dans la zone de données du programme et identifiées par les étiquettes « N » et « M ». Le programme affiche chaque multiple sur une ligne séparée. (Note : le code ascii du caractère de retour à la ligne est 10 ou `'\n'`.)

2. **Modifier le programme de l'exercice 1 de manière à calculer et stocker tous les résultats en mémoire avant de les afficher.**

   Pour stocker les résultats, le plus simple est de réserver une zone de mémoire de taille suffisante dans la zone de données du programme au moyen de la directive « `.word 0:n` ». Pour rappel, nous travaillons avec des entiers exprimés sur 32 bits. Chaque multiple calculé sera sauvegardé dans cette zone.

   Pour l'affichage final, on peut définir un sous-programme « PRINT » qui reçoit en entrée le nombre d'éléments à afficher ainsi que l'adresse de début de la zone de stockage.

3. **Etant donnée une chaîne de caractères délimitée par le caractère de fin de chaîne « \0 », compter et afficher combien de fois elle contient un caractère donné.**

   La chaîne de caractères « TEXT » et le caractère (byte) recherché « CHAR » sont définis de manière statique dans la zone de données du programme.

4. **Modifier le programme de l'exercice 3, de manière à compter le nombre de voyelles (« AEIOUY ») dans la chaîne de caractères.**

   Les voyelles sont définies de manière statique dans la zone de données du programme par une chaîne de caractères délimitée par un « \0 » et étiquetée « CHARS ».

   Remplacer la comparaison du caractère courant avec la lettre recherchée du programme 2 par un appel à un sous-programme « VALID » qui détermine si la lettre courante est un caractère valide (une voyelle dans notre cas).

   Lorsque le sous-programme « VALID » est appelé :

   - $a0 contient le code ascii du caractère à valider,
   - $a1 contient l'adresse de début de la chaîne des caractères autorisés.

   Le sous-programme retourne la valeur 0=non valide ou 1=valide dans le registre $v0.

5. **Modifier le programme de l'exercice 3 de manière à définir un sous-programme « NB_OCC » qui compte le nombre d'occurrences d'une lettre dans une chaîne et un programme principal « MAIN » qui l'utilise.**

   Lorsque le sous-programme « NB_OCC » est appelé :

   - $a0 contient l'adresse de début de la chaîne de caractères à analyser,
   - $a1 contient le code ascii du caractère recherché.

   Le sous-programme retourne le nombre d'occurrences trouvées dans le registre $v0.

6. **Modifier le programme de l'exercice 5 de manière à déterminer le nombre d'occurrences de chaque lettre de l'alphabet (« A » à « Z ») et à afficher le nombre d'occurrence d'une lettre uniquement lorsqu'il n'est pas nul.**

   La plage des caractères à analyser est définie en toute généralité en précisant le caractère de début « FIRST » et le caractère de fin « LAST » dans la zone de données du programme.

   La portion « MAIN » du programme doit itérer sur chaque caractère, appeler le sous-programme « NB_OCC » et afficher le nombre d'occurrences trouvées (s'il n'est pas nul) sous la forme d'un message « A -> 3 ».

   Suggestions :

   - Définir un sous-programme d'impression « PRINT » qui reçoit le nombre d'occurrences et le code ascii du caractère concerné.
   - L'appel système 4 permet d'afficher facilement la chaîne de caractères « -> ».

7. **Modifier le programme de l'exercice 6 de manière à ne pas tenir compte de la casse (minuscule / majuscule) d'une lettre.**

   Rappel : le code ascii d'une lettre minuscule s'obtient en ajoutant 32 à celui de la lettre majuscule correspondante. Exemple : « A » = 65 et « a » = 97.

SOLUTIONS

1. Histoire de l'informatique

1.1 Des nombres et des hommes

8. Donner le nom du système de numération et calculer la valeur décimale des nombres suivants :

a. 1010_2 binaire $= 1x2^3 + 1x2^1 = 8 + 2 = 10_{10}$

b. 42031_5 quinaire $= 4x5^4 + 2x5^3 + 3x5^1 + 1x5^0 = 4x625 + 2x125 + 3x5 + 1x1 = 2766_{10}$

c. 123_7 septénaire $= 1x7^2 + 2x7^1 + 3x7^0 = 1x49 + 2x7 + 3x1 = 66_{10}$

d. 777_8 octal $= 7x8^2 + 7x8^1 + 7x8^0 = 7x64 + 7x8 + 7x1 = 511_{10}$

e. 12345_{12} duodécimal $= 1x12^4 + 2x12^3 + 3x12^2 + 4x12^1 + 5x12^0$

$= 1x20736 + 2x1728 + 3x144 + 4x12 + 5x1 = 24677_{10}$

2. Présentation générale d'un ordinateur

Pas d'exercices sur le chapitre 2

3. Codage des informations

3.1 Systèmes de numération

3.1.1 Systèmes de numération utilisés en informatique

1. **Compter de 0 à 15 en binaire.**

   Base 2 -> symboles de la base {0, 1}.

   0000, 0001,
   0010, 0011,
   0100, 0101,
   0110, 0111,
   1000, 1001,
   1010, 1011,
   1100, 1101,
   1110, 1111.

2. **Compter de 0 à 15 en base 5.**

   Base 5 -> symboles de la base {0, 1, 2, 3, 4}.

   00, 01, 02, 03, 04,
   10, 11, 12, 13, 14,
   20, 21, 22, 23, 24,
   30.

3. **Compter de 0 à 15 en octal.**

   Base 8 -> symboles de la base {0, 1, 2, 3, 4, 5, 6, 7}.

   00, 01, 02, 03, 04, 05, 06, 07,
   10, 11, 12, 13, 14, 15, 16, 17.

3.1.2 Formule générale de définition d'un nombre

On peut définir de manière générale un nombre **N** exprimé en base **B** de la manière suivante :

$N_B = a_n B^n + a_{n-1} B^{n-1} + \dots + a_0 B^0 + a_{-1} B^{-1} + \dots + a_{-m} B^{-m} = \sum_{i=-m}^{n} a_i B^i$ **Forme expansée**

$N_B = a_n a_{n-1} \dots a_2 a_1 a_0 , a_{-1} \dots a_{-m}$ (ex: 1234,56) **Forme normale**

Où a_i est un des symboles de la base B.

Les exposants positifs (ou nul) correspondent à la partie entière du nombre et les exposants négatifs à sa partie fractionnaire.

3.1.3 Opérations arithmétiques

3.1.3.1 Addition et multiplication en décimal (base 10)

1. Calculez à la main la somme : 12.345.678 + 23.456.789.

```
  0011111  ← retenues reportées
  12345678
+ 23456789
  35802467
```

2. Calculez à la main le produit : 1234,56 x 678,9.

```
          1 2 3 4 , 5 6 (2 décimales)
x           6 7 8 , 9   (1 décimale)
          1 1 1 1 1 0 4 ← 9 x 123456 [9x6=54, 9x5+5=50, 9x4+5=41, …]
          9 8 7 6 4 8   ← 8 x 123456
        8 6 4 1 9 2     ← 7 x 123456
      7 4 0 7 3 6       ← 6 x 123456
      8 3 8 1 4 2,7 8 4 (1+2=3 décimales)
```

3.1.3.2 Addition et multiplication en base quelconque

1. Calculez à la main la somme binaire : $10110011_2 + 10111010_2$.

```
 10110010  ← retenues reportées
  10110011
+ 10111010
 101101101
```

2. Calculez à la main la somme en base 7 : $456321_7 + 126154_7$.

```
   011010  ← retenues reportées
   456321
 + 126154
   615505
```

3. Calculez à la main la somme en base 16 : $1A2B3C4D_{16} + A012D789_{16}$.

```
  00001101  ← retenues reportées
   1A2B3C4D
 + A012D789
   BA3E13D6
```

4. Considérons l'addition suivante : 17731 + 81847 = 99078.
Quelle doit être la base de numération utilisée pour que cette addition soit correcte ?
Dans cette base, quel sera le résultat de l'addition suivante 58735 + 25866 ?

Pour déterminer la base utilisée, il faut identifier une colonne pour laquelle un report a lieu.

```
  17731
+ 81847
  99078
```

Dans la 3ème colonne, 7 + 8 donne 0 comme résultat. Cela signifie que l'on travaille en base 15, car 7 + 8 = 15 = **15** + 0.

On peut à présent réaliser la deuxième addition proposée en base 15.

```
  00100  ← retenues reportées
   58735
 + 25866
   7E09B
```

5. Considérons l'addition suivante : 33825 + 64613 = 98238.
Quelle doit être la base de numération utilisée pour que cette addition soit correcte ?
Dans cette base, quel sera le résultat de l'addition suivante 27544 + 44772 ?

Pour déterminer la base utilisée, il faut identifier une colonne pour laquelle un report a lieu.

```
   33825
 + 64613
   98238
```

Dans la 3ème colonne, 8 + 6 donne 2 comme résultat. Cela signifie que l'on travaille en base 12, car 8 + 6 = 14 = **12** + 2.

On peut à présent réaliser la deuxième addition proposée en base 12.

```
  01100  ← retenues reportées
   27544
 + 44772
   700B6
```

3.2 Conversions entre bases

3.2.1 Nombres entiers et réels : base B vers décimal

1. Quel est le plus grand nombre de 3 chiffres en base 7 ?

$666_7 = 6 \times 7^2 + 6 \times 7^1 + 6 \times 7^0 = 6 \times 49 + 6 \times 7 + 6 = 342_{10} = 7^3-1$

2. Quel est le plus grand nombre de 4 chiffres en base 16 ?

$FFFF_{16} = 15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15 = 15 \times 4096 + 15 \times 256 + 15 \times 16 + 15 = 65535_{10} = 16^4-1$

3. Calculez la valeur décimale de nombres suivants :

Il suffit d'appliquer la formule générale et d'additionner les puissances successives de la base B.

a. $110011_2 = 2^5 + 2^4 + 2^1 + 2^0 = 32 + 16 + 2 + 1 = 51_{10}$

b. $1000110011101110_2 = 2^{15} + 2^{11} + 2^{10} + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1$
$= 32768 + 2048 + 1024 + 128 + 64 + 32 + 8 + 4 + 2 = 36078_{10}$

c. $210012_3 = 2x3^5 + 1x3^4 + 1x3^1 + 2x3^0 = 2x243 + 1x81 + 1x3 + 2x1 = 572_{10}$

d. $3210_4 = 3x4^3 + 2x4^2 + 1x4^1 = 3x64 + 2x16 + 1x4 = 228_{10}$

e. $5632_7 = 5x7^3 + 6x7^2 + 3x7^1 + 2x7^0 = 5x343 + 6x49 + 3x7 + 2x1 = 2032_{10}$

f. $007_8 = 7x8^0 = 7x1 = 7_{10}$

g. $428_8 \rightarrow$ impossible, car 8 n'est pas un symbole valide en base 8 !

h. $2014_8 = 2x8^3 + 1x8^1 + 4x8^0 = 2x512 + 1x8 + 4x1 = 1036_{10}$

i. $2014_{16} = 2x16^3 + 1x16^1 + 4x16^0 = 2x4096 + 1x16 + 4x1 = 8212_{10}$

j. $ADA_{16} = 10x16^2 + 13x16^1 + 10x16^0 = 2778_{10}$

k. $BEBE_{16} = 11x16^3 + 14x16^2 + 11x16^1 + 14x16^0 = 11x4096 + 14x256 + 11x16 + 14x1 = 48830_{10}$

l. $CAT_{16} \rightarrow$ impossible, car T n'est pas un symbole valide en base 16 !

m. $BAFFE_{16} = 11x16^4 + 10x16^3 + 15x16^2 + 15x16^1 + 14x16^0 = 765950_{10}$

4. Calculez la valeur décimale de nombres réels suivants :

Pour les nombres réels, on applique également la formule générale, mais on additionne les puissances positives et négatives de la base B.

a. $10111,1001_2$

$10111,1001_2 = 1x2^4 + 1x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-4}$
$= 16 + 4 + 2 + 1 + 0,5 + 0,0625$
$= 23,5625_{10}$

Réponse : $10111,1001_2 = 23,5625_{10}$

b. Convertir le nombre $123,456_8$ en base 10.

$123,456_8 = 1x8^2 + 2x8^1 + 3x8^0 + 4x8^{-1} + 5x8^{-2} + 6x8^{-3}$
$= 1x64 + 2x8 + 3x1 + 4x0,125 + 5x0,015625 + 6x0.001953125$
$= 83,58984375_{10}$

Réponse : $123,456_8 = 83,58984375_{10}$

c. Convertir le nombre CA,FE_{16} en base 10.

$CA,FE_{16} = 12x16^1 + 10x16^0 + 15x16^{-1} + 14x16^{-2}$
$= 12x16 + 10x1 + 15x0,0625 + 14x0,00390625$
$= 202,9921875_{10}$

Réponse : $CA,FE_{16} = 202, 9921875_{10}$

3.2.2 Nombres entiers : décimal vers base B

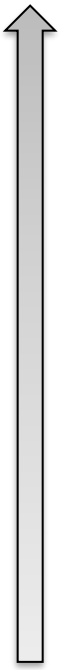
3.2.2.1 Division par la base

La technique de *division par la base* consiste à répéter les opérations suivantes :

4. Effectuer la division <u>entière</u> de N par la base B ($\frac{N}{B}$) et déterminez le quotient <u>entier</u> Q.
5. Le reste $R = N - Q \times B$ est le coefficient a_0 du terme $a_0 B^0$.
6. Recommencer au point 1 avec $N' = Q$ afin de déterminer les coefficients suivants $(a_1, a_2, \ldots)$ et cela tant que $Q \neq 0$.

