

# Codage et Assembleur

- 1- Codage
- 2- Fonctions et structure d'une machine
- 3- Algèbre de Boole
- 4- Les microprocesseurs
- 5- Programmation
- 6- Procédure, récursivité et interruptions
- 7- Assembleur ARM 7

# Ch 1 - CODAGE

## **1 Généralités**

1.1 Notion d'information

1.2 Principe de codification

1.3 Notion de langage de codification

1.4 Codage des valeurs booléennes

1.5 Représentation des entiers naturels en base  $b$

1.6 Codages des caractères

1.7 Codages des chiffres décimaux

Détection et correction d'erreurs

# Chapitre 1 - CODAGE

## **2 Représentation des nombres**

2.1 Entiers naturels

2.2 Nombres positifs à partie fractionnaire

2.3 Entiers relatifs en binaire

2.4 Réels en virgule fixe

2.5 Réels en virgule flottante

# 1.1 Notion d'information

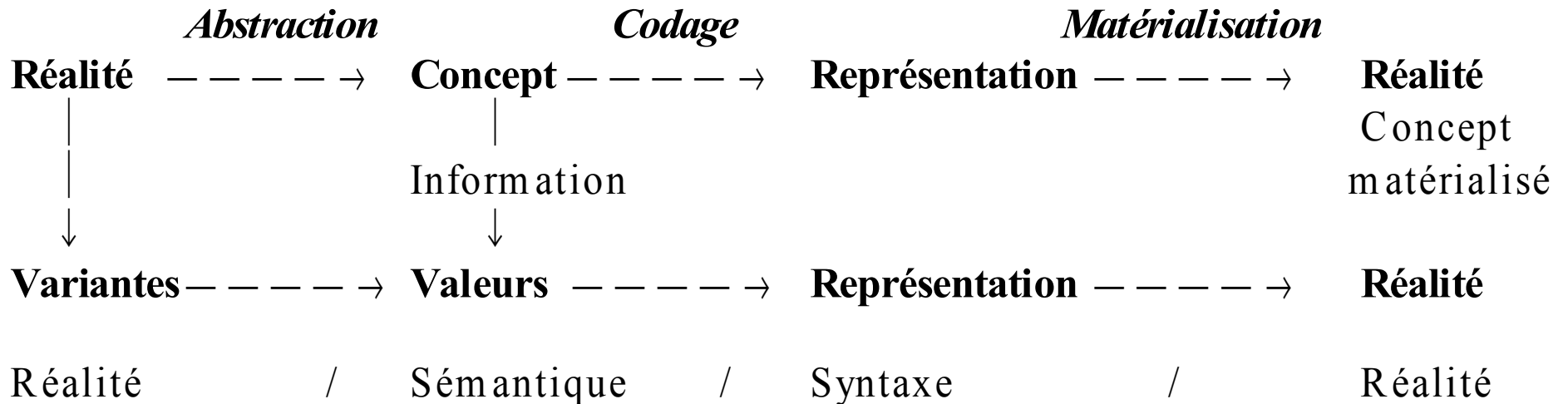
- Renseignement, nouvelle : Radio, TV, Presse...
- Renseigne un observateur sur l'état d'un système
- Porte sur les différentes valeurs d'un concept

Exemples de concept :

couleur, nom, âge, température, vitesse...

# 1.1 Notion d'information (2)

## Genèse d'un concept



# 1.2 Principe de codification

**V** champ sémantique (ensemble de valeurs d'un concept)

**C** un ensemble de valeurs de code (représentation)

Un codage est une relation **R** sur **V**x**C**

Un codage fonctionnel est une **application injective**

$$\text{code} : \mathbf{V} \dashrightarrow \mathbf{C} ; \underline{v} \dashrightarrow c = \text{code}(\underline{v})$$

Toute valeur sémantique a sa propre valeur de code

Exemple : l'entier 19

19	codage en base 10
10011	codage en base 2
dix-neuf	codage français
XIX	codage romain

# 1.3 Notion de langage de codification

Un **langage**  $L$  est défini par :

