Université des Sciences et Techniques de Masuku

Faculté des Sciences

Département de Mathématiques et Informatique

Licence Mathématiques Physique et Chimie

(L1MPC)

CIRCUITS LOGIQUES ET CODAGE DE L'INFORMATION

2023/2024

Enseignant : Henri Mesmin Noël BOUITYVOUBOU

henri.bouityvoubou@gmail.com

February 16, 2024

Table des matières

1. Introduction

Le cours de circuits logiques et codage de l'information, souvent appelé INFO1B, vise à enseigner les fondamentaux du codage de l'information en informatique qui sont pertinents pour comprendre les principes sous-jacents à de nombreux domaines de l'informatique. Ce type de cours est généralement inclus dans les programmes d'études en informatique ou en electronique.

1.1. Objectif du cours

Ce cours de Logique et Codage de l'information est un cours proposé aux étudiants en première année de licence de Mathématique-Informatique et Physique-Chimie.

Il a pour but d'initier les étudiants à différentes problématiques du codage de l'information et de la programmation en python des opérations de transcodage :

1. Représentation des nombres en informatique
2. Représentation des nombres en entiers non signés
3. Représentation des nombres en entiers signés
4. Représentation des nombres flottants
5. Opérations logiques sur les entiers
6. Représentation des caractères
7. Les circuits logiques
8. La Programmation en Python

1.2. Le document du cours

Ce document est une compilation de notes de cours sur la réprésentation des nombres en informatique. Il est certe incomplet, et doit sans doute contenir des erreurs. Pour ces erreurs merci de les signaler à l'adresse ***henri.bouityvoubou@gmail.com***.

Ce document de cours n'est certainement pas exhaustif, et ne permet certainement pas à un apprenant de se dispenser des séances de cours et de TD.

Ce document va évoluer chaque année par la correction des erreurs qu'il peut contenir, et par la complétion de certaines sections.

Ce document est inspiré d'une version réalisée par Eric Cariou, elle-même refondue d'une version précédente.

2. Représentation des nombres en informatique

Il y a 10 sortes de personnes : celles qui comprennent le binaire, et les autres.

La représentation numérique en informatique est la méthode utilisée dans les systèmes informatiques pour stocker et manipuler des valeurs numériques. Les ordinateurs utilisent un système de nombres binaires, ce qui signifie qu'ils utilisent la base 2 pour représenter les nombres. La représentation binaire utilise uniquement les nombres 0 et 1 pour représenter les valeurs.

2.1. Bases courantes en informatique

En informatique les bases les plus utilisées sont le binaire, l'octal, et l'hexadécimal :

1. Le binaire est utilisé pour des raisons évidentes, un bit (binary digit) étant la quantité minimale d'information pouvant être transmise.

2. L'octal sert dans le cadre d'une représentation par paquets de 3 bits. Dans les années 1960, des ordinateurs travaillaient avec des registres de 12 ou 18 bits (par exemple le DEC PDP-8).

3. De nos jours, l'hexadécimal est particulièrement utilisé car il sert à représenter des paquets de 4 bits : les processeurs actuels travaillent avec des registres de 8, 16, 32 ou 64 bits.

Table 1: *Table de correspondance*

Décimal	Hexadécimal	Octal	Binaire
00	0	00	0000
01	1	01	0001
02	2	02	0010
03	3	03	0011
04	4	04	0100
05	5	05	0101
06	6	06	0110
07	7	07	0111
08	8	10	1000
09	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

2.2. Autres représentations

Bien que la représentation de nombres en binaire, octal ou hexadécimal soit courante, il ne s'agit pas de la seule représentation possible. D'autres représentations ont été utilisées ou continuent à être utilisées.

2.2.1 Biquinaire

Le biquinaire permet la représentation de chires décimaux. Cette représentation, avant d'être employée dans des ordinateurs, l'a été dans des bouliers depuis environ deux-mille ans. La représentation biquinaire, comme son nom l'indique, se décompose en deux éléments. Un élément à deux états (binaire) et un élément à 5 états (quinaire). L'élément à 5 états est utilisé pour coder les chires de 0 à 4. L'élément à 2 états sert à pouvoir ajouter 5 au chire représenté par le composant quinaire. Il est donc possible de représenter tous les chires compris entre 0 et 9. La représentation biquinaire peut se faire sur 7 bits, dans ce cas l'élément à deux états est sur deux bits (un seul est à 1), et l'élément à cinq états est sur cinq bits (dont un seul est à 1). Voir des exemples dans la Table 1.2 page suivante.