1. Convertir 51529 en binaire.

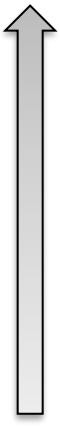
- 51529 / 2 → Q = 25764 R = a_0 = 1
- 25764 / 2 → Q = 12882 R = a_1 = 0
- 12882/ 2 → Q = 6441 R = a_2 = 0
- 6441/ 2 → Q = 3220 R = a_3 = 1
- 3220 / 2 → Q = 1610 R = a_4 = 0
- 1610 / 2 → Q = 805 R = a_5 = 0
- 805 / 2 → Q = 402 R = a_6 = 1
- 402 / 2 → Q = 201 R = a_7 = 0
- 201 / 2 → Q = 100 R = a_8 = 1
- 100 / 2 → Q = 50 R = a_9 = 0
- 50 / 2 → Q = 25 R = a_{10} = 0
- 25 / 2 → Q = 12 R = a_{11} = 1
- 12 / 2 → Q = 6 R = a_{12} = 0
- 6 / 2 → Q = 3 R = a_{13} = 0
- 3 / 2 → Q = 1 R = a_{14} = 1
- 1 / 2 → Q = 0 R = a_{15} = 1



Réponse : $51529_{10} = 1100100101001001_2$

2. Convertir 816 en binaire.

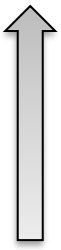
- 816 / 2 → Q = 408 R = a_0 = 0
- 408 / 2 → Q = 204 R = a_1 = 0
- 204 / 2 → Q = 102 R = a_2 = 0
- 102 / 2 → Q = 51 R = a_3 = 0
- 51 / 2 → Q = 25 R = a_4 = 1
- 25 / 2 → Q = 12 R = a_5 = 1
- 12 / 2 → Q = 6 R = a_6 = 0
- 6 / 2 → Q = 3 R = a_7 = 0
- 3 / 2 → Q = 1 R = a_8 = 1
- 1 / 2 → Q = 0 R = a_9 = 1



Réponse : $816_{10} = 1100110000_2$

3. Convertir 65536 en octal.

- 65536 / 8 → Q = 8192 — R = a_0 = 65536 - 8192x8 = 0
- 8192 / 8 → Q = 1024 — R = a_1 = 8192 - 1024x8 = 0
- 1024 / 8 → Q = 128 — R = a_2 = 1024 - 128x8 = 0
- 128 / 8 → Q = 16 — R = a_3 = 128 - 16x8 = 0
- 16 / 8 → Q = 2 — R = a_4 = 16 - 2x8 = 0
- 2 / 8 → Q = 0 — R = a_5 = 2 - 0x8 = 2



Réponse : $65536_{10} = 200000_8$

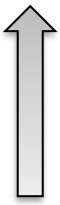
4. Convertir 47802 en hexadécimal.

- 47802 / 16 → Q = 2987 — R = a_0 = 47802 - 2987x16 = 10 (A)
- 2987 / 16 → Q = 186 — R = a_1 = 2987 - 186x16 = 11 (B)
- 186 / 16 → Q = 11 — R = a_2 = 186 - 11x16 = 10 (A)
- 11 / 16 → Q = 0 — R = a_3 = 11 - 0x16 = 11 (B)

Réponse : $47802_{10} = BABA_{16}$

5. Convertir 12345 en base 7.

- 12345 / 7 → Q = 1763 — R = a_0 = 12345 - 1763x7 = 4
- 1763 / 7 → Q = 251 — R = a_1 = 1763 - 251x7 = 6
- 251 / 7 → Q = 35 — R = a_2 = 251 - 35x7 = 6
- 35 / 7 → Q = 5 — R = a_3 = 35 - 5x7 = 0
- 5 / 7 → Q = 0 — R = a_4 = 5 - 0x7 = 5



Réponse : $12345_{10} = 50664_7$

3.2.2.2 Conversion en binaire par soustraction des puissances de 2

Soustraire les puissances décroissantes successives de 2 du nombre N à convertir, en commençant par la grande puissance de 2 qui est inférieure ou égale à N. On indique 1 à la position *n* du résultat lorsqu'il est possible de soustraire 2^n, 0 sinon

1. Convertir 227 en binaire

La plus grande puissance de 2 ≤ 227 est $2^7 = 128$ → le nombre binaire comporte 8 chiffres (puissances de 2 de 2^7 à 2^0).

- 2^7 (128) → Q = 1 a_7 R = 227 - **128** = 99
- 2^6 (64) → Q = 1 a_6 R = 99 - **64** = 35
- 2^5 (32) → Q = 1 a_5 R = 35 - **32** = 3
- 2^4 (16) → Q = 0 a_4 R = 3
- 2^3 (8) → Q = 0 a_3 R = 3
- 2^2 (4) → Q = 0 a_2 R = 3
- 2^1 (2) → Q = 1 a_1 R = 3 - **2** = 1
- 2^0 (1) → Q = 1 a_0 R = 1 - **1** = 0

Réponse : $227_{10} = 11100011_2$

2. Convertir 176 en binaire.

La plus grande puissance de 2 ≤ 176 est $2^7 = 128$ → le nombre binaire comporte 8 chiffres (puissances de 2 de 2^7 à 2^0).

- 2^7 (128) → Q = 1 a_7 R = 176 - **128** = 48
- 2^6 (64) → Q = 0 a_6 R = 48
- 2^5 (32) → Q = 1 a_5 R = 48 - **32** = 16
- 2^4 (16) → Q = 1 a_4 R = 16 - **16** = 0
- 2^3 (8) → Q = 0 a_3 R = 0
- 2^2 (4) → Q = 0 a_2 R = 0
- 2^1 (2) → Q = 0 a_1 R = 0
- 2^0 (1) → Q = 0 a_0 R = 0

Réponse : $176_{10} = 10110000_2$ Attention à ne pas oublier les 4 derniers chiffres qui valent 0 !

3. Convertir 2876 en binaire.

La plus grande puissance de 2 ≤ 2876 est 2^{11} = 2048 → le nombre binaire comporte 8 chiffres (puissances de 2 de 2^7 à 2^0).

- 2^{11} (2048) → Q = 1 a_{11} R = 2876 - **2048** = 828
- 2^{10} (1024) → Q = 0 a_{10} R = 828
- 2^9 (512) → Q = 1 a_9 R = 828 - **512** = 316
- 2^8 (256) → Q = 1 a_8 R = 316 - **256** = 60
- 2^7 (128) → Q = 0 a_7 R = 60
- 2^6 (64) → Q = 0 a_6 R = 60
- 2^5 (32) → Q = 1 a_5 R = 60 - **32** = 28
- 2^4 (16) → Q = 1 a_4 R = 28 - **16** = 12
- 2^3 (8) → Q = 1 a_3 R = 12 - **8** = 4
- 2^2 (4) → Q = 1 a_2 R = 4 - **4** =0
- 2^1 (2) → Q = 0 a_1 R = 0
- 2^0 (1) → Q = 0 a_0 R = 0

Réponse : $2876_{10} = 101100111100_2$ Attention à ne pas oublier les 2 derniers chiffres qui valent 0 !

3.2.3 Nombres réels : décimal vers base B

Soit N = X,Y un nombre réel. On souhaite convertir la partie fractionnaire Y vers une base B.
La ***méthode de multiplication par la base*** consiste à répéter les opérations suivantes :

1. On multiplie Y par la base B.
2. On obtient un nombre réel N' = B x Y = X',Y'.
3. X' est le premier chiffre de la partie fractionnaire en base B.
4. On répète le processus à l'étape 1 en utilisant Y', jusqu'à obtenir Y'=0 ou avoir atteint la précision maximale disponible.

Attention, une partie fractionnaire finie dans une base donnée peut conduire à une partie fractionnaire qui est illimitée dans une autre base !

1. Convertir $5{,}796875_{10}$ en binaire.

Partie entière : $5_{10} = 101_2$
Partie fractionnaire : $0{,}796875_{10}$

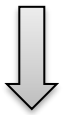
- 0,796875 x 2 = 1,59375 → 1 + 0,59375
- 0,59375 x 2 = 1,1875 → 1 + 0,1875
- 0,1875 x 2 = 0,375 → 0 + 0,375
- 0,375 x 2 = 0,75 → 0 + 0,75
- 0,75 x 2 = 1,5 → 1 + 0,5
- 0,5 x 2 = 1 → 1 + 0

Réponse : $5{,}796875_{10} = 101{,}110011_2$

2. Convertir $59{,}85546875_{10}$ en octal.

Partie entière : $59_{10} = 73_8$
Partie fractionnaire : $0{,}85546875_{10}$

- $0{,}85546875 \times 8 = 6{,}84375 \rightarrow 6 + 0{,}84375$
- $0{,}84375 \times 8 = 6{,}75 \rightarrow 6 + 0{,}75$
- $0{,}75 \times 8 = 6 \rightarrow 6 + 0$



Réponse : $59{,}85546875_{10} = 73{,}666_8$

3. Convertir $15{,}0859375_{10}$ en hexadécimal.

Partie entière : $15_{10} = F_{16}$
Partie fractionnaire : $0{,}0859375_{10}$

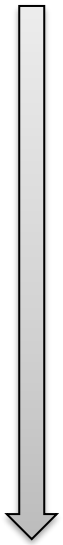
- $0{,}0859375 \times 16 = 1{,}375 \rightarrow 1 + 0{,}375$
- $0{,}375 \times 16 = 6 \rightarrow 6 + 0$

Réponse : $15{,}0859375_{10} = F{,}16_{16}$

4. Convertir $12{,}34_{10}$ en binaire.

Partie entière : $12_{10} = 1100_2$
Partie fractionnaire : $0{,}34_{10}$

- $0{,}34 \times 2 = 0{,}68 \rightarrow 0 + 0{,}68$
- $0{,}68 \times 2 = 1{,}36 \rightarrow 1 + 0{,}36$
- $0{,}36 \times 2 = 0{,}72 \rightarrow 0 + 0{,}72$
- $0{,}72 \times 2 = 1{,}44 \rightarrow 1 + 0{,}44$
- $0{,}44 \times 2 = 0{,}88 \rightarrow 0 + 0{,}88$
- $0{,}88 \times 2 = 1{,}76 \rightarrow 1 + 0{,}76$
- $0{,}76 \times 2 = 1{,}52 \rightarrow 1 + 0{,}52$
- $0{,}52 \times 2 = 1{,}04 \rightarrow 1 + 0{,}04$
- $0{,}04 \times 2 = 0{,}08 \rightarrow 0 + 0{,}08$
- $0{,}08 \times 2 = 0{,}16 \rightarrow 0 + 0{,}16$
- $0{,}16 \times 2 = 0{,}32 \rightarrow 0 + 0{,}32$
- $0{,}32 \times 2 = 0{,}64 \rightarrow 0 + 0{,}64$
- $0{,}64 \times 2 = 1{,}28 \rightarrow 1 + 0{,}28$
- Etc.



Réponse : $12{,}34_{10} = 1100{,}0101011100001..._2$

3.2.4 Bases qui sont des puissances de 2

Pour convertir **d'une base B = 2^n vers le binaire**, on transforme chaque chiffre de la base B en sa représentation en *n* chiffres binaires ou **bits** (= *binary digits*) et on concatène les différents morceaux.

1. Convertir $ABCD_{16}$ en binaire.

- $A_{16} \rightarrow 1010_2$ (= 10_{10})
- $B_{16} \rightarrow 1011_2$ (= 11_{10})
- $C_{16} \rightarrow 1100_2$ (= 12_{10})
- $D_{16} \rightarrow 1101_2$ (= 13_{10})

Réponse : $ABCD_{16} = 1010\ 1011\ 1100\ 1101_2 = 1010101111001101_2$

2. Convertir 123456_8 en binaire.

- $1_8 \rightarrow 001_2$
- $2_8 \rightarrow 010_2$
- $3_8 \rightarrow 011_2$
- $4_8 \rightarrow 100_2$
- $5_8 \rightarrow 101_2$
- $6_8 \rightarrow 110_2$

Réponse : $123456_8 = 001\ 010\ 011\ 100\ 101\ 110_2 = 1010011100101110_2$

Pour convertir **du binaire vers une base B = 2^n**, on découpe le nombre binaire en tronçons de *n* bits en allant de la droite vers la gauche. Chaque tronçon correspond à un chiffre en base B. Il faut éventuellement compléter le dernier tronçon de gauche avec des 0.

3. Convertir 1011111010111110_2 en hexadécimal.

$16 = 2^4 \rightarrow$ 4 bits par chiffre hexadécimal.

$1011111010111110_2 \rightarrow$	1011	1110	1011	1110_2
	B (11)	E (14)	B (11)	E (14)

Réponse : $1011111010111110_2 = BEBE_{16}$

4. Convertir 10111000110101_2 en octal.

$8 = 2^3 \rightarrow$ 3 bits par chiffre hexadécimal.

$10111000110101_2 \rightarrow$	010	111	000	110	101_2
	2	7	0	6	5

Réponse : $10111000110101_2 = 27065_8$

Pour convertir **entre deux bases B et B'** qui sont des puissances de 2, on procède en deux étapes :
1. Conversion de la base B vers le binaire.
2. Conversion du binaire vers la base B'.