- un **alphabet**  $A$  (ensemble de symboles)  
 $b = |A|$  cardinal de  $A$  est la **base** du langage
- un **format**  $n$  (nombre de symboles) d'un mot  
un **mot** étant une suite de symboles
- des règles de construction des mots  
**contraintes lexicographiques** ou syntaxiques

La **puissance lexicographique** d'un langage  $L$  est le cardinal de l'ensemble de ses mots, notée  $|L|$

# 1.3 Notion de langage de codification

## **Puissance lexicographique des langages sans contrainte**

Langage  $L$  de **format fixe**  $n$  : mots de longueur imposée  $n$

$$|L| = b^n$$

Langage de **format variable**  $n$  : mots dont la longueur varie de 1 à  $n$

$$|L| = \sum_{i=1}^n b^i$$

Le **mot vide** à la longueur 0

La **réunion**  $L$  de  $p$  langages disjoints  $L_i$

$$|L| = \sum_{i=1}^p |L_i|$$

La **concaténation**  $L$  de langages  $L_i$

$$|L| = \prod_{i=1}^p |L_i|$$



# 1.4 Codage des valeurs de vérité

La plus petite information : deux valeurs  $\{\underline{v}_1, \underline{v}_2\}$

$\underline{v}_1$	$\underline{v}_2$
non	oui
faux	vrai
false	true
0	1

Codage **binaire** ou codage **booléen**,  
les valeurs sont les valeurs booléennes **bits**  $\{0,1\}$

# 1.5 Représentation des entiers naturels en base $b$

## Système de numération en base $b$

base dix             $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

base deux           $\{0, 1\}$

base seize          $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Représenter  $p$  entiers naturels de 0 à  $p-1$  en format fixe  $n$

$$b^{n-1} < p \leq b^n \quad \text{ou} \quad n = \text{Entiersup}(\log_b p)$$

La valeur  $\underline{v}$  est représentée par la suite de chiffres

$$v = c_{n-1}c_{n-2}\dots c_1c_0 \quad \text{et} \quad \underline{v} = \sum_{i=0}^{n-1} \underline{c}_i \underline{b}^i$$

# 1.6 Codages des caractères

Les caractères sont en général ordonnés (chiffres, lettres...)

1- Constituer une liste des  $p$  caractères utilisés

2- Coder un caractère par son rang de 0 à  $p-1$

Les codes informatiques utilisent généralement un **format fixe**

Certains codes sont **contextuels** (code Baudot 5 bits)

Les codes **7 bits ASCII**, ISO... sont les plus utilisés  
ainsi que des extensions **8 bits** et **16 bits** du code ASCII

Pages de codes, codes UTF

Des valeurs numériques sont représentées avec ces codes

Il est difficile d'effectuer des calculs sur les représentations

# Code télégraphique international 5 bits

	LETTRES	CHIFFRES	CODE						LETTRES	CHIFFRES	CODE				
1	A	—	1	1	0	0	0	23	W	2	1	1	0	0	1
2	B	?	1	0	0	1	1	24	X	/	1	0	1	1	1
3	C	:	0	1	1	1	0	25	Y	6	1	0	1	0	1
4	D	¶	1	0	0	1	0	26	Z	+	1	0	0	0	1
5	E	3	1	0	0	0	0	27	Retour du chariot		0	0	0	1	0
6	F	É	1	0	1	1	0	28	Changement de ligne		0	1	0	0	0
7	G	%	0	1	0	1	1	29	Lettres		1	1	1	1	1
8	H	H	0	0	1	0	1	30	Chiffres		1	1	0	1	1
9	I	8	0	1	1	0	0	31	Espace		0	0	1	0	0
10	J	Déclenche- ment sonnerie	1	1	0	1	0	32	Non employé		0	0	0	0	0
11	K	(	1	1	1	1	0								
12	L	)	0	1	0	0	1								
13	M	.	0	0	1	1	1								
14	N	,	0	0	1	1	0								
15	O	9	0	0	0	1	1								
16	P	0	0	1	1	0	1								
17	Q	1	1	1	1	0	1								
18	R	4	0	1	0	1	0								
19	S	'	1	0	1	0	0								
20	T	5	0	0	0	0	1								
21	U	7	1	1	1	0	0								
22	V	=	0	1	1	1	1								