2.2.2 Binaire codé décimal (BCD)

La représentation BCD utilise un nombre xe de bits pour représenter chacun des chires décimaux par leur représentation binaire. Le nombre minimal de bits an de représenter dix valeurs étant de quatre 1 , un chire en BCD est représenté sur quatre bits, quelque soit sa valeur (voir Table 1.2). L'addition de deux nombres BCD peut donner des résultats semblant absurdes. Par exemple si on additionne 7 et 8 en représentation BCD, cela nous donne 0111 + 1000 = 1111 . Or 1111 n'est pas un chire en BCD. Dans une telle situation, lorsque le résultat dépasse le plus grand chire représentable (1001), il faut ajouter six (0110) au chire dépassant la capacité. Dans le cas présent cela nous donne 1111 + 0110 = 10101 qui, remis sur un multiple de quatre bits, correspond à 00010101 , soit la représentation de 15 en BCD.

Table 2: *Table de conversion*

Décimal	Biquinaire	BCD
0	0100001	0000
1	0100010	0001
2	0100100	0010
3	0101000	0011
4	0110000	0100
5	1000001	0101
6	1000010	0110
7	1000100	0111
8	1001000	1000
9	1010000	1001

3. Représentation des nombres entiers non signés

3.1. Decimal vers une base 2, 8 ou 16

- Lorsque l'on souhaite représenter un nombre décimal (base 10) en une base donnée , on peut choisir différentes méthodes :
  - mémoriser l'ensemble des correspondances des 16 premiers entiers vers la base concernéé;
  - ou bien, faire des divisions successives du nombre donné par la base j'usqu'à obtenir un quotient égal à 0.
  - sinon, utiliser le tableau magique.

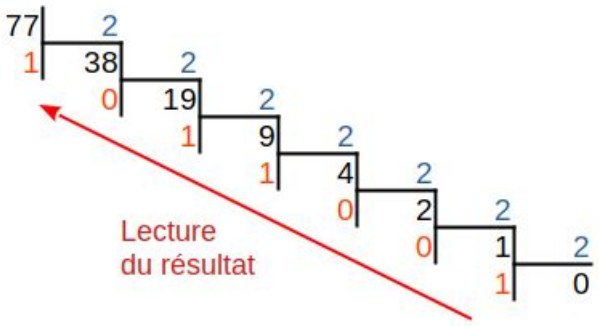
128	64	32	16	8	4	2	1	
1	0	0	0	0	0	1	0	= 130

Figure 1: *Méthode du tableau magique*

Déc.	Hexa.	Octal	Bin.
00	0	00	0000
01	1	01	0001
02	2	02	0010
03	3	03	0011
04	4	04	0100
05	5	05	0101
06	6	06	0110
07	7	07	0111
08	8	10	1000
09	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Figure 2: *tableau de correspondance*

- Par la méthode de divisions successives.



1. Par la base 2 :

3.2. Représentation des nombres flottants

Le passage de la base 10 à la base 2 est défini par :

- Partie entière est codée sur p bits (division successive par 2)
- Partie décimale est codée sur q bits en multipliant par 2 successivement jusqu'à ce que la partie décimale soit nulle ou le nombre de bits q est atteint.

Exemple : 4, 25 10 = ? (2) format virgule fixe :

- 4 10 = $100_{(2)}$

- 0, 25x2 = 0, 5 — 0
- 0, 5x2 = 1, 0 — 1.
- donc 4, 25 (10) = $100,01_{(2)}$

4. Représentation des nombres entiers signés

Nous venons de voir différentes représentations pour des entiers non signés, c'est-à-dire pour des entiers positifs. Nous allons présenter deux techniques permettant également la représentation d'entiers négatifs.