5. **Convertir $4F16_{16}$ en octal.**

   $4F16_{16} \rightarrow 0100\ 1111\ 0001\ 0110_2 \rightarrow \cancel{0}\ 100\ 111\ 100\ 010\ 110_2 \rightarrow 47426_8$

6. **Convertir 777_8 en hexadécimal.**

   $777_8 \rightarrow 111\ 111\ 111_2 \rightarrow 0001\ 1111\ 1111_2 \rightarrow 1FF_{16}$

3.3 Représentation interne des informations

Pas d'exercices

3.4 Représentation des nombres entiers

3.4.1 Nombres naturels

1. **Combien de nombres naturels peut-on représenter sur 32 bits ?**

   Réponses : 2^{32} = 4294967296 nombres (les naturels de 0 à 4294967295).

2. **Quelle est la valeur du plus grand nombre naturel représentable sur 16 bits ?**

   Réponse : 2^{16}-1 = 65535

3. **Si je souhaite représenter les nombres naturels de 0 à 63, de combien de bits ai-je besoin ?**

   Réponse : 64 nombres → Il faut 6 bits, car 2^6 = 64.

4. **Calculez la représentation binaire sur 16 bits des nombres décimaux suivants :**

   a. $11111_{10} \rightarrow$ 0010 1011 0110 0111

   b. $54321_{10} \rightarrow$ 1101 0100 0011 0001

   c. $66666_{10} \rightarrow$ 66666 > 2^{16}-1 = 65535 → dépassement de capacité ! Il faudrait 17 bits.

   Note : il faut convertir du décimal vers le binaire par la technique du resserrement d'intervalle ou de la division par la base, puis aligner la réponse obtenue à droite dans le mot mémoire de 16 bits, en complétant les bits de gauche inutilisés par des zéros.

5. **Calculez la représentation binaire sur 16 bits des nombres hexadécimaux suivants :**

   a. $1111_{16} \rightarrow$ 0001 0001 0001 0001

   b. $ABC_{16} \rightarrow$ 0000 1010 1011 1100

   c. $R2D2_{16} \rightarrow$ impossible, R n'est pas un symbole valide en base 16 !

d. $A9F2_{16} \rightarrow 1010\ 1001\ 1111\ 0010$

Note : on peut convertir directement chaque chiffre hexadécimal vers sa représentation binaire sur 4 bits. L'alignement s'effectue également à droite dans le mot mémoire.

6. Quelle est la caractéristique de la représentation interne binaire de tous les nombres naturels qui sont des multiples de 8 (0, 8, 16, 24, …, 800, …, 8000, …, 36504, …) ?

Réponse : les 3 bits de poids faible, c'est-à-dire les 3 bits situés à droite, valent toujours 0.
Cela est dû au fait que $8 = 2^3 = 1000_2$ Tous les multiples de 8 sont donc des multiples de 1000_2, ce qui donnera toujours 0 dans les 3 derniers bits.

7. En utilisant la représentation binaire, déterminez le plus grand multiple de 32 représentable sur 16 bits ?

$32 = 2^5 = 100000_2 \rightarrow$ Tous les multiples de 32 auront les 5 derniers bits à 0. Le plus grand multiple de 32 est donc le nombre binaire dont tous les bits sont à 1 sauf les 5 derniers.

Réponse : Le plus grand multiple de 32 est $1111\ 1111\ 1110\ 0000_2 = 65504_{10}$

8. Représentez le nombre 102_{10} en binaire sur 8 bits.

$102_{10} = 0110\ 0110_2$

a. Que se passe-t-il si on décale tous les bits de cette représentation d'une position vers la droite ?

$00110011_2 = 51_{10} \rightarrow$ division par 2

b. Que se passe-t-il si on décale tous les bits de cette représentation d'une position vers la gauche ?

$11001100_2 = 204_{10} \rightarrow$ multiplication par 2

9. Si on dispose de la représentation binaire d'un nombre décimal, comment peut-on obtenir facilement le quotient et le reste (modulo) de la division de ce nombre par 2, 4, 8, 16, … ?

Réponse : les nombres 2, 4, 8, 16, … correspondent aux puissances de 2. Pour une puissance 2^n, le reste de la division $\frac{N}{2^n}$ correspond aux *n* derniers bits de ce nombre. Le quotient est le nombre constitué par les autres bits (c.-à-d. tous sauf les n derniers).

Exemple : calculez le quotient et le reste de la division par 16 du nombre 13322_{10}.

$16_{10} = 2^4 \rightarrow$ Q = les 12 premiers bits, R = les 4 derniers bits.

$13322_{10} =$ $0011\ 0100\ 0000$ 1010_2

Quotient = $0011\ 0100\ 0000_2 = 832_{10}$ Reste = $1010_2 = 10_{10}$

10. Considérons un circuit mémoire de 64K (64x1024 octets). Pour que chaque octet soit accessible au moyen d'une adresse mémoire unique, de combien d'adresses avons-nous besoin ? Quelle sera la taille (minimale) en bits d'une adresse ?

Réponse : 64x1024 = 65536 = 2^{16} → adresses sur 16 bits.

11. On décide de diviser cette mémoire en 64 blocs de 1024 octets qui se suivent en mémoire. Les blocs sont numérotés de 0 à 63, les octets au sein de chaque bloc de 0 à 1023. Le bloc 0 commence à l'adresse 0, le bloc 1 à l'adresse 1024, le bloc 2 à l'adresse 2048, etc.

a. Quelle est l'adresse de l'octet 723 du bloc 45 ?

Réponse : 45x1024 + 723 = 46803

b. En partant de la représentation binaire d'une adresse, comment peut-on facilement obtenir le numéro du bloc et le numéro de l'octet au sein du bloc (offset) ?

Réponse : il y a 64 (= 2^6) blocs de 1024 (= 2^{10}) octets. Le numéro du bloc est le quotient de la division entière de l'adresse par 1024, il s'obtient directement en gardant les 6 premiers bits (15 - 10) de l'adresse. Le numéro de l'octet est le reste de la division par 1024, il s'obtient en gardant les 10 derniers bits de l'adresse.

Exemple : considérons l'octet dont l'adresse vaut 13322_{10} = $\underline{0011\ 0100}\ \underline{0000\ 1010}_2$

Bloc = $0011\ 01_2 = 13_{10}$ Offset = $00\ 0000\ 1010_2 = 10_{10}$

Preuve : 13x1024 + 10 = 13322

12. Dans le système RGB (Red/Green/Blue), une couleur est représentée par un triplet de nombres compris entre 0 et 255 (8 bits) qui indiquent l'intensité de chaque couleur de base pour obtenir la couleur composée. Ainsi, le triplet (153, 51, 255) représente une nuance de mauve. On peut également représenter le code couleur par un nombre hexadécimal de 6 chiffres (2 chiffres hexa par composante). Pour notre nuance de mauve, cela donne 0x9933FF (=$9933FF_{16}$).

a. Calculer le code hexadécimal de la couleur RGB(57, 122, 206).

$57_{10} = 39_{16}$ / $122_{10} = 7A_{16}$ / $206_{10} = CE_{16}$ → 0x397ACE

b. Calculer les composantes RGB décimales de la couleur 0x12B022.

$12_{16} = 18_{10}$ / $B0_{16} = 176_{10}$ / $22_{16} = 34_{10}$ → RGB(18, 176, 34)

c. Calculer le code hexadécimal et les composantes RGB décimales de la couleur représentée par le nombre décimal 15971855_{10}.

$15971855_{10} = F3B60F_{16}$

$F3_{16} = 243_{10}$ / $B6_{16} = 182_{10}$ / $0F_{16} = 15_{10}$

→ Couleur 0xF3B60F ou RGB(243, 182, 15)

3.4.2 Stockage en mémoire

3.4.2.1 Alignement des mots mémoires

Pas d'exercices

3.4.2.2 Représentations petit-boutiste et gros-boutiste

1. **Considérons le nombre entier sur 32 bits : 2544829550.**
   Calculez les représentations hexadécimales gros-boutiste et petit-boutiste de ce nombre.

   Valeur hexadécimale : 97AF046E

   Représentation en mémoire

Adresse mémoire →	N	N+1	N+2	N+3
Représentation gros-boutiste	97	AF	04	6E
Représentation petit-boutiste	6E	04	AF	97

2. **Considérons le nombre entier sur 32 bits : 935693578.**
   Calculez les représentations hexadécimales gros-boutiste et petit-boutiste de ce nombre.

   Valeur hexadécimale : 37C58D0A

   Représentation en mémoire

Adresse mémoire →	N	N+1	N+2	N+3
Représentation gros-boutiste	37	C5	8D	0A
Représentation petit-boutiste	0A	8D	C5	37

3.4.3 Nombres entiers relatifs

3.4.3.1 Signe et valeur absolue

Le bit le plus significatif (c.-à-d. le plus à gauche) représente le *signe*. Les autres bits sont utilisés pour représenter la *valeur absolue* du nombre. Par convention, le signe + est représenté par la valeur 0 et le signe - par la valeur 1.

1. Calculer la représentation binaire sur 8 bits en signe et valeur absolue des nombres suivants :

a. 108 **0**110 1100_2 signe positif → premier bit = 0
b. -108 **1**110 1100_2 signe négatif → premier bit = 1
c. -127 +127 = **0**$111\ 111_2$ → -127 = **1**$111\ 1111_2$
d. -1 +1 = **0**$000\ 0001_2$ → -1 = **1**$000\ 0001_2$
e. -128 impossible sur 8 bits [-127, +127]

2. Calculer la valeur décimale des nombres binaires suivants, sachant qu'ils sont représentés en signe et valeur absolue :

a. **0**1010101 signe positif / $1010101_2 = 85_{10}$ → +85
b. **1**1010101 signe négatif / $1010101_2 = 85_{10}$ → -85
c. **1**1100011 signe négatif / $1100011_2 = 99_{10}$ → -99

3.4.3.2 Complément à 2

Le *complément à 2*, ou *complément arithmétique*, s'obtient en ajoutant +1 à la valeur du complément à 1. **Méthode rapide** : on recopie tous les bits du nombre positif en commençant par la droite jusqu'au premier 1 inclus, après quoi on inverse tous les bits qui suivent.

1. Calculer la représentation binaire sur 8 bits en complément à 2 des nombres suivants :

a. 108 **0**110 1100_2 signe positif → premier bit = 0
b. -108 **1**001 0100_2 signe négatif → complément à 2 → premier bit = 1
c. -127 +127 = **0**$111\ 1111_2$ → -127 = **1**$000\ 0001_2$
d. -1 +1 = **0**$000\ 0001_2$ → -1 = **1**$111\ 1111_2$
e. -128 **1**0000000 par définition !

2. Calculer la valeur décimale des nombres binaires suivants, sachant qu'ils sont représentés en complément à 2 :

a. **0**1010101 signe positif / $01010101_2 = 85_{10}$ → +85
b. **1**1010101 signe négatif, complément à 2 / $00101011_2 = 43_{10}$ → -43
c. **1**1100011 signe négatif, complément à 2 / $00011101_2 = 29_{10}$ → -29

3. Exercice récapitulatif 1 – calculer la représentation binaire sur 8 bits des nombres suivants :

a. 91 → 01011011
b. -91 en signe et valeur absolue → **1**1011011
c. -91 en complément à 1 → **1**0100100
d. -91 en complément à 2 → **1**0100101

1. Exercice récapitulatif 2 – calculer la valeur décimale du nombre représenté par 10101010 si :

a. entier non signé → +170
b. signe et valeur absolue → -42
c. complément à 1 → -85
d. complément à 2 → -86

Pour soustraire un nombre, il suffit juste d'additionner son complément. On additionne tous les chiffres et on laisse tomber le report final éventuel. Un *dépassement de capacité* est détecté par le fait que la retenue générée sur le bit de signe est différente de celle générée sur le bit juste avant.

2. Réaliser les opérations suivantes en arithmétique binaire sur 8 bits en complément à 2. Vérifier si le résultat binaire obtenu correspond bien à la valeur attendue.

a. 5 + 122 = 127

```
        00000000
   5     00000101
+122    +01111010
 127     01111111 = +127
```

b. 122 + 64 = 186

```
        01000000
 122     01111010
+ 64    +01000000
 186     10111010 = −70  → dépassement de capacité !
```

c. 64 - 96 = -32

```
        00000000
  64     01000000
 -96    +10100000
 -32     11100000 = −32
```

d. -96 - 32 = -128

```
        11100000
 -96     10100000
 -32    +11100000
-128     10000000 = −128
```

e. -96 - 64 = -160

```
        10000000
 -96     10100000
 -64    +11000000
-160     01100000 = +96  → dépassement de capacité !
```

3.4.3.3 Extension de signe

Extension de signe : pour « agrandir » un nombre entier signé codé en complément à 2 (ou à 1) sur 8 bits vers son équivalent sur 16 bits (32 bits ou 64 bits), il convient de *recopier le bit de signe* dans les octets ajoutés à gauche.

1. Étendre les nombres binaires suivants de 8 à 16 bits :

a. 01110111 → 00000000 01110111
b. 10011100 → 11111111 10011100

3.5 Représentation des nombres réels

3.5.1 Virgule fixe

Pas d'exercices.