Sym- boles	Transmission par		Ruban per- foré
	Circuit ouvert ou fermé	Double courant	
0	Pas de courant	Courant négatif	Trou absent
1	Courant positif	Courant positif	Trou présent

# Code Sixbit à 6 bits

CARACTÈRE	CODE	CARACTÈRE	CODE
0	0 0 0 0 0 0	Blanc	0 1 0 0 0 0
1	0 0 0 0 0 1	/	0 1 0 0 0 1
2	0 0 0 0 0 1 0	S	0 1 0 0 0 1 0
3	0 0 0 0 0 1 1	T	0 1 0 0 0 1 1
4	0 0 0 0 1 0 0	U	0 1 0 0 1 0 0
5	0 0 0 0 1 0 1	V	0 1 0 0 1 0 1
6	0 0 0 0 1 1 0	W	0 1 0 0 1 1 0
7	0 0 0 0 1 1 1	X	0 1 0 0 1 1 1
8	0 0 1 0 0 0 0	Y	0 1 1 0 0 0 0
9	0 0 1 0 0 0 1	Z	0 1 1 0 0 0 1
Espace	0 0 1 0 0 1 0	#	0 1 1 0 0 1 0
=	0 0 1 0 0 1 1	.	0 1 1 0 0 1 1
Apostrophe	0 0 1 1 0 0 0	(	0 1 1 0 1 0 0
>	0 0 1 1 1 1 1	-	0 1 1 0 1 0 1
√	0 0 1 1 1 1 0	/	0 1 1 0 1 1 0
		Annulation	0 1 1 0 1 1 1
—	1 0 0 0 0 0 0	+	1 1 0 0 0 0 0
J	1 0 0 0 0 0 1	A	1 1 0 0 0 0 1
K	1 0 0 0 0 1 0	B	1 1 0 0 0 1 0
L	1 0 0 0 0 1 1	C	1 1 0 0 0 1 1
M	1 0 0 0 1 0 0	D	1 1 0 0 1 0 0
N	1 0 0 0 1 0 1	E	1 1 0 0 1 0 1
O	1 0 0 0 1 1 0	F	1 1 0 0 1 1 0
P	1 0 0 0 1 1 1	G	1 1 0 0 1 1 1
Q	1 0 1 0 0 0 0	H	1 1 1 0 0 0 0
R	1 0 1 0 0 0 1	I	1 1 1 0 0 0 1
!	1 0 1 0 0 1 0	?	1 1 1 0 0 1 0
\$	1 0 1 0 0 1 1	.	1 1 1 0 0 1 1
*	1 0 1 0 1 0 0	)	1 1 1 0 1 0 0
]	1 0 1 0 1 0 1	[	1 1 1 0 1 0 1
;	1 0 1 0 1 1 0	<	1 1 1 0 1 1 0
Δ	1 0 1 0 1 1 1	≠	1 1 1 0 1 1 1

# Code ISO à 7 bits

valeur des éléments							
binaires successifs							
eb	eb	eb	eb	eb	eb	eb	eb
7	6	5	4	3	2	1	
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	1
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	1
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	1
1	0	1	1	0	1	1	1
1	1	0	0	0	1	1	1
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1

0	1	2	3	4	5	6	7
NUL	ESPACE	0	à	P	/	p	
1	1	1	A	Q	a	q	
»	2	B	R	b	r		
\$	3	C	S	c	s		
£	4	D	T	d	t		
%	5	E	U	e	u		
&	6	F	V	f	v		
Apostrophe	7	G	W	g	w		
(	8	H	X	h	x		
)	9	I	Y	i	y		
*	:	J	Z	j	z		
+	;	K	Degré °	k	é		
,	<	L	ç	l	ù		
—	=	M	§	m	è		
.	>	N	^	n	~		
/	?	O	Souligné —	o	Annulation		

Codes de commande réservés à certaines fonctions spécifiques de la transmission et à certains délimiteurs d'informations.