Pour un entier représenté sur n bits, les n 1 bits de poids faible sont réservés à la représentation de la valeur absolue de l'entier tandis que le bit de poids fort permet d'indiquer le signe de cet entier. Le bit de poids fort est à 1 si et seulement si l'entier est négatif.

4.1. Binaire Signé (BS)

La technique consiste à conserver le bit le plus significatif, le plus à gauche, pour représenter le signe du nombre :

- le bit le plus à gauche = 1, si le nombre est négatif;
- le bit le plus à gauche = 0, si le nombre est positif
- Les autres bits restants, codent la valeur absolue du nombre.

Exemple : Sur 8 bits, coder les nombres -24 et -128 et -100 en BS :

- 24 est codé en binaire signé par : $10011000_{(bs)}$
- -128 hors limite, car cela nécessite 9 bits au minimum
- 100 (10) est codé en binaire signé par : $11100100_{(bs)}$

4.2. Complément à un (CA1)

Aussi appelé Complément Logique (CL) ou Complément Restreint (CR) :

- Ne concerne que les nombre négatifs ;
- On inverse chaque bit de la représentation en valeur absolue
- Le bit le plus à gauche est utilisé pour représenter le signe :
  - le bit le plus fort, c'est-à-dire le plus à gauche = 1

Exemple : -24 en complément à 1 sur 8 bits :

- Abs(-24) en binaire pur = 00011000 2
- On inverse les bits = 11100111 pour obtenir le CA1

4.3. Complément à deux (CA2)

Aussi appelé Complément Vrai (CV), il concerne aussi les nombres négatifs :

- un nombre négatif est codé en ajoutant la valeur 1 à son CA1
- Le bit le plus significatif est utilisé pour représenter le signe.

Exemple : -24 en complément à 1 sur 8 bits :

- On converti 24 en binaire pur = 00011000_2
- On inverse les bits de la valeur $binaire = 11100111_{(ca1)}$
- On avjoute 1 au complément à $1 = 11101000_{(ca2)}$.

5. Transcodage

C'est la transformation de la représentation d'un nombre écrit dans une base donnée vers une autre représentation selon une base différente.

J'usqu'à présent nous avons vu comment représenter un nombre dans les bases courantes 2, 8, 10, 16 et nous avons retenu un principe commun pour représenter un nombre issu des bases courantes vers la base 10 en appliquant la formule suivante : $a_n....a_1a_0 = \sum_{i=0}^{n}(a_i * b^i)$

- $1010_{(2)} = 1_3 * 2^3 + 0_2 * 2^2 + 1_1 * 2^1 + 0 * 2^0 = 8 + 2 = 10$

Dans cette partie du cours, nous allons procéder aux différents conversions entre les bases 2, 8, 10 et 16.

5.1. Décimal vers le Binaire

- Décimal vers le Binaire
  - Exemple : Donnons l'écriture en binaire de 2395 par la méthode des divisions euclidiennes successives par la base (2), jusqu'à obtenir un quotient égal à 0.

5.2. Décimal vers l'Octal

- Décimal vers l'Octal
  - Exemple : Donnons l'écriture en Octal de 2395 par la méthode des divisions euclidiennes successives par la base (8), jusqu'à obtenir un quotient égal à 0. On relève les restes de bas en haut = $4533_{(8)}$

5.3. Décimal vers l'Hexadécimal

- Décimal vers l'Hexadécimal
  - Exemple : Donnons l'écriture en Hexadécimal de 2395 par la méthode des divisions euclidiennes successives par la base (8), jusqu'à obtenir un quotient égal à 0. On relève les restes de bas en haut = $95B_{(16)}$

5.4. Binaire vers le Décimal

- Binaire vers le Décimal

- Soit N un nombre représenté en binaire par :

  $N = 1010011101_{(2)}$

- Représentation Décimale?

$$N=1.2^9+0.2^8+1.2^7+0.2^6+0.2^5+1.2^4+1.2^3+1.2^2+0.2^1+1.2^0$$
$$=512+0+128+0+0+16+8+4+0+1$$
$$=669_{(10)}$$
$$1010011101_{(2)}=669_{(10)}$$