3.5.2 Virgule flottante

La représentation des nombres réels en virgule flottante telle que définie par le standard IEEE 754 consiste à stocker en mémoire la forme normalisée d'un nombre réel binaire de la manière suivante :

SM	EXPOSANT	MANTISSE

où

- SM est le signe de la mantisse. Par convention, 0 signifie + (positif) et 1 signifie – (négatif).
- L'exposant utilise la notation biaisée.
- Seule la partie fractionnaire de la mantisse normalisée est stockée (la partie entière vaut 1).

Rappel : la *forme normalisée d'un nombre réel binaire* définie par le standard IEEE 754 s'obtient en ajustant l'exposant de telle sorte que la partie entière de la mantisse soit égale à 1. La seule exception est le nombre 0 qui est représenté par un mot mémoire dont tous les bits sont à 0.

Précision	SM	EXPOSANT / BIAIS		MANTISSE
Simple (32 bits)	1 bit	8 bits / +127	[-127, +128]	23 bits
Double (64 bits)	1 bit	11 bits / +1023	[-1023, +1024]	52 bits
Étendue (80 bits)	1 bit	15 bits / +16383	[-16383, +16384]	64 bits

3.5.3 Conversion vers le format IEEE 754

Nous pouvons résumer le processus de conversion comme suit :

1. Identifier le *signe* du nombre à convertir.
2. Convertir la *partie entière* du nombre réel décimal vers son équivalent binaire.
3. Convertir la *partie fractionnaire* du nombre réel décimal vers son équivalent binaire.
4. *Normaliser* le nombre réel binaire obtenu aux étapes 2 et 3, ce qui nous donne la valeur de l'*exposant*.
5. Calculer la valeur décimale biaisée de l'exposant en lui ajoutant 127 (simple précision).
6. Convertir la valeur décimale biaisée de l'exposant vers son équivalent binaire sur 8 bits.
7. Assembler les différents morceaux pour obtenir la représentation finale en virgule flottante. Pour rappel, on ne stocke pas le bit qui correspond à la partie entière de la mantisse. Si la partie fractionnaire de la mantisse comporte moins de 23 bits (simple précision), on complète à droite avec des zéros

1. Convertir $456{,}8125_{10}$ en float IEEE 754 sur 32 bits.

Signe positif : 0

Partie entière : $456_{10} = 111001000_2$ (= 256+128+64+8)

Partie fractionnaire : $0{,}8125_{10}$ -> $0{,}1101_2$

Nombre binaire : $111001000{,}1101_2$

Nombre binaire normalisé : $1{,}110010001101 \times 2^8$

Exposant = 8

Exposant biaisé : 8 + 127 = $135_{10} = 10000111_2$ (= 128+4+2+1)

Mantisse à stocker : 110010001101 (on ne stocke pas la partie entière qui vaut 1)

Réponse : **0 10000111 11001000110100000000000**

2. Convertir $-123{,}321_{10}$ en float IEEE 754 sur 32 bits.

Signe négatif : 1

Partie entière : $123_{10} = 1111011_2$ (= 64+32+16+8+2+1)

Partie fractionnaire : $0{,}321_{10}$ -> $0{,}01010010001011010_2$ (*)

Nombre binaire : $1111011{,}01010010001011010_2$

Nombre binaire normalisé : $1{,}11101101010010001011010 \times 2^6$

Exposant = 6

Exposant biaisé : 6 + 127 = $133_{10} = 10000101_2$ (= 128+4+1)

Mantisse à stocker : 11101101010010001011010 (on ne stocke pas la partie entière qui vaut 1)

Réponse : **1 10000101 11101101010010001011010**

(*) le nombre de décimales est limité à 17, car la mantisse ne comporte que 23 bits dont 6 seront par la partie entière (7 bits). ArchiApp n'affiche pas la dernière décimale, car elle vaut 0.

3.5.4 Conversion depuis le format IEEE 754

Nous pouvons résumer le processus de conversion comme suit :

1. Isoler les différentes portions du code binaire : S, E, M.
2. Le bit de signe S donne le signe du nombre réel (0=positif, 1=négatif).
3. Convertir la portion « E » du code binaire vers le décimal et soustraire le biais (127 ou 1023) pour obtenir l'exposant du nombre réel normalisé.
4. Exprimer le nombre réel sous sa forme binaire normalisée : $1,<M> \times 2^{<exposant>}$
5. Dénormaliser le nombre, c'est-à-dire ajuster la position de la virgule pour annuler l'exposant.
   Exemples : $1,110110 \times 2^4 = 11101,10$ $1,011011 \times 2^{-3} = 0,001011011$
6. Convertir le nombre binaire dénormalisé vers le décimal et l'affecter du signe correct.

1. Quelle est la valeur décimale du float IEEE 754 suivant ?

1 10000101 11010010010000000000000

Signe : 1 → nombre négatif

Exposant biaisé : $10000101_2 = 133_{10}$

Exposant effectif : 133 - 127 = 6

Mantisse : 1,1101001001

Nombre réel binaire : $-1,1101001001 \times 2^6 = -1110100,1001_2$

Nombre réel décimal : $-116,5625_{10}$

3.5.5 Différences entre nombres réels et nombres en virgule flottante

Pas d'exercices.

3.5.6 Opérations arithmétiques en virgule flottante

Pas d'exercices.

3.6 Représentation des caractères

Pas d'exercices.

3.7 Représentation d'autres données

Pas d'exercices.

3.8 Représentation des instructions

3.8.1 Code machine

Voir 3.8.4.

3.8.2 Registre du CPU

Pas d'exercices.

3.8.3 Langage d'assemblage

Pas d'exercices.

3.8.4 Assembleur

1. **Déterminer le code machine binaire et hexadécimal des instructions suivantes :**

   (Note : les constantes sont représentées en complément à 2 sur 16 bits.)

   a. `addi $t0, $t1, 50`

   ```
   [FORMAT I]
   OP:6-- RS:5- RT:5- CST:16----------
   8      9     8     50
   001000 01001 01000 0000000000110010
   0010 0001 0010 1000 0000 0000 0011 0010
   HEXA = 0x21280032
   ```

   b. `lw $t0, 25($s0)`

   ```
   [FORMAT I]
   OP:6-- RS:5- RT:5- CST:16----------
   35     16    8     25
   100011 10000 01000 0000000000011001
   1000 1110 0000 1000 0000 0000 0001 1001
   HEXA = 0x8E080019
   ```

   c. `sll $t6, $s6, 16`

   ```
   [FORMAT R]
   OP:6-- RS:5- RT:5- RD:5- SHT:5 FCT:6-
   0      0     22    14    16     0
   000000 00000 10110 01110 10000 000000
   0000 0000 0001 0110 0111 0100 0000 0000
   HEXA = 0x00167400
   ```

d. beq $s0, $s1, -25

```
[FORMAT I]
OP:6-- RS:5- RT:5- CST:16----------
4      16    17    -25 (en CPL2)
000100 10000 10001 1111111111110111
0001 0010 0001 0001 1111 1111 1110 0111
HEXA = 0x1211FFE7
```

2. Déterminer l'instruction assembleur représentée par le code machine hexadécimal suivant :

(Note : les constantes sont représentées en complément à 2 sur 16 bits.)

a. 0x2108FFFF

0x2108FFFF = **0010 00**01 0000 1000 1111 1111 1111 1111
OPCODE = $001000_2 = 8_{16} = 8_{10}$ -> addi

```
OP:6-- RS:5- RT:5- CST:16----------
001000 01000 01000 1111111111111111
8      8     8     -1
```

Réponse: addi $t0, $t0, -1

b. 0x1A800007

0x1A800007 = **0001 10**10 1000 0000 0000 0000 0000 0111
OPCODE = $000110_2 = 6_{16} = 6_{10}$ -> blez

```
OP:6-- RS:5- RT:5- CST:16----------
000110 10100 00000 0000000000000111
6      20    0     7
```

Réponse: blez $s4, 7

c. 0x912A0000

0x912A0000 = **1001 00**01 0010 1010 0000 0000 0000 0000
OPCODE = $100100_2 = 24_{16} = 36_{10}$ -> lbu

```
OP:6-- RS:5- RT:5- CST:16----------
100100 01001 01010 0000000000000000
36     9     10    0
```

Réponse: lbu $t2, 0($t1)

3.9 Détection et correction d'erreurs

3.9.1 Introduction

Pas d'exercices.

3.9.2 Bit de parité

La technique du *contrôle de parité* consiste à ajouter un bit supplémentaire, appelé *bit de parité*, aux données à protéger. La valeur du bit de parité est calculée de telle sorte que le nombre total de bits à 1 soit pair (parité paire) ou impair (parité impaire).

1. Calculer le bit de parité paire/impaire pour chaque lettre du mot « LOVE » codé en ASCII sur 7 bits.

- L 76 1001100 PP=1 PI=0
- O 79 1001111 PP=1 PI=0
- V 86 1010110 PP=0 PI=1
- E 69 1000101 PP=1 PI=0

2. Les codes suivants sont-ils corrects si on utilise une parité paire ?

a. 10110011110001 → oui, il y a bien un nombre pair de bits à 1.
b. 01110001110010 → non, il y a un nombre impair de bits à 1.

3.9.3 Double parité

1. Compléter le tableau de double parité pour le mot « WORLD » en ASCII sur 7 bits. On suppose que les lignes utilisent une parité paire (PP) et les colonnes une parité impaire (PI).

Car.	b6	b5	b4	b3	b2	b1	b0	PP
W	1	0	1	0	1	1	1	1
O	1	0	0	1	1	1	1	1
R	1	0	1	0	0	1	0	1
L	1	0	0	1	1	0	0	1
D	1	0	0	0	1	0	0	0
PI	0	1	1	1	1	0	1	

2. Déterminer le mot codé en ASCII sur 7 bits qui correspond au tableau de double parité suivant. On suppose que les lignes utilisent une parité impaire (PI) et les colonnes une parité paire (PP).

On constate que le bit de parité impaire n'est pas correct pour les lignes 1, 3 et 5.
De même, le bit de parité paire n'est pas correct pour la colonne b4.
On a donc plusieurs erreurs localisées dans une seule colonne.
Pour rétablir des parités correctes, il faut inverser le bit 4 dans les lignes 1, 3 et 5

Car.	Code	b6	b5	b4	b3	b2	b1	b0	PI
H	72	1	0	1→0	1	0	0	0	1
E	69	1	0	0	0	1	0	1	0
L	76	1	0	1→0	1	1	0	0	0
M	77	1	0	0	1	1	0	1	1
o	111	1	1	1→0	1	1	1	1	1
	PP	1	1	0	0	0	1	1	

Réponse : HELMo

3.9.4 Détection d'erreurs groupées (CRC)

Pas d'exercices.

3.9.5 Mémoire à code correcteur d'erreurs (ECC)

Pas d'exercices.

4. Circuits logiques

4.1 Algèbre de Boole et portes logiques

1. Déterminer les 2 formes normales correspondant à la table de vérité de l'opérateur NAND.

a	b	NAND	Formes normales
0	0	1	$\mathrm{FN1} = (\bar{a}\bar{b}) + (\bar{a}b) + (a\bar{b})$
0	1	1	
1	0	1	$\mathrm{FN2} = (\bar{a} + \bar{b})$
1	1	0	

2. Déterminer les 2 formes normales correspondant à la table de vérité de l'opérateur NOR.

a	b	NOR	Formes normales
0	0	1	$\mathrm{FN1} = (\bar{a}\bar{b})$
0	1	0	
1	0	0	$\mathrm{FN2} = (a + \bar{b})(\bar{a} + b)(\bar{a} + \bar{b})$
1	1	0	

3. Déterminez les 2 formes normales de la fonction F() donnée par la table de vérité suivante.

A	B	C	F()
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

FN1 (somme de mintermes) :

$$\overline{A}.\overline{B}.C \;+\; \overline{A}.B.C \;+\; A.\overline{B}.\overline{C} \;+\; A.\overline{B}.C \;+\; A.B.\overline{C}$$

FN2 (produit de maxtermes) :

$$(A + B + C)\,.\,(A + \overline{B} + C)\,.\,(\overline{A} + \overline{B} + \overline{C})$$

4. Déterminez les 2 formes normales de la fonction F() donnée par la table de vérité suivante.

A	B	C	F()
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

FN1 (somme de mintermes) :

$$\overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C + A.B.C$$

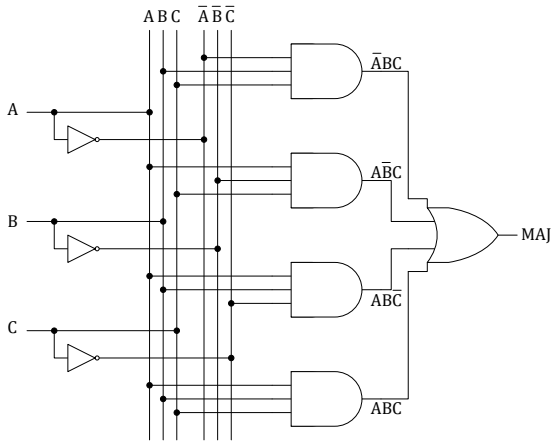
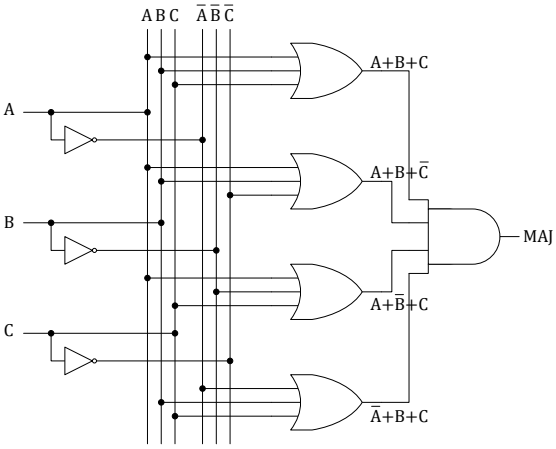
FN2 (produit de maxtermes) :

$$\left(A + \overline{B} + C\right).\left(\overline{A} + B + C\right).\left(\overline{A} + \overline{B} + C\right)$$

5. Considérons la fonction de 3 variables « majorité », c'est-à-dire une fonction F(a,b,c) dont la valeur vaut 1 lorsqu'au moins deux des trois variables a, b, c valent 1.

a. Déterminer la table de vérité de la fonction majorité.
b. Déterminer ses deux formes normales.
c. Dessiner le circuit générique (=non optimisé) qui correspond à chaque forme normale.