Sortie du code

Entrée dans le code



# Code EBCDIC d'IBM

		Équivalent des quatre éléments binaires de forts poids															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Équivalent des quatre éléments binaires de faibles poids	0	Nul				Blanc	&	—									0
	1									a	j			A	J		1
	2									b	k	s		B	K	S	2
	3									c	l	t		C	L	T	3
	4	Codes de commande								d	m	u		D	M	U	4
	5									e	n	v		E	N	V	5
	6									f	o	w		F	O	W	6
	7									g	p	x		G	P	X	7
	8									h	q	y		H	Q	Y	8
	9									i	r	z		I	R	Z	9
	10					#	!	.	:								
	11					.	\$	.	#								
	12					<	.	%	@								
	13					(	)	—	'								
	14					+	;	>	=								
	15					/	7	?	.								

# Code ASCII et page de code USA

0	32	64	96	128	160	192	224
1	33	65	97	129	161	193	225
2	34	66	98	130	162	194	226
3	35	67	99	131	163	195	227
4	36	68	100	132	164	196	228
5	37	69	101	133	165	197	229
6	38	70	102	134	166	198	230
7	39	71	103	135	167	199	231
8	40	72	104	136	168	200	232
9	41	73	105	137	169	201	233
10	42	74	106	138	170	202	234
11	43	75	107	139	171	203	235
12	44	76	108	140	172	204	236
13	45	77	109	141	173	205	237
14	46	78	110	142	174	206	238
15	47	79	111	143	175	207	239
16	48	80	112	144	176	208	240
17	49	81	113	145	177	209	241
18	50	82	114	146	178	210	242
19	51	83	115	147	179	211	243
20	52	84	116	148	180	212	244
21	53	85	117	149	181	213	245
22	54	86	118	150	182	214	246
23	55	87	119	151	183	215	247
24	56	88	120	152	184	216	248
25	57	89	121	153	185	217	249
26	58	90	122	154	186	218	250
27	59	91	123	155	187	219	251
28	60	92	124	156	188	220	252
29	61	93	125	157	189	221	253
30	62	94	126	158	190	222	254
31	63	95	127	159	191	223	255



# Code ASCII page de code Latin

0	32	64	Q	96	`	128	Ç	160	Á	192	Ł	224	Ó
1	☐	33	!	65	À	97	a	129	Ā	193	Ł	225	Ô
2	☐	34	"	66	B	98	b	130	É	194	Ť	226	Õ
3	♥	35	#	67	C	99	c	131	Â	195	Ţ	227	Ö
4	♦	36	\$	68	D	100	d	132	Ä	196	—	228	Ø
5	♣	37	%	69	E	101	e	133	À	197	†	229	Ō
6	♠	38	&	70	F	102	f	134	Å	198	ã	230	µ
7	•	39	'	71	G	103	g	135	Ş	199	ă	231	þ
8	☐	40	(	72	H	104	h	136	Ê	200	ü	232	þ
9	◊	41	)	73	I	105	i	137	Ë	201	ŋ	233	ú
10	☐	42	*	74	J	106	j	138	È	202	μ	234	û
11	♂	43	+	75	K	107	k	139	Ï	203	ŋ	235	ù
12	♀	44	,	76	L	108	l	140	Î	204	ŋ	236	ý
13	♂	45	-	77	M	109	m	141	Ì	205	—	237	ÿ
14	♂	46	.	78	N	110	n	142	Ā	206	ŋ	238	·
15	*	47	/	79	O	111	o	143	Â	207	Ō	239	˘
16	▶	48	0	80	P	112	p	144	É	208	ð	240	·
17	◀	49	1	81	Q	113	q	145	×	209	Đ	241	±
18	±	50	2	82	R	114	r	146	ff	210	Ê	242	=
19	!!	51	3	83	S	115	s	147	ô	211	Ë	243	¼
20	¶	52	4	84	T	116	t	148	ö	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	213	Ì	245	§
22	—	54	6	86	V	118	v	150	û	214	Í	246	+
23	±	55	7	87	W	119	w	151	ù	215	Î	247	˘
24	↑	56	8	88	X	120	x	152	ÿ	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	ÿ	217	Ĵ	249	˙
26	→	58	:	90	Z	122	z	154	ÿ	218	Ť	250	·
27	←	59	;	91	[	123	{	155	œ	219	■	251	ı
28	└	60	<	92	\	124		156	£	220	■	252	ž
29	+	61	=	93	]	125	}	157	Œ	221	ı	253	z
30	▲	62	>	94	^	126	~	158	×	222	ÿ	254	■
31	▼	63	?	95	_	127	Δ	159	f	223	■	255	

# 1.7 Codages des chiffres décimaux

**Codes spéciaux** sur 4 bits ou plus

DCB, STIBITZ, GRAY, 2 parmi 5...