Figure 3: *Conversion binaire-décimal*

5.5. Binaire vers l'Hexadécimal

- Binaire vers l'Hexadécimal
  - L'idée est dans un premier temps, de pouvoir convertir un nombre provenant de la base 2 vers la base 16 à partir des exemples ci-dessous :
    - $11010011_{(2)} = (.....)_{(16)}$
    - $111110010110_{(2)} = (.....)_{(16)}$
  - La méthode consiste à regrouper 4 par 4 de la droite vers la gauche, les bits du nombre donné. Puis, de convertir les différents groupes en octal.
  - Maintenant dans le cas inverse, Hexadécimal vers le Binaire, il suffira de considérer chaque terme de l'octal comme étant la représentation d'un nombre binaire codé en hexadécimal.
    - Ainsi il suffira simplement de décoder chaque terme en un groupe de 4 bits.

- Soit N un nombre représenté en base binaire par :
  $N = 1010011101_{(2)}$
- Représentation Hexadécimale?

$$N = 0010\ \ 1001\ \ 1101_{(2)}$$
$$= 2\ \ 9\ \ D_{(16)}$$
$$1010011101_{(2)} = 29D_{(16)}$$

5.6. Octal vers l'Hexadécimal

- L'idée est de pouvoir convertir un nombre provenant de la base 8 vers la base 16 à partir des exemples ci-dessous :

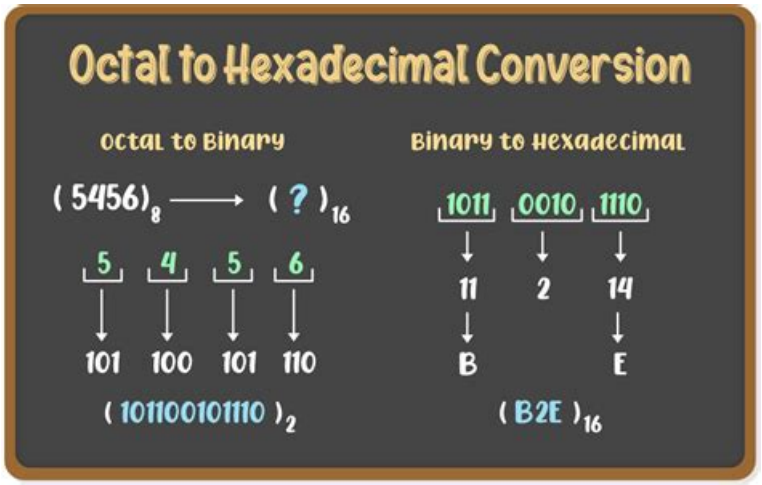
- $25_{(8)} = (.....)_{(16)}$
- $6401_{(8)} = (.....)_{(16)}$

- La méthode consiste à convertir le nombre donné en passant par le bianaire, puis du binaire à l'hexadécimal.
- Ainsi, d'après les principes déjà étudiés concernant la conversion en binaire, on applique la méthode associée.

• Octal vers l'Hexadécimal

Exemple : Donnons l'écriture en Hexadécimal du nombre Octal $1235_{(8)}$.

- Convertir chacun des chiffres en binaire :
- $1\ 2\ 3\ 5_{(8)} = 001_{(2)}\ 010_{(2)}\ 011_{(2)}\ 101_{(2)}$
- Faites des regroupements de quatre par quatre bits, de la droite vers la gauche :
- $1010011101_{(2)} = 0010_{(2)}\ 1001_{(2)}\ 1101_{(2)}$
- Convertir chaque groupe de quatre bits, par sa valeur en hexadécimal :
- $2\ 9\ D_{(16)}$



- La méthode consiste à convertir le nombre donné en passant par le bianaire, puis du binaire à l'hexadécimal et inversement.
- Ainsi, d'après les principes déjà étudiés concernant la conversion en binaire, on applique la méthode associée.

6. Représentation des caractères

En informatique, un caractère est une unité de base de texte. Il peut représenter une lettre, un chiffre, un signe de ponctuation, un espace, ou un autre symbole. Les caractères sont représentés numériquement par des nombres.

La représentation des caractères en informatique est un sujet complexe qui nécessite une compréhension des bases de l'informatique. Les codages de caractères sont essentiels pour la communication et le traitement du texte dans les systèmes informatiques.