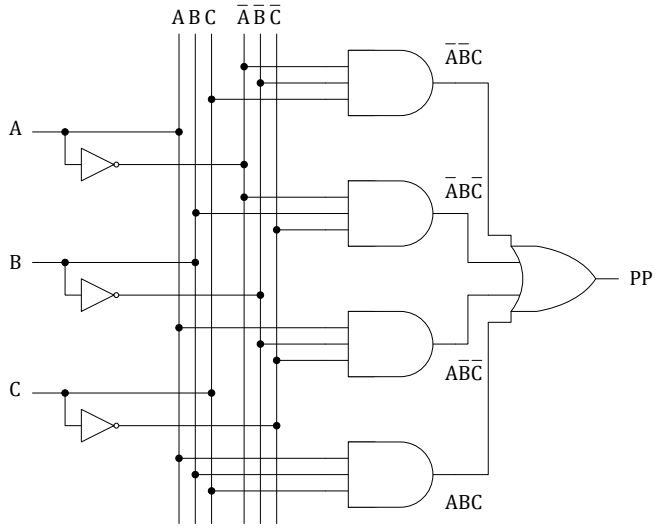
a	b	c	maj	Formes normales
0	0	0	0	$\mathrm{FN1} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$
0	0	1	0	
0	1	0	0	
0	1	1	1	
1	0	0	0	$\mathrm{FN2} = (a + b + c)(a + b + \bar{c})(a + \bar{b} + c)(\bar{a} + b + c)$
1	0	1	1	
1	1	0	1	
1	1	1	1	

FN1	FN2
	

6. Considérons la fonction de 3 variables « parité paire », c'est-à-dire une fonction F(a,b,c) dont la valeur vaut 1 lorsqu'un nombre impair des trois variables a, b, c valent 1.

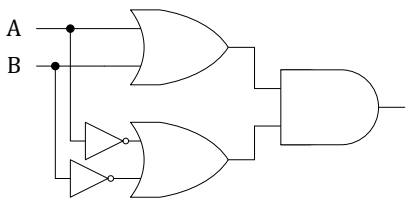
a. Déterminer la table de vérité de cette fonction.
b. Déterminer la <u>première</u> forme normale de cette fonction.
c. Dessiner le circuit générique (=non optimisé) qui correspond à la première forme normale.

a	b	c	PP	FN1
0	0	0	0	$\mathrm{FN1} = (\bar{a}\bar{b}c) + (\bar{a}b\bar{c}) + (a\bar{b}\bar{c}) + (abc)$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	



7. Soit l'expression algébrique suivante : $\boldsymbol{F(a,\ b) = (a + b)(\bar{a} + \bar{b})}$

a. Dessiner le circuit logique non simplifié.
b. Déterminer la table de vérité de F.
c. À quoi sert cette fonction ?

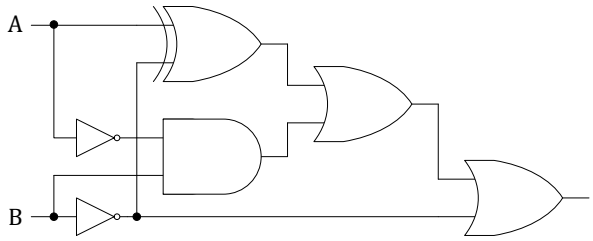


a	b	$a + b$	$\bar{a} + \bar{b}$	F
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

$\rightarrow F(a,b) = a \oplus b \rightarrow$ XOR

8. Soit l'expression algébrique suivante : $\mathbf{F(a,\ b) = \left((a \oplus \bar{b}) + (\bar{a}b)\right) + \bar{b}}$

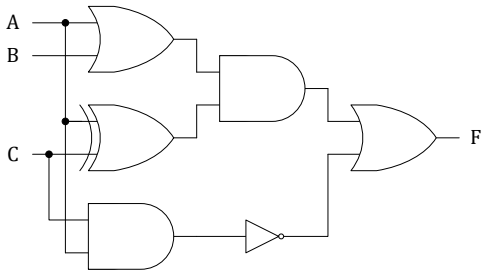
a. Dessiner le circuit logique non simplifié.
b. Déterminer la table de vérité de *F*.
c. À quoi sert cette fonction ?



		T1	T2	T3	
a	**b**	$a \oplus \overline{b}$	$\overline{a}b$	**T1+T2**	**F**
0	0	1	0	1	1
0	1	0	1	1	1
1	0	0	0	0	1
1	1	1	0	1	1

$\rightarrow F(a,b) = 1 \rightarrow$ constante 1

9. Considérons le circuit logique suivant :



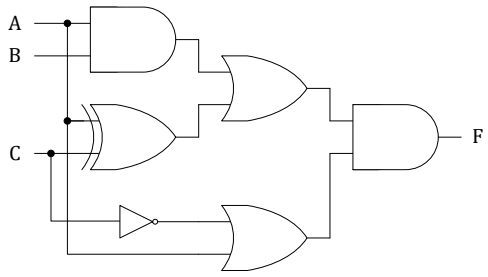
a. Déterminer l'expression algébrique de F.

$$F(A,B,C) = \left((A+B).(A \oplus C)\right) + \overline{(A.C)}$$

b. Déterminer la table de vérité de F.

A	B	C	F()
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

10. Considérons le circuit logique suivant :



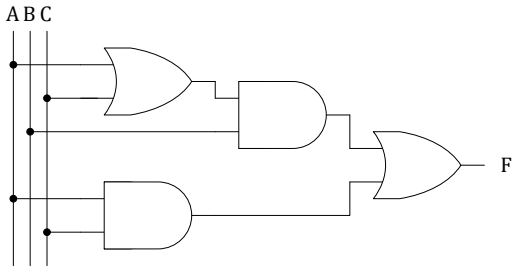
a. Déterminer l'expression algébrique de F.

$$F(A,B,C) = \big((A.B) + (A \oplus C)\big) . (A + \overline{C})$$

b. Déterminer la table de vérité de F.

A	B	C	F()
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

11. Considérons le circuit logique suivant :



a. Déterminer l'expression algébrique de F.

$$F(A,B,C) = \big((A + C).B\big) + (A.C)$$

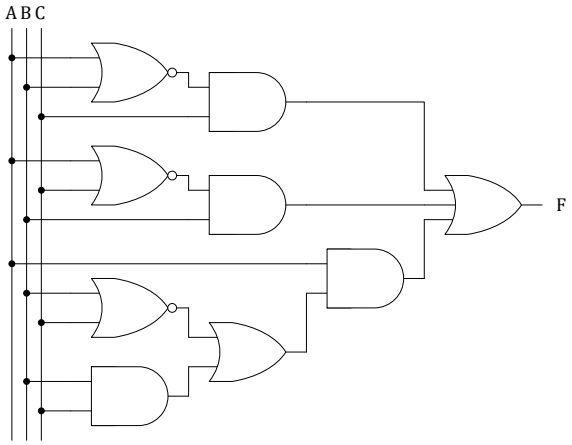
b. Déterminer la table de vérité de F.

c. À quoi sert cette fonction ?

A	B	C	A+C	(A+C).B	A.C	F
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	1	1	0	1
1	0	0	1	0	0	0
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	1

Il s'agit de la fonction majorité.

12. Considérons le circuit logique suivant :



a. Déterminer l'expression algébrique de F.
b. Déterminer la table de vérité de F.
c. A quoi sert cette fonction ?

$$F(a,b,c) = \big((a\ nor\ b).\mathrm{c}\big) + \big((a\ nor\ c).b\big) + a.\big((b\ nor\ c) + (b.c)\big)$$

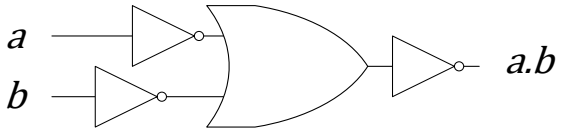
			T1	x	T2	x	T3	T4	T5	x	
a	b	c	a nor b	T1.c	a nor c	T2.b	b nor c	b.c	T3+T4	a.T5	F
0	0	0	1	0	1	0	1	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	1
0	1	0	0	0	1	1	0	0	0	0	1
0	1	1	0	0	0	0	0	1	1	0	0
1	0	0	0	0	0	0	1	0	1	1	1
1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	1	1	1	1

Il s'agit de la fonction parité paire.

13. Implémenter l'opérateur ET en utilisant uniquement des portes OU et NON.

Suggestion : utiliser le théorème de De Morgan.

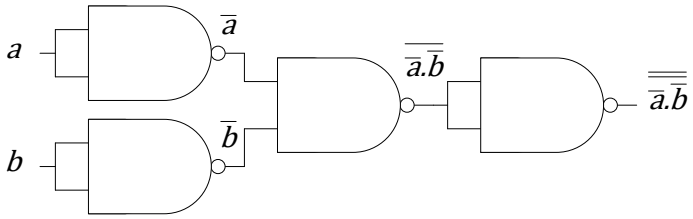
$$a.b = \overline{\overline{(a.b)}} = \overline{(\overline{a} + \overline{b})}$$



14. Implémenter l'opérateur NOR en utilisant uniquement des portes NAND.

Suggestion : utiliser le théorème de De Morgan.

$$a\ nor\ b = \overline{a + b} = \overline{a}.\overline{b} = \overline{\overline{\overline{a}.\overline{b}}} = \overline{\left(\overline{\overline{a}.\overline{b}}\right)}$$



15. Considérons une fonction logique de 4 variables dont la valeur vaut 1 lorsque (exactement) 2 des 4 variables d'entrée valent 1.

a. Construire la table de vérité de cette fonction.

Attention à l'ordre des lignes de la table de vérité : il doit correspondre à l'énumération des valeurs binaires des nombres de 0 à 15.

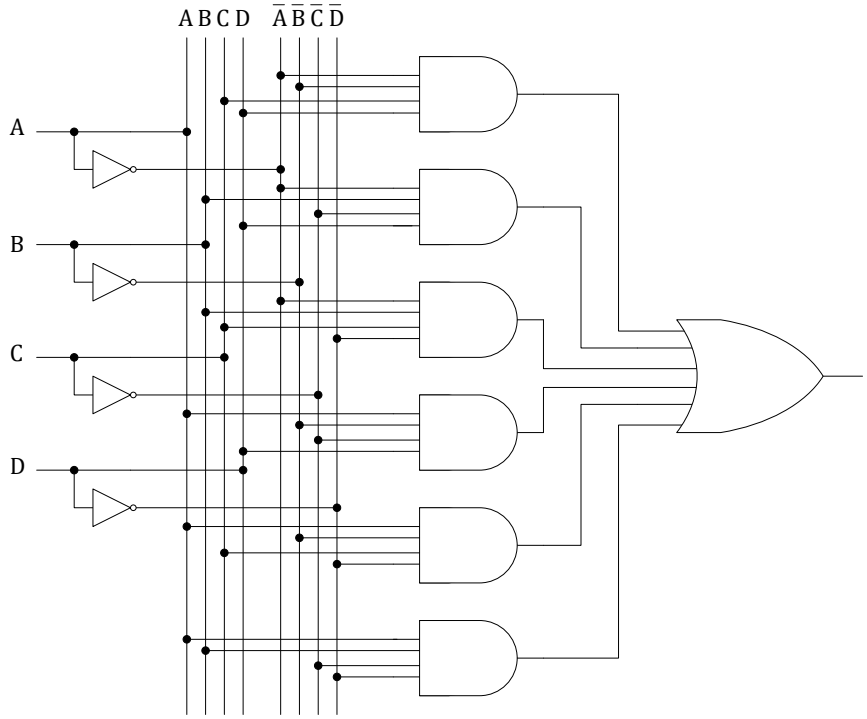
	A	B	C	D	F()
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

b. Déterminer la forme normale la plus compacte.

Il y a 6 combinaisons pour lesquelles la fonction vaut 1 et 10 pour lesquelles elle vaut 0. Nous choisissons par conséquent la première forme normale (somme de mintermes).