Codes pondérés

Représentation des entiers par concaténation des codes des chiffres

On construit ainsi des représentations de représentations

Des bits supplémentaires sont ajoutés aux représentations pour la **détection** et la **correction d'erreurs** de transmission

Bits de parité

Codes linéaires

Distance de Hamming

# Code DCB : Décimal Codé Binaire

Chiffres	Représentation DCB
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

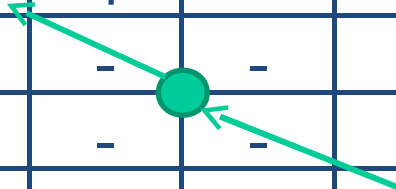
# Code STIBITZ excès de trois

Chiffres	Représentation STIBITZ
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

# Code Acken

Table du code Acken

	b4	0	0	1	1
b2	b1 \ b3	0	1	0	1
0	0	0	4	-	6
0	1	1	-	-	7
1	0	2	-	-	8
1	1	3	-	5	9



Point de symétrie

# Les codes décimaux pondérés

- **Codes pondérés**

La valeur  $c$  du chiffre représenté est

$$c = a_1.b_1 + a_2.b_2 + a_3.b_3 + a_4.b_4$$

$a_1, a_2, a_3, a_4$  sont les poids du code

$b_1, b_2, b_3, b_4$  sont les bits de la représentation

- **Systèmes de poids**

1,1,2,5 – 1,1,3,4 – 1,1,3,5 – 1,1,3,6 – 1,2,2,4 – 1,2,2,5 –  
1,2,2,6 – 1,2,3,3 – 1,2,3,4 – 1,2,3,5 – 1,2,3,6 – 1,2,3,7 –  
1,2,4,4 – 1,2,4,5 – 1,2,4,6 – 1,2,4,7 – 1,2,4,8.

- Des poids peuvent être négatifs (6, 3, 1, -1)

# Codes décimaux pondéré

## code 5, 2, 1, 1

Comment est représenté le nombre 1789 avec le code de poids 5, 2, 1,1 ? Vérifier le code et coder.

Chiffres \ Poids	5	2	1	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	1	0	0
8	1	1	1	0
9	1	1	1	1

1789

0001 1100 1110 1111

# Propriétés des codes décimaux

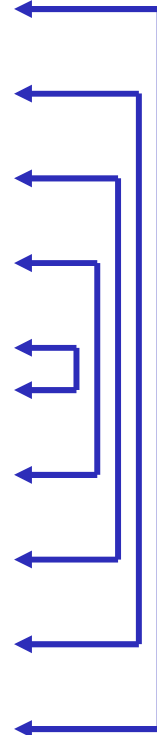
- Les **codifications à suite ordonnée**  
L'assignation sémantique respecte l'ordre lexicographique des combinaisons croissant ou décroissant.
- Les **codifications à suite naturelle**  
Ce sont des codes à suite ordonnée composée des successeurs ou des prédécesseurs.
- Les **codifications auto-complémentaires**  
La représentation du complément à 9 d'un chiffre décimal  $c$  s'obtient en complémentant tous les bits de la représentation du chiffre initial  $c$ .



# Code décimal pondéré 6, 3, 1, -1

Exemple de codification auto-complémentaire

Chiffres \ Poids	6	3	1	-1
0	0	0	1	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	0	1	1	0
5	1	0	0	1
6	1	0	0	0
7	1	0	1	0
8	1	1	0	1
9	1	1	0	0



# Contrôle automatique de parité

## Code DCB avec parité

Chiffres	Représentation DCB	Clé
0	0000	1
1	0001	0
2	0010	0
3	0011	1
4	0100	0
5	0101	1
6	0110	1
7	0111	0
8	1000	0
9	1001	1