Il existe de nombreux codages de caractères différents, chacun avec ses propres avantages et inconvénients.

6.1. Le codage Morse

Le codage en Morse des 26 lettres de l'alphabet latin, ainsi que des 10 chires décimaux. En réalité, le Morse peut aussi coder les principaux symboles de ponctuation, ainsi que certains caractères accentués.

a	·−	j	·−−−	s	···	1	·−−−−
b	−···	k	−·−	t	−	2	··−−−
c	−·−·	l	·−··	u	··−	3	···−−
d	−··	m	−−	v	···−	4	····−
e	·	n	−·	w	·−−	5	·····
f	··−·	o	−−−	x	−··−	6	−····
g	−−·	p	·−−·	y	−·−−	7	−−···
h	····	q	−−·−	z	−−··	8	−−−··
i	··	r	·−·	0	−−−−−	9	−−−−·

6.2. Le codage Baudot

Codage créé en 1874 par Émile Baudot, puis modifié et utilisé pour le réseau télégraphique commuté, connu aussi sous le nom CCIT, depuis 1917. Il permet de coder 57 caractères avec des codes de 5 bits.

6.3. Le codage ASCII

Code initié en 1963 puis complété en 1967, le code ASCII-1967 est devenu une norme ISO-646 en 1983. Il permet de coder 128 caractères, à l'origine sur 7 bits, puis sur huit bits.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	ESP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Le codage ASCII est l'un des codages de caractères les plus courants. Il utilise 7 bits pour représenter chaque caractère, ce qui permet de coder un total de 128 caractères. L'ASCII est suffisant pour représenter les caractères les plus courants des langues européennes, mais il ne permet pas de représenter les caractères des langues asiatiques ou des langues avec un alphabet étendu.

7. Circuits Logiques

7.1. Définition

1. Un circuit logique est un circuit électronique réalisant une ou plusieurs fonctions logiques. Un circuit logique est composé de :

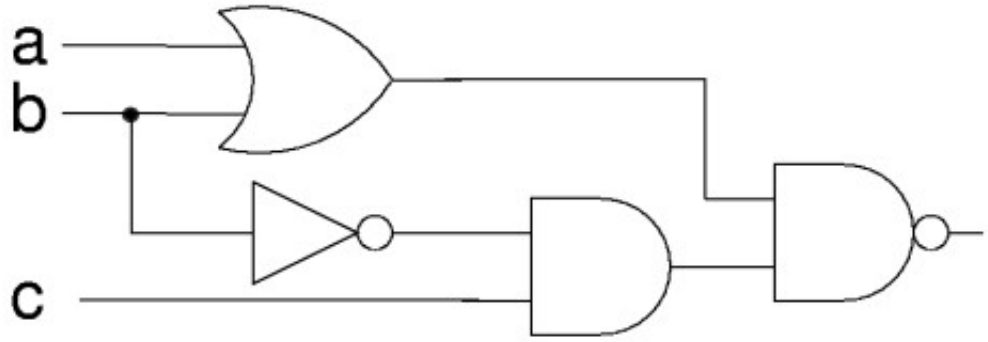
- D'un ensemble de portes logiques
- De circuits logiques
- Le tout interconnectés entre eux

Il existe deux(2) types de circuits logiques :

- Circuits combinatoires : S = f(E)
- Circuits séquentiels : notion d'état et de mémoire

Un circuit se représente par un logigramme, constitué de portes logiques de base :

- Exemple de circuit logique
  - 3 entrées, 1 sortie
  - Composé uniquement de portes logiques



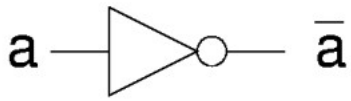
2. Une porte logique est un circuit combinatoire de base réalisant une opération logique de base. Exemple : OU, ET, NON, correspondant aux opérateurs de l'algèbre de Boole. Une porte logique possède :
   - Une table de vérité et/ou une expression logique définissant son résultat en fonction de son/ses entrée(s)
   - Un symbole graphique.