FN1 = $\bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + AB\bar{C}\bar{D}$

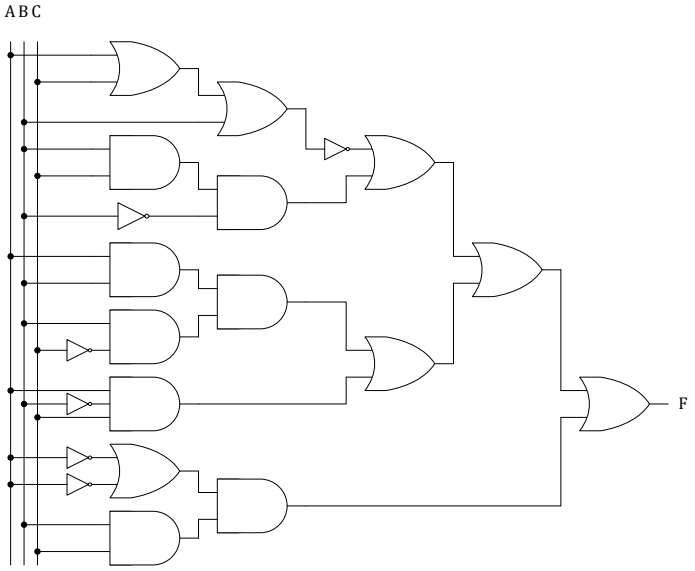
c. Dessiner le circuit correspondant.



d. Combien faudrait-il de portes à 2 ou 1 entrée(s) pour implémenter cette forme normale ?

18 portes ET, 5 portes OU et 4 portes NON → 27 portes

16. Considérons le circuit logique suivant :



a. Calculer l'expression algébrique qui correspond à ce circuit.

F = $\left(\overline{(A+C)+\bar{B}}+BC\bar{B}\right)+(AB\bar{B}\bar{C}+A\bar{B}C)+(\bar{A}+\bar{A})BC$

b. Calculer la table de vérité du circuit.

A	B	C	F()
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

c. Quel est le rôle de ce circuit ?

C'est un générateur de parité impaire.

17. Déterminez la table de vérité des fonctions $F(A,B,C)$ et $G(A,B,C)$ dont on vous donne une des deux formes normales. En déduire, l'autre forme normale.

$F(A,B,C) = \bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + A.\bar{B}.\bar{C} + A.B.\bar{C}$ Forme normale : 1

$G(A,B,C) = (\bar{A}+B+C).\ (\bar{A}+\bar{B}+C).\ (\bar{A}+B+\bar{C})$ Forme normale : 2

A	B	C	F(A,B,C)	G(A,B,C)
0	0	0	1	1
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1

Autre forme normale de chaque fonction :

$F(A,B,C) = \left(A+B+\overline{C}\right).\left(A+\overline{B}+\overline{C}\right).\left(\overline{A}+B+\overline{C}\right).\left(\overline{A}+\overline{B}+\overline{C}\right)$

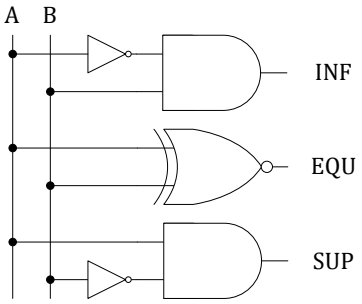
$G(A,B,C) = \left(\overline{A}.\overline{B}.\overline{C}\right) + \left(\overline{A}.\overline{B}.C\right) + \left(\overline{A}.B.\overline{C}\right) + \left(\overline{A}.B.C\right) + (A.B.C)$

4.2 Circuits combinatoires

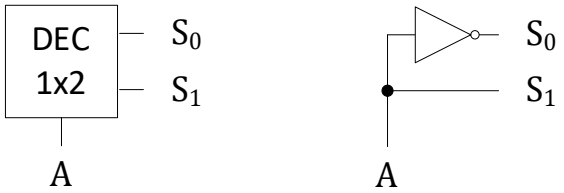
1. Concevoir un circuit comparateur qui reçoit en entrée deux nombres A et B exprimé sur 1 bit possède trois sorties INF, EQU et SUP qui valent 1 respectivement si $A < B$, $A == B$ ou $A > B$.

Suggestion : commencer par établir la table de vérité du circuit.

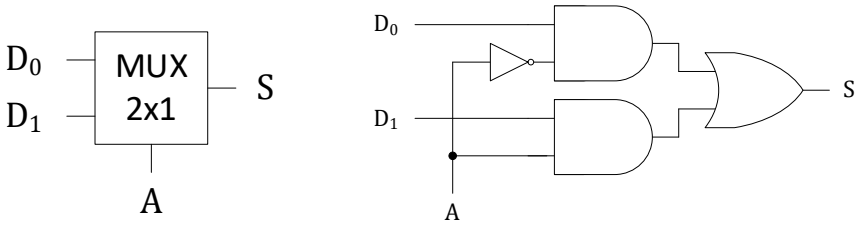
A	B	INF	EQU	SUP
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



2. Réaliser un circuit décodeur 1×2.



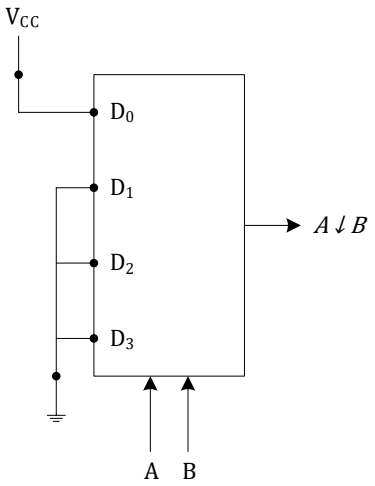
3. Réaliser un circuit multiplexeur 2×1.



4. Utiliser un multiplexeur pour implémenter les fonctions suivantes :

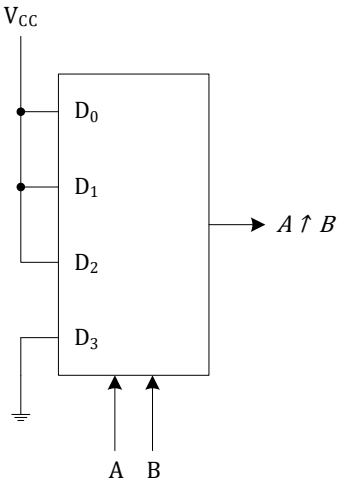
a. NOR

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0



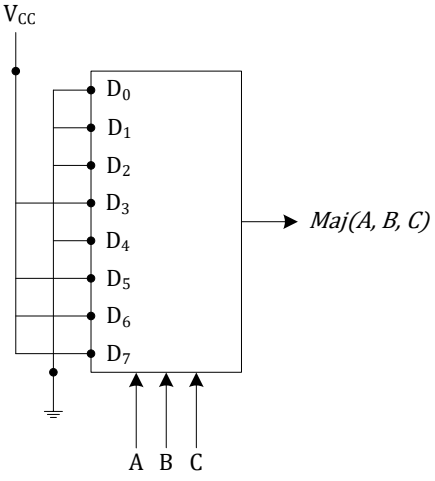
b. NAND

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0



c. Majorité de 3 variables

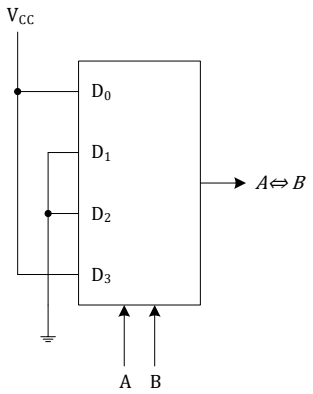
A	B	C	MAJ
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



5. Déterminer la table de vérité de la fonction $F(A,B)$ implémentée par le multiplexeur suivant :

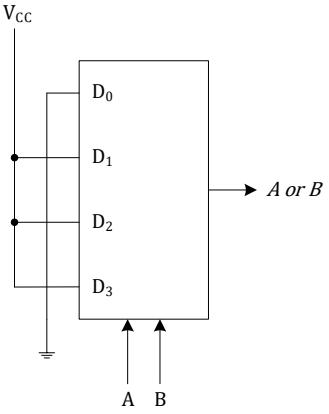
a. Fonction 1

A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



b. Fonction 2

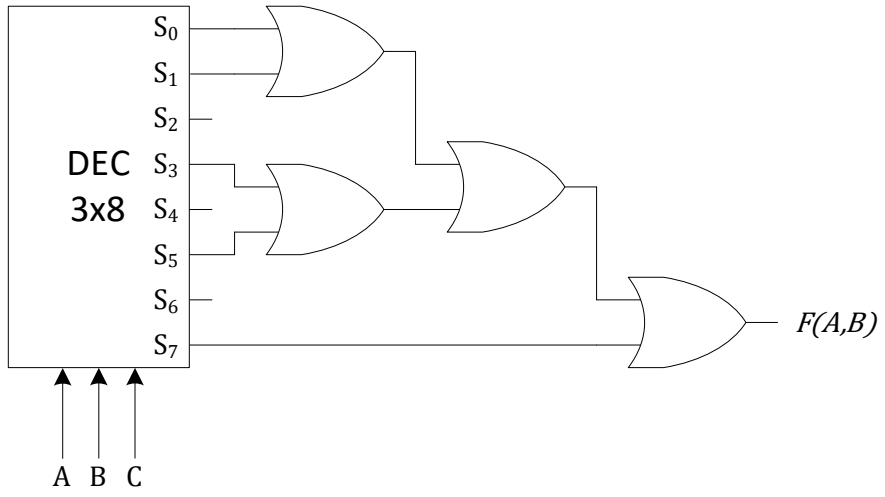
A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1



6. En utilisant un décodeur 3×8 et des portes OU à 2 entrées, implémenter la fonction $F(A,B,C)$ dont la première forme normale est :

$$F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C + A.B.C$$

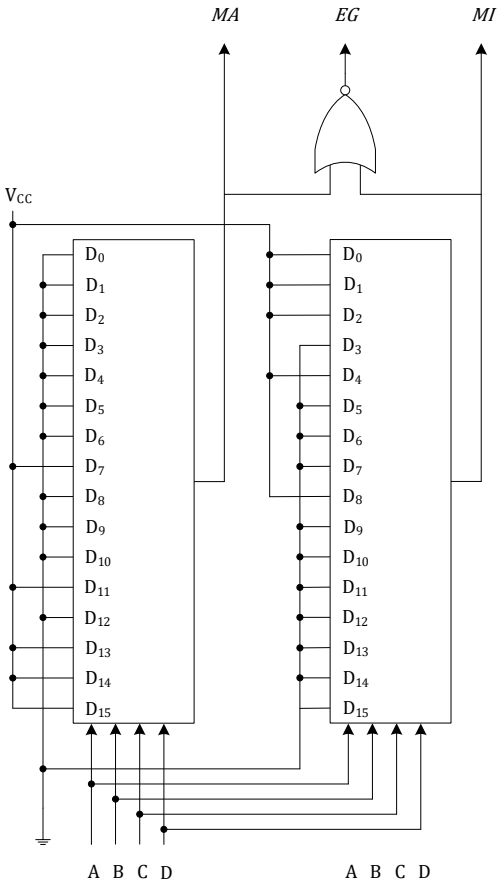
Les mintermes de F correspondent aux sorties 000 (S_0), 001 (S_1), 011 (S_3), 101 (S_5) et 111 (S_7) du décodeur, qu'il faut ensuite combiner avec des portes OU.



7. Au moyen de deux multiplexeurs et d'une porte NOR, implémenter un circuit qui détermine si une majorité, une égalité ou une minorité de ses 4 d'entrées valent 1.

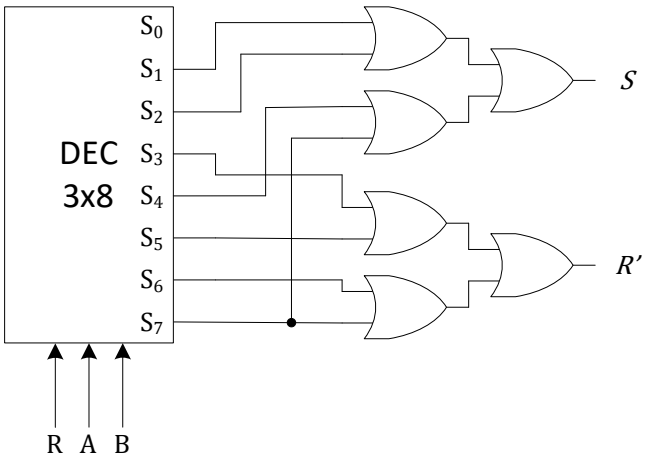
La table vérité nous donne les valeurs des différentes sorties MA, EG et MI. Une seule de ces trois sorties peut être active à un instant donné puisque les trois conditions sont exclusives entre elles. On peut utiliser deux multiplexeurs 16x1 pour calculer deux des trois conditions et implémenter la troisième avec la porte NOR, car elle sera vraie si aucune des deux autres ne l'est.

	A	B	C	D	MA	EG	MI
0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	1
3	0	0	1	1	0	1	0
4	0	1	0	0	0	0	1
5	0	1	0	1	0	1	0
6	0	1	1	0	0	1	0
7	0	1	1	1	1	0	0
8	1	0	0	0	0	0	1
9	1	0	0	1	0	1	0
10	1	0	1	0	0	1	0
11	1	0	1	1	1	0	0
12	1	1	0	0	0	1	0
13	1	1	0	1	1	0	0
14	1	1	1	0	1	0	0
15	1	1	1	1	1	0	0



8. Réaliser un additionneur complet en utilisant un décodeur 3×8 et des portes OU à 2 entrées.

R	A	B	S	R'
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



4.3 Unité arithmétique et logique (ALU)

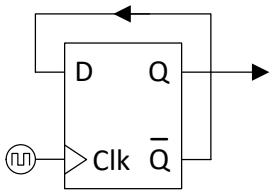
Pas d'exercices.