# Contrôle de parité croisée

Clés de parité transversale ↓

1 <sup>er</sup> chiffre :	0	0	0	0	1
2 <sup>ème</sup> chiffre :	0	1	0	1	1
3 <sup>ème</sup> chiffre :	0	1	1	1	0
4 <sup>ème</sup> chiffre :	1	0	0	1	1
5 <sup>ème</sup> chiffre :	0	0	1	1	0
6 <sup>ème</sup> chiffre :	0	1	0	0	0
7 <sup>ème</sup> chiffre :	0	0	0	0	1
8 <sup>ème</sup> chiffre :	0	0	1	0	0
parité longitudinale →	0	1	0	1	1

# Code auto-correcteur de Hamming

- M bits de donnée  $m_i$  et K bits de contrôle  $k_j$   
K décrit  $M + K + 1$  situations : message correct et  $M + K$  positions d'erreur donc K vérifie  $2^k \geq M + K + 1$
- Pour  $M = 4$ , il suffit de prendre  $K = 3$ 
  - $m_4 m_3 m_2 k_3 m_1 k_2 k_1$
  - $k_3$  contrôle la parité de  $m_4 m_3 m_2$
  - $k_2$  contrôle la parité de  $m_4 m_3 m_1$
  - $k_1$  contrôle la parité de  $m_4 m_2 m_1$

**Tester 1 bit erroné puis 2**

# Codes linéaires

**Exemple : un seul bit de données  $m$  et  $K$  bits de contrôle  $k_j$  redondants**

Il n'y a que 2 messages corrects : 0...0 et 1...1

Les erreurs sont corrigées en prenant le message correct le plus proche suivant la distance de Hamming (nombre de bits différents).

**Pour  $K = 1, 2, 3$  et 4**

**déterminer le nombre d'erreurs détectées et le nombre d'erreurs corrigées**

## 2.1 Les entiers naturels

### Codage en base $b$ :

Un entier naturel  $\underline{v}$  est représenté par la suite de  $n$  chiffres

$$v = (c_{n-1}c_{n-2}\dots c_1c_0)_b \quad \text{et} \quad \underline{v} = \sum_{i=0}^{n-1} \underline{c_i} \underline{b^i}$$

Exemple : 1996 est codé

$(11111001100)_2$

en binaire

$(2201221)_3$

en ternaire

$(3714)_8$

en octal

$(1996)_{10}$

en décimal

$(7CC)_{16}$

en hexadécimal

Nombre \ Base	2	8	16
0	<b>0</b>	<b>0</b>	<b>0</b>
1	<b>1</b>	<b>1</b>	<b>1</b>
2	<b>10</b>	<b>2</b>	<b>2</b>
3	<b>11</b>	<b>3</b>	<b>3</b>
4	<b>100</b>	<b>4</b>	<b>4</b>
5	<b>101</b>	<b>5</b>	<b>5</b>
6	<b>110</b>	<b>6</b>	<b>6</b>
7	<b>111</b>	<b>7</b>	<b>7</b>
8	<b>1000</b>	10	<b>8</b>
9	<b>1001</b>	11	<b>9</b>
10	<b>1010</b>	12	<b>A</b>
11	<b>1011</b>	13	<b>B</b>
12	<b>1100</b>	14	<b>C</b>
13	<b>1101</b>	15	<b>D</b>
14	<b>1110</b>	16	<b>E</b>
15	<b>1111</b>	17	<b>F</b>
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13

## Conversion base $b$ base $b^k$

- Regrouper les chiffres par  $k$  en commençant par les unités
- Remplacer chaque groupe par le chiffre en base  $b^k$

$$\begin{aligned}
 (19)_{10} &= (010\ 011)_2 \\
 &= (23)_8 \\
 &= (0001\ 0011)_2 \\
 &= (13)_{16}
 \end{aligned}$$