7.2. Portes Logiques de base

NON ou NOT : Est une fonction logique possédant :

- 1 entrée, 1 sortie
- noté A/

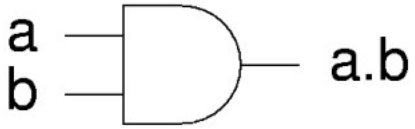
a	$\overline{a}$
0	1
1	0



ET ou AND : Est une fonction logique possédant :

- 2 entrées, 1 sortie
- a ET b est noté a.b ou ab

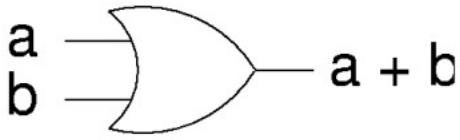
a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1



OU ou OR : Est une fonction logique possédant :

- 2 entrées, 1 sortie
- a OU b est noté a + b

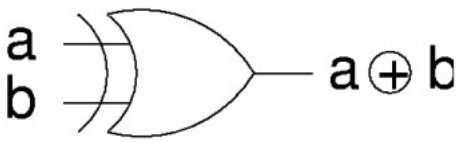
a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	1



OU-Exclusif ou XOR, Est une fonction logique possédant :

- 2 entrées, 1 sortie
- a OU-exclusif b est noté a (+) b

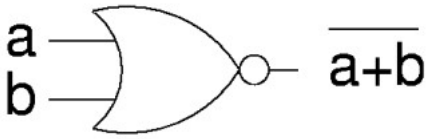
a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0



NON-OU ou NOR, Est une fonction logique possédant :

- 2 entrées, 1 sortie
- a NOR b est noté (a + b)/

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0



7.3. Règles et Propriétès de l'Algèbre de Boole

Définit en 1847 par Georges Boole (1815-1864), physicien Anglais. Algèbre applicable au raisonnement logique qui traite des fonctions à variables binaires (deux valeurs). Ne s'applique pas aux systèmes à plus de deux états d'équilibre. Permet d'étudier les circuits logiques (un système logique sert à modifier des signaux) L 'algèbre de Boole permet de manipuler des valeurs logiques :

- Une valeur logique n'a que deux états possibles, Vraie(1) ou Fausse(0).
- Plusieurs valeurs logiques peuvent être combinées pour donner un résultat qui est lui aussi une valeur logique

Exemple :

1. arrêt marche
2. ouvert fermé
3. enclenché déclenché
4. avant arrière
5. vrai faux
6. conduction blocage

7.4. Théorème de De Morgan

- $\overline{A+B} = \overline{A}.\overline{B}$

Vérification :

A	B	$\overline{A+B}$	$\overline{A}.\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Equivalent

- $\overline{A.B} = \overline{A}+\overline{B}$

Vérification :

A	B	$\overline{A.B}$	$\overline{A}+\overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Equivalent

$$S = A \cdot B \cdot C + A \cdot \overline{B} \cdot \left(\overline{\overline{A} \cdot \overline{C}}\right)$$

- **Transformation**

$$S = A \cdot B \cdot C + A \cdot \overline{B} \cdot (A + C)$$
$$= A \cdot B \cdot C + A \cdot \overline{B} \cdot A + A \cdot \overline{B} \cdot C$$
$$= A \cdot B \cdot C + A \cdot \overline{B} + A \cdot \overline{B} \cdot C$$

- **Variables communes**

$$S = A \cdot \overline{B} + A \cdot C \cdot \left(B + \overline{B}\right)$$
$$S = A \cdot \overline{B} + A \cdot C$$
$$\boxed{S = A \cdot \left(\overline{B} + C\right)}$$

7.5. Simplification des fonctions et Circuits logiques

1. A partir d'une fonction logique, on peut trouver le logigramme correspondant à cette fonction. Le principe est de simplifier la fonction logique à partir de l'une des 2 méthodes :

   - La méthode algébrique (algèbre de Boole)
   - La méthode des tableaux de Karnaugh

   et d'en déduire après le le logigramme correspondant.