4.4 Circuits asynchrones et synchrones

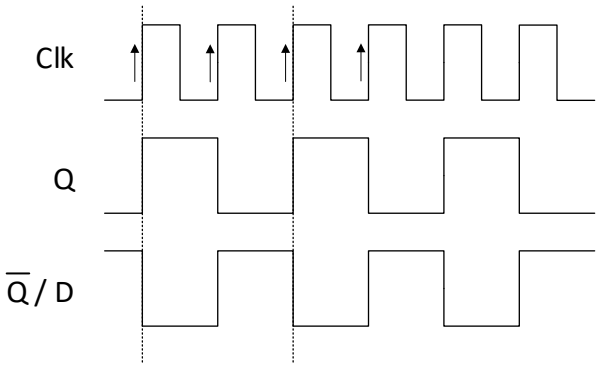
Pas d'exercices.

4.5 Circuits séquentiels

1. Quelle est l'utilité du circuit suivant ?



Le schéma suivant montre le signal d'horloge et la sortie du circuit au cours du temps.



À chaque transition montante du signal d'horloge, l'état de la bascule s'inverse, car l'entrée D est connectée à $\overline{Q} \Rightarrow$ 0, 1, 0, 1, 0, 1, etc. La sortie du circuit vaut donc 0 (ou 1) tous les deux cycles d'horloge.

Conclusion : il s'agit d'un *diviseur de fréquence*. En effet, le circuit divise la fréquence de l'horloge par deux.

2. Construire un compteur d'impulsions sur 4 bits en utilisant des bascules D.

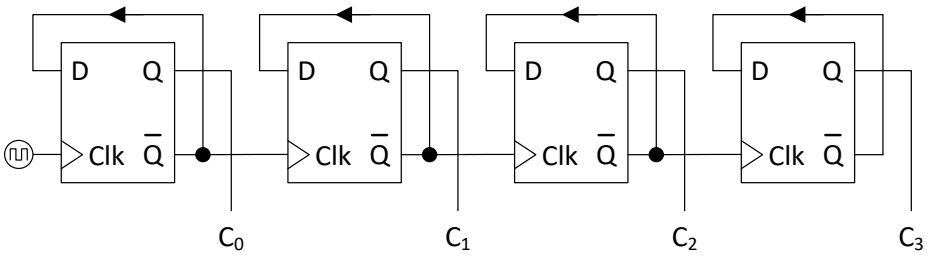
Suggestion : la solution est basée sur le circuit de l'exercice précédent.

En effet, le circuit de l'exercice 1 passe alternativement de 0 à 1 à chaque cycle d'horloge et génère un signal dont la fréquence est la moitié de celle du signal d'horloge de départ.

Si on utilise ce signal de sortie comme signal d'horloge pour un deuxième circuit similaire à celui de l'exercice 1, l'état de la deuxième bascule va lui aussi passer alternativement de 0 à 1 à chaque cycle complet de la première bascule, c'est-à-dire deux fois plus lentement. Le signal de sortie de la deuxième bascule aura donc une fréquence qui est le quart de celle de l'horloge.

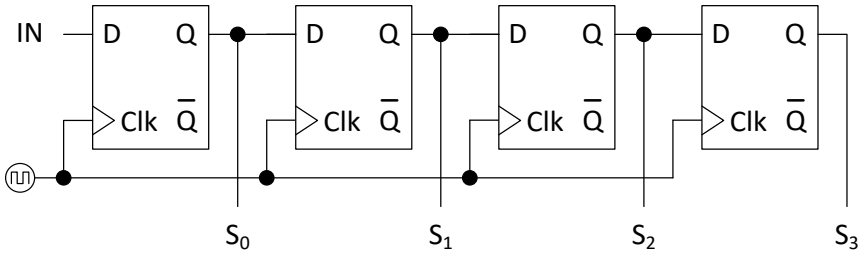
Pour obtenir un compteur, il faut que la deuxième bascule change d'état chaque fois que la première passe de l'état $Q = 1$ à l'état $Q = 0$ (Note : la sortie $\bar{Q}$ passe quant à elle de 0 à 1). Comme nous utilisons des bascules D qui changent d'état lors d'une transition montante du signal d'horloge, nous devons donc utiliser $\bar{Q}$ comme signal d'horloge pour la bascule suivante.

Il suffit de mettre en série quatre circuits pour obtenir un compteur d'impulsions sur 4 bits.



	C_0	C_1	C_2	C_3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

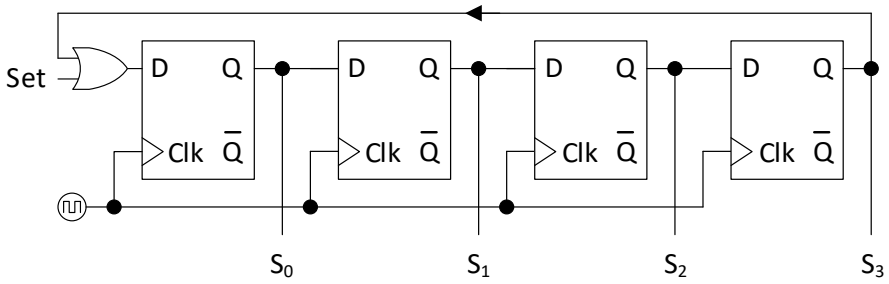
3. Quelle est l'utilité du circuit suivant ?



Il s'agit d'un *registre à décalage*.

À chaque impulsion de l'horloge, les données contenues dans le registre sont décalées d'un bit vers la droite : IN → S0 → S1 → S2 → S3

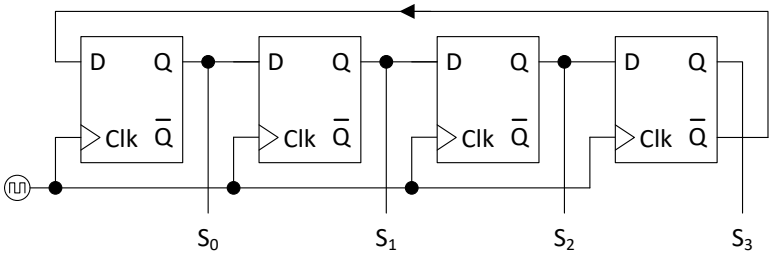
4. Quelle est l'utilité du circuit suivant ?



Il s'agit d'un *compteur en anneau*.

Si on initialise la première bascule à 1, le circuit passe par les **4 états distincts** 1000, 0100, 0010, 0001, puis il recommence la séquence. Il n'y a qu'un seul bit de sortie actif à un instant donné et ce bit se décale de manière circulaire dans le registre. Ce type de compteur est utilisé pour générer des signaux de synchronisation.

5. Quelle est l'utilité du circuit suivant ?

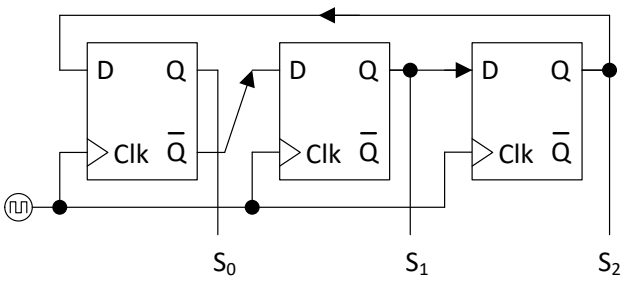


Il s'agit d'un *compteur Johnson*.

C'est une variante du compteur en anneau dans laquelle le signal d'entrée de la première bascule est égal au signal $\overline{Q}$ de la dernière. Si on démarre dans l'état 0000, on obtient une séquence cyclique contenant **8 états distincts** : 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, …

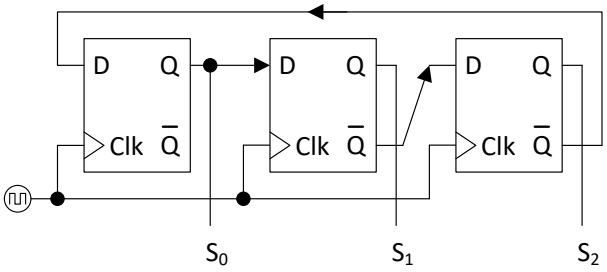
Il est possible de se synchroniser sur un des 8 états avec une porte ET qui combine deux sorties successives : $\underline{0}\mathtt{000}\underline{0} \rightarrow \overline{S_0}.\overline{S_3}$, $\underline{1}\mathtt{000}\underline{0} \rightarrow S_0.\overline{S_1}$, $\mathtt{1}\underline{1}\mathtt{00}\underline{0} \rightarrow S_1.\overline{S_2}$, $\mathtt{11}\underline{1}\mathtt{0}\underline{0} \rightarrow S_2.\overline{S_3}$, etc.

6. Si l'état initial (T_0) du circuit représenté ci-dessous est $Q_0Q_1Q_2 = 000$, quel sera son état aux instants suivants T_1, T_2, T_3, … ?



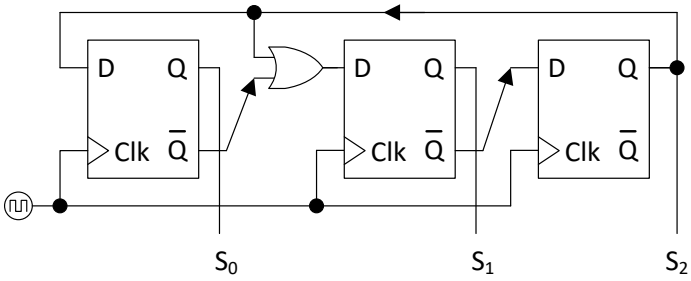
	Q_0	Q_1	Q_2
T+1 →	$= Q_2$	$= \overline{Q_0}$	$= Q_1$
T_0	**0**	**0**	**0**
T_1	0	1	0
T_2	0	1	1
T_3	1	1	1
T_4	1	0	1
T_5	1	0	0
T_6	0	0	0
T_7	0	1	0

7. Si l'état initial (T_0) du circuit représenté ci-dessous est $Q_0Q_1Q_2 = 000$, quel sera son état aux instants suivants T_1, T_2, T_3, … ?



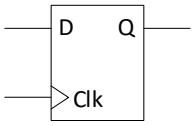
	Q_0	Q_1	Q_2
T+1 →	$= \overline{Q_2}$	$= Q_0$	$= \overline{Q_1}$
T_0	**0**	**0**	**0**
T_1	1	0	1
T_2	0	1	1
T_3	0	0	0
T_4	1	0	1
T_5	0	1	1
T_6	0	0	0
T_7	1	0	1

8. Si l'état initial (T_0) du circuit représenté ci-dessous est $Q_0Q_1Q_2$ = 000, quel sera son état aux instants suivants T_1, T_2, T_3, … ?

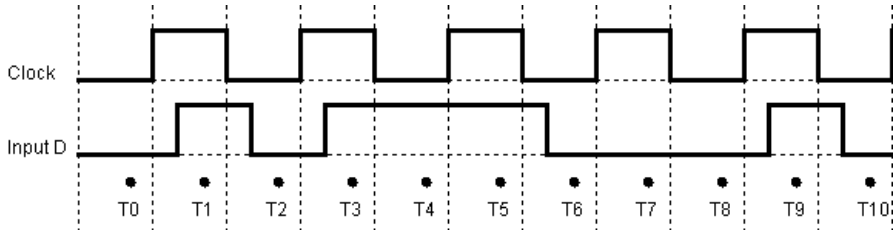


	Q_0	Q_1	Q_2
T+1 →	$= Q_2$	$= \overline{Q_0} + Q_2$	$= \overline{Q_1}$
T_0	**0**	**0**	**0**
T_1	0	1	1
T_2	1	1	0
T_3	0	0	0
T_4	0	1	1
T_5	1	1	0
T_6	0	0	0
T_7	0	1	1

9. Considérons une bascule D dont le changement d'état est synchronisé sur une transition <u>montante</u> du signal d'horloge :



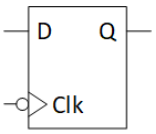
Le chronogramme ci-dessous indique les variations du signal d'horloge (Clock) et du signal à l'entrée D de la bascule au fil du temps. Un signal de niveau haut signifie 1 et un signal de niveau bas signifie 0.



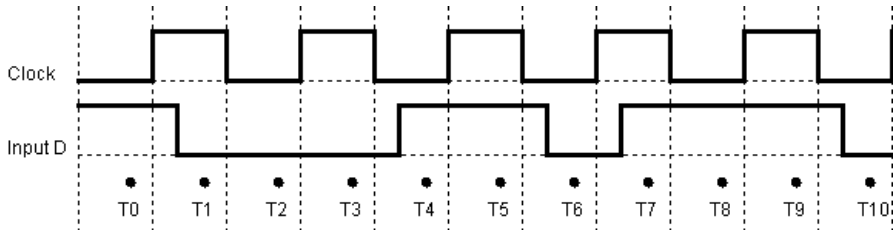
À l'instant T_0, la bascule se trouve dans l'état 0 ($Q = 0$). Déterminez l'état Q de la bascule (0 ou 1) aux instants $T_1, T_2, \dots, T_{10}$.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
0	0	0	0	0	1	1	0	0	0	0

10. Considérons une bascule D dont le changement d'état est synchronisé sur une transition <u>descendante</u> du signal d'horloge :



Le chronogramme ci-dessous indique les variations du signal d'horloge (Clock) et du signal à l'entrée D de la bascule au fil du temps. Un signal de niveau haut signifie 1 et un signal de niveau bas signifie 0.



À l'instant T_0, la bascule se trouve dans l'état 1 ($Q = 1$). Déterminez l'état Q de la bascule (0 ou 1) aux instants $T_1, T_2, \dots, T_{10}$.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
1	1	0	0	0	0	1	1	1	1	1

5. Mémoires

5.5 Mémoire cache

1. Soit une mémoire cache à <u>correspondance directe</u>.

- **La mémoire cache comporte 6 lignes.**
- **L'état initial de la mémoire est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		18	**27**	**3**	**11**	**22**	**29**	**15**	**25**	**13**	**1**
Ligne	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	30	**18**	18	18	18	18	18	18	18	18	18
1	7	7	7	7	7	7	7	7	**25**	**13**	**1**
2	-	-	-	-	-	-	-	-	-	-	-
3	9	9	**27**	**3**	3	3	3	**15**	15	15	15
4	-	-	-	-	-	**22**	22	22	22	22	22
5	-	-	-	-	**11**	11	**29**	29	29	29	29

2. Soit une mémoire cache à <u>complètement associative</u> qui utilise un algorithme de <u>remplacement circulaire</u>.

- **La mémoire cache comporte 6 lignes.**
- **L'état initial de la mémoire est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		11	**17**	**28**	**29**	**7**	**17**	**31**	**12**	**18**	**14**
Ligne	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	22	22	**17**	17	17	17	**17**	17	17	**18**	18
1	12	12	12	**28**	28	28	28	28	28	28	**14**
2	7	7	7	7	**29**	29	29	29	29	29	29
3	55	55	55	55	55	**7**	7	7	7	7	7
4	37	37	37	37	37	37	37	**31**	31	31	31
5	-	**11**	11	11	11	11	11	11	**12**	12	12

3. Soit une mémoire cache <u>associative par ensemble</u> qui utilise un algorithme de <u>remplacement circulaire</u>.

- **La mémoire cache comporte 4 ensembles (sets) de 2 lignes chacun.**
- **L'état initial de la mémoire cache est indiqué dans la colonne « Init ».**

Indiquez dans chaque colonne du tableau le contenu de la mémoire cache <u>après</u> avoir demandé un accès au bloc mémoire dont le numéro figure au-dessus de la colonne.

		22	**23**	**20**	**21**	**17**	**16**	**20**	**25**	**29**	**17**
Set	**Init**	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	4	4	4	4	4	4	**16**	16	16	16	16
	-	-	-	**20**	20	20	20	**20**	20	20	20
1	17	17	17	17	17	**17**	17	17	**25**	25	**17**
	-	-	-	-	**21**	21	21	21	21	**29**	29
2	18	18	18	18	18	18	18	18	18	18	18
	-	**22**	22	22	22	22	22	22	22	22	22
3	19	19	19	19	19	19	19	19	19	19	19
	-	-	**23**	23	23	23	23	23	23	23	23

6. Architectures avancées

Pas d'exercices sur le chapitre 6.

7. Programmation en assembleur

Le code assembleur des solutions est disponible dans l'espace e-learning du cours.

Nombres de 0 à 15

Décimal	Hexa	Binaire	Octal	Binaire
0	0	0000	0	000
1	1	0001	1	001
2	2	0010	2	010
3	3	0011	3	011
4	4	0100	4	100
5	5	0101	5	101
6	6	0110	6	110
7	7	0111	7	111
8	8	1000	-	-
9	9	1001	-	-
10	A	1010	-	-
11	B	1011	-	-
12	C	1100	-	-
13	D	1101	-	-
14	E	1110	-	-
15	F	1111	-	-

Puissances de 2, 8 et 16

n	2^n	8^n	16^n
-4	0,0625	0,000244140625	-
-3	0,125	0,001953125	0,000244140625
-2	0,25	0,015625	0,00390625
-1	0,5	0,125	0,0625
0	1	1	1
1	2	8	16
2	4	64	256
3	8	512	4.096
4	16	4.096	65.536
5	32	32.768	1.048.576
6	64	262.144	16.777.216
7	128	2.097.152	268.435.456
8	256	16.777.216	4.294.967.296
9	512	134.217.728	68.719.476.736
10	1.024	1.073.741.824	-
11	2.048	8.589.934.592	-

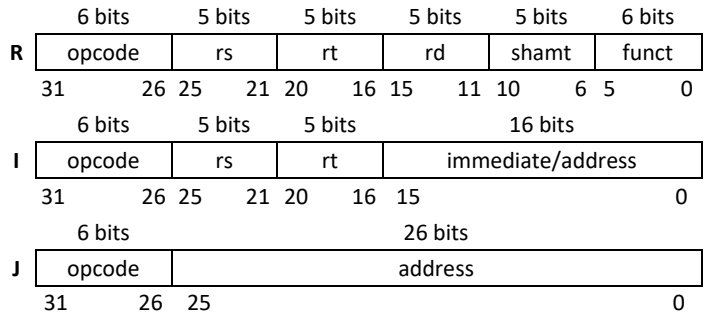
Code ASCII (7 bits) - Extrait

Code	Binaire	Car	Code	Binaire	Car
64	1000000	@	96	1100000	`
65	1000001	A	97	1100001	a
66	1000010	B	98	1100010	b
67	1000011	C	99	1100011	c
68	1000100	D	100	1100100	d
69	1000101	E	101	1100101	e
70	1000110	F	102	1100110	f
71	1000111	G	103	1100111	g
72	1001000	H	104	1101000	h
73	1001001	I	105	1101001	i
74	1001010	J	106	1101010	j
75	1001011	K	107	1101011	k
76	1001100	L	108	1101100	l
77	1001101	M	109	1101101	m
78	1001110	N	110	1101110	n
79	1001111	O	111	1101111	o
80	1010000	P	112	1110000	p
81	1010001	Q	113	1110001	q
82	1010010	R	114	1110010	r
83	1010011	S	115	1110011	s
84	1010100	T	116	1110100	t
85	1010101	U	117	1110101	u
86	1010110	V	118	1110110	v
87	1010111	W	119	1110111	w
88	1011000	X	120	1111000	x
89	1011001	Y	121	1111001	y
90	1011010	Z	122	1111010	z
91	1011011	[	123	1111011	{
92	1011100	\	124	1111100	|
93	1011101	]	125	1111101	}
94	1011110	^	126	1111110	~
95	1011111	_	127	1111111	DEL

Registres généraux du processeur MIPS

Name	Number	Use	Preserv
$zero	0	Constant value 0	NA
$v0-$v1	2-3	Function results, expr. evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved temporaries	Yes
$t8-t9	24-25	Temporaries	No
$sp	29	Stack pointer	Yes
$ra	31	Return address	Yes

Formats des instructions MIPS



Instructions et pseudo-instructions utilitaires

Instr.	Syntax	Meaning
la	`la $t1, label`	Load Address : Set $t1 to label's address
li	`li $t1, imm`	Load Immediate : Set $t1 to immediate
move	`move $t1, $t2`	Move : Set $t1 to contents of $t2
mfhi	`mfhi $t1`	Move from Hi : Set $t1 to contents of Hi
mflo	`mflo $t1`	Move from Lo : Set $t1 to contents of Lo
mthi	`mthi $t1`	Move to Hi : Set Hi to contents of $t1
mtlo	`mtlo $t1`	Move to Lo : Set Lo to contents of $t1
break	`break imm`	Terminate execution with exception code
nop	nop	Null operation
syscall	syscall	System call specified by value in $v0

Instructions de base de l'assembleur MIPS

Instr.	Syntax	Operation	Fmt	Op/Fct	Meaning
add	add rd, rs, rt	R[rd] = R[rs] + R[rt]	R	$0/20_{hex}$	Add
addi	addi rt, rs, imm	R[rt] = R[rs] + SignExtImm	I	8_{hex}	Add Immediate
addiu	addiu rt, rs, imm	R[rt] = R[rs] + SignExtImm	I	9_{hex}	Add Immediate Unsigned
addu	addu rd, rs, rt	R[rd] = R[rs] + R[rt]	R	$0/21_{hex}$	Add Unsigned
and	and rd, rs, rt	R[rd] = R[rs] & R[rt]	R	$0/24_{hex}$	And
andi	andi rt, rs, imm	R[rt] = R[rs] & ZeroExtImm	I	C_{hex}	And Immediate
beq	beq rs, rt, addr	If (R[rs] == R[rt]) PC = PC + 4 + BranchAddr	I	4_{hex}	Branch On Equal
bgtz	bgtz rs, addr	If (R[rs] > 0) PC = PC + 4 + BranchAddr	I	7_{hex}	Branch On Greater Than Zero
blez	blez rs, addr	If (R[rs] <= 0) PC = PC + 4 + BranchAddr	I	6_{hex}	Branch On Less Than Or Equal Zero
bne	bne rs, rt, addr	If (R[rs] != R[rt]) PC = PC + 4 + BranchAddr	I	5_{hex}	Branch On Not Equal
j	j addr	PC = JumpAddr	J	2_{hex}	Jump
jal	jal addr	R[31] = PC + 8 ; PC = JumpAddr	J	3_{hex}	Jump And Link
jalr	jalr rs	R[31] = PC + 8 ; PC = R[rs] ;	R	$0/09_{hex}$	Jump And Link Register
jalr	jalr rd, rs	R[rd] = PC + 8 ; PC = R[rs] ;	R	$0/09_{hex}$	Jump And Link Register
jr	jr rs	PC = R[rs]	R	$0/08_{hex}$	Jump Register
lb	lb rt, imm(rs)	R[rt] = SignExt(M[R[rs] + SignExtImm](7:0))	I	20_{hex}	Load Byte
lbu	lbu rt, imm(rs)	R[rt] = ZeroExt(M[R[rs] + SignExtImm](7:0))	I	24_{hex}	Load Byte Unsigned
lh	lh rt, imm(rs)	R[rt] = SignExt(M[R[rs] + SignExtImm](15:0))	I	21_{hex}	Load Halfword
lhu	lhu rt, imm(rs)	R[rt] = ZeroExt(M[R[rs] + SignExtImm](15:0))	I	25_{hex}	Load Halfword Unsigned
lui	lui rt, imm	R[rt] = {imm, 16'b0}	I	F_{hex}	Load Upper Immediate
lw	lw rt, imm(rs)	R[rt] = M[R[rs] + SignExtImm]	I	23_{hex}	Load Word
or	or rd, rs, rt	R[rd] = R[rs] | R[rt]	R	$0/25_{hex}$	Or
ori	ori rt, rs, imm	R[rt] = R[rs] | ZeroExtImm	I	D_{hex}	Or Immediate
nor	nor rd, rs, rt	R[rd] = ~ (R[rs] | R[rt])	R	$0/27_{hex}$	Nor
xor	xor rd, rs, rt	R[rd] = R[rs] ^ R[rt]	R	$0/26_{hex}$	Xor
xori	xori rt, rs, imm	R[rt] = R[rs] ^ ZeroExtImm	I	E_{hex}	Xor Immediate
sll	sll rd, rt, shamt	R[rd] = R[rt] << shamt	R	$0/00_{hex}$	Shift Left Logical
sllv	sllv rd, rt, rs	R[rd] = R[rt] << R[rs]	R	$0/04_{hex}$	Shift Left Logical Variable
sra	sra rd, rt, shamt	R[rd] = R[rt] >> shamt	R	$0/03_{hex}$	Shift Right Arithmetic
srav	srav rd, rt, rs	R[rd] = R[rt] >> R[rs]	R	$0/07_{hex}$	Shift Right Arithmetic Variable
srl	srl rd, rt, shamt	R[rd] = R[rt] >>> shamt	R	$0/02_{hex}$	Shift Right Logical
srlv	srlv rd, rt, rs	R[rd] = R[rt] >>> R[rs]	R	$0/06_{hex}$	Shift Right Logical Variable
slt	slt rd, rs, rt	R[rd] = (R[rs] < R[rt]) ? 1 : 0	R	$0/2A_{hex}$	Set Less Than
slti	slti rt, rs, imm	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	I	A_{hex}	Set Less Than Immediate
sltiu	sltiu rt, rs, imm	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	I	B_{hex}	Set Less Than Immediate Unsigned
sltu	sltu rd, rs, rt	R[rd] = (R[rs] < R[rt]) ? 1 : 0	R	$0/2B_{hex}$	Set Less Than Unsigned
sb	sb rt, imm(rs)	M[R[rs] + SignExtImm](7:0) = R[rt](7:0)	I	28_{hex}	Store Byte
sh	sh rt, imm(rs)	M[R[rs] + SignExtImm](15:0) = R[rt](15:0)	I	29_{hex}	Store Halfword
sw	sw rt, imm(rs)	M[R[rs] + SignExtImm] = R[rt]	I	$2B_{hex}$	Store Word
sub	sub rd, rs, rt	R[rd] = R[rs] - R[rt]	R	$0/22_{hex}$	Subtract
subu	subu rd, rs, rt	R[rd] = R[rs] - R[rt]	R	$0/23_{hex}$	Subtract Unsigned

SignExtImm = extension du signe / ZeroExtImm = ajout de zéros / (7:0) = bit 7 à 0 / 16'b0 = 16 bits à 0