2. A partir du logigramme d'un circuit, on peut trouver sa fonction logique. Le principe est de :

   - Donner l'expression des sorties de chaque porte/composant en fonction des valeurs de ses entrées.
   - En déduire au final la (ou les) fonction(s) logique(s) du circuit

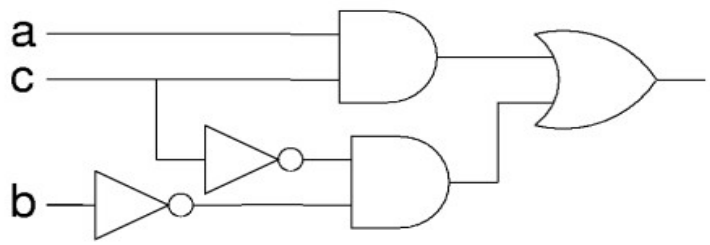
   Puis, par la suite, déterminer la table de vérité du circuit en simplifiant la fonction logique à l'aide des propriétés de l'algèbre de Boole ou les tableaux de Karnaugh.

◆ Soit la fonction

$$f(a, b, c) = abc + a\overline{b}\,\overline{c} + \overline{a}\,\overline{b}\,\overline{c} + a\overline{b}\,c$$

◆ Après simplification, on obtient

$$f(a, b, c) = ac + \overline{b}\,\overline{c}$$



8. Programmation Python

- Python a été créé par Guido van Rossum et a été publié pour la première fois en 1991.
- Il suit les principes du Zen de Python, qui mettent l'accent sur la lisibilité du code et la simplicité.
    1. Syntaxe claire et expressive qui favorise la lisibilité.
    2. Une vaste communauté de développeurs qui contribuent constamment à l'écosystème;
    3. Les types de données sont déterminés à l'exécution;
    4. Les types sont stricts, mais la conversion entre types est souvent implicite.

Dans cette partie du cours, nous allons explorer quelques notions de base :

1. les variables et fonctions, les structures de contrôle,
2. les structures de données, les modules matplotlib et numpy.

8.1. Les bases

- En Python, une variable est un espace réservé pour stocker des valeurs. C'est un moyen d'associer un nom (le nom de la variable) à une valeur. Les variables sont utilisées pour stocker des données de différentes natures, comme des nombres, des chaînes de caractères, des listes, des objets, etc.

8.1.1 Variables et Fonctions

- En Python, vous n'avez pas besoin de déclarer explicitement le type d'une variable. Le type est déterminé dynamiquement lors de l'assignation.
    - Variable entière : nombre = 42
    - Variable chaîne de caractères : nom = "John"
    - Variable à virgule flottante : pi = 3.14
    - Variable booléenne : est_vrai = True
    - Variable liste : liste_nombres = [1, 2, 3]
- **Utilisation des variables :**
    - Vous pouvez utiliser les variables dans des expressions, les modifier et les manipuler comme suit :
        * a = 10
        * b = 20
        * somme = a + b
        * print(somme)
        * nom = "Alice"
        * nom = nom + " Smith"
        * print(nom)
        * nombres = [1, 2, 3]
        * nombres.append(4)
        * print(nombres)
- **Lambda :**

  - Elles sont utilisées pour definir des fonctions mathématiques :
    * f = lambda x : 2*x + 5
    * print(f(5))

- **Personnalisées :**
  - Elles sont définies avec le mot-clé **def** :
    * def multiplication(x, y):
    * return x * y
    * result = multiplication(3, 4)
    * print(result)
  - Ou encore :
    * def afficher_personne(nom, age):
    * print(f"Nom : nom, Age : age")
    * afficher_personne(age=25, nom="Alice")

8.1.2 Structures de contrôle

- **Boucles :**
  - Les boucles **for** sont utilisées pour l'itération, par exemple :
    * for fruit in range(10):
    * print(i)
  - Les boucles **while** sont aussi utilisées pour l'itération :
    * count = 0
    * while count <5 :
    * print(count)
    * count += 1

- **Conditions :**
  - Les instructions **if, elif,** et **else** permettent le contrôle conditionnel :

```
        if x > 0:
            print("x est POSITIF")
        elif x == 0:
            print("x est NUL ")
        else:
           print("x est NEGATIF ")
```

8.1.3 Structure de données

- **Les listes et les Tuples :**
  - Les listes et les tuples sont des structures de données séquentielles:
    * nombres = [1, 2, 3, 4, 5] (liste)
    * coordonnees = (3, 4) (tuple)

- **Les Dictionnaires :**

  - Les dictionnaires sont des paires clé-valeur :
    * personne = "nom": "Alice", "age": 30, "ville": "Paris"

Ces exemples illustrent l'utilisation de diverses structures de contrôle, de fonctions et de structures de données en Python. N'hésitez pas à les tester dans un environnement Python pour mieux comprendre leur fonctionnement.

8.1.4 Matplotlib et Numpy

- Dans ce cours nous allons abordés les notions de base de la programmation Python pour faire du calcul matriciel.
- Au coeur de cette formation, nous utiliserons un package très important pour le calcul matriciel, appelé Numpy.
- Dans ce package, on exploitera les tableaux à N-dimensions (ND-array).
- ndarray est beaucoup plus rapide que les listes et meilleur pour le calcul scientifique.
  1. cela occupe moins de place en mémoire
  2. cela permet d'executer des programmes plus rapidement.
- shape :
  - L'attribut shape de ndarray, permet d'obtenir les dimensions d'un tableau (matrice):
    1. shape : renvoie la dimension du tableau (matrice);
    2. shape[0] : renvoie le nombre de ligne du tableau (matrice)
    3. shape[1] : renvoie le nombre de colonne du tableau (matrice).

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

  - A.shape() = (3,4)
  - A.shape[0] = 3
  - A.shape[1] = 4

8.2. Installation des packages

- Comment installer des packages sur une machine hors ligne :
  1. Sur une autre machine disposant d'une connexion internet, téléchargez les packages nécessaires et leurs dépendances (ex : pip download numpy pandas)
  2. A partir d'une clés USB par exemple, transférez les fichiers téléchargés vers la machine sans connexion internet.
  3. Sur la machine sans connexion internet, utilisez pip pour installer les packages à partir des fichiers téléchargés. Par exemple :
     - pip install numpy-1.21.2-cp39-cp39-win_amd64.whl
     - pip install matplotlib-1.3.3-cp39-cp39-win_amd64.whl
  4. Assurez-vous de remplacer les noms de fichiers par ceux que vous avez réellement téléchargés.
- Cela devrait vous permettre d'installer Python avec les paquets nécessaires sur une machine sans connexion internet.

9. TDs / TPs :

1. TD1 : Convertir en binaire les nombres suivants
   - $10_{(10)} = (.....)_{(2)}$
   - $100_{(10)} = (.....)_{(2)}$
   - $10_{(10)} = (.....)_{(8)}$
   - $100_{(10)} = (.....)_{(8)}$
   - $10_{(10)} = (.....)_{(16)}$
   - $100_{(10)} = (.....)_{(16)}$
2. TD2 : Transcodage

Table 3: *Table de conversion*

Nombre	Binaire	Octal	Héxadécimal
00101101_2	00101101_2		
00037_8		00037_8	
$F2_16$			$F2_16$

1. **TP1 :** Écrire un programme en Python, permettant de convertir vers une base b donnée, un entier n. Le calcul des chiffres de l'écriture d'un entier naturel n en base b s'effectue par des divisions euclidiennes successives par b, en partant de l'entier n jusqu'à obtenir un quotient plus petit que b. L'algorithme ci-dessous montre comment obtenir cette écriture.

```
LIRE : b la base de numeration, n un entier naturel.
AFFICHER : x_0 , x_1 , .... , x_(p-1)
m := n, i := 0
tant que m >= b faire
    r := m (mod b)
    m := m / b
    x_i := r
    i := i + 1
fin tant que
x_i := m
renvoyer x_0 , x_1 , . . . , x_i .
```

2. **TP2 :** Calcul d'un entier à partir de son écriture dans une base. A partir de la connaissance de l'écriture dans une base b d'un entier n, écrire en Python, un programme permettant de calculer cet entier par une évaluation de la somme

```
LIRE : b la base de numeration d'un entier.
LIRE : x_0 , x_1 , .... , x_(p-1) l'ecriture en base b d'un entier.
n := l'entier correspondant au chiffre x 0
pour k variant de 1 A p-1 faire
    n := b*n+ x_k
fin pour
afficher n
```