## Conversion base 10 base $b$ (partie entière)

Algorithme

$i := 0;$

répéter  $r_i := x \bmod b; x := x \operatorname{div} b; i := i + 1;$

jusqu'à  $x = 0;$

$n := i;$

La suite des restes donne la représentation de  $x$  en base  $b$

$$x = (r_{n-1}r_{n-2}\dots r_1r_0)_b$$

Le **dernier reste** trouvé correspond au chiffre le **plus significatif**

## Conversion base $b$ base 10

Développer la somme  $x = \sum_{i=0}^{n-1} \underline{c}_i \underline{b}^i$  avec l'algorithme de Horner

$$(2345)_8 = (((2) \times 8 + 3) \times 8 + 4) \times 8 + 5$$



# Passage de base 10 à base b

Example :  $(131)_{10} = (?)_2$

Diagram illustrating the conversion of the decimal number 131 to binary using the division-by-2 method. The divisions are arranged in a staircase pattern from top-left to bottom-right. Each step shows a division of a number by 2, with the quotient and remainder. The remainders, read from bottom-right to top-left, form the binary number 10000011. A blue arrow points from the bottom-right division to the top-left division.

131		2									
1		65		2							
	1		32		2						
		0		16		2					
			0		8		2				
				0		4		2			
					0		2		2		
						0		1		2	
							1		0		

(131)<sub>10</sub> = (10000011)<sub>2</sub>

$$(131)_{10} = (10000011)_2$$



# Passage de base b à base 10

Exemple :  $(\text{FEAC9B})_{16} = (?)_{10}$

Il suffit de développer  $\sum_{i=0}^{n-1} \underline{c_i} \underline{b^i}$

$$\begin{aligned}(\text{FEAC9B})_{16} &= (11*16^0 + 9*16^1 + 12*16^2 + 10*16^3 + 14*16^4 + 15*16^5)_{10} \\&= (11*1 + 9*16 + 12*256 + 10*4096 + 14*65536 + 15*1048576)_{10} \\&= (16690331)_{10}\end{aligned}$$

On peut utiliser l'algorithme de Horner

# Passage base $b$ base $b^k$

Exemple :  $(111110000101)_2 = (?)_{16}$

$$\begin{array}{ccc} (1111)_2 & (1000)_2 & (0101)_2 \\ (15)_{10} & (8)_{10} & (5)_{10} \\ (F)_{16} & (8)_{16} & (5)_{16} \\ (111110000101)_2 = (F85)_{16} \end{array}$$

Exemple :  $(7605)_8 = (?)_2$

$$\begin{array}{cccc} (7)_8 & (6)_8 & (0)_8 & (5)_8 \\ (111)_2 & (110)_2 & (000)_2 & (101)_2 \\ (7605)_8 = (111110000101)_2 \end{array}$$

## 2.2 Les nombres positifs à partie fractionnaire

La partie fractionnaire n'a pas toujours un nombre fini de chiffres  
Utiliser une **valeur approchée** avec  $p$  chiffres après la virgule  
On parle d'**erreur de chute**

$$\underline{x} = \sum_{i=-p}^{n-1} \underline{c_i} \underline{b^i} \quad \text{et} \quad x = (c_{n-1}c_{n-2}\dots c_1c_0, c_{-1}c_{-2}\dots c_{-p+1}c_{-p})_b$$

$$1/3 = (0,1)_3 = (0,333333\dots)_{10}$$

## Conversion base 10 base $b$ (partie fractionnaire)

### Algorithme

pour  $i := -1$  pas  $-1$  à  $-p$  faire

$x := (x - \text{INT}(x)) * b; c_i := \text{INT}(x);$

fait

*Exercice* : Convertir 109,125 en binaire

## Conversion base $b$ base 10

Calculer la somme  $\underline{x} = \sum_{i=-p}^{n-1} \underline{c_i} \underline{b^i}$  avec l'algorithme de Horner

$$(0,2345)_8 = \frac{1}{8} \left( 2 + \frac{1}{8} \left( 3 + \frac{1}{8} \left( 4 + \frac{1}{8} (5) \right) \right) \right)$$

## Conversion entre les bases $b_1$ et $b_2$

Passer par la base dix ; si  $b_1 = b^{k_1}$  et  $b_2 = b^{k_2}$  passer par la base  $b$

# Passage base 10 base b

Exemple :  $(100,125)_{10} = (?)_2$

- On convertit la partie entière :  $(100)_{10} = (1100100)_2$
- Pour la partie fractionnaire on applique l'algorithme :

$$0,125 * 2 = 0,25$$

$$0,25 * 2 = 0,5$$

$$0,5 * 2 = 1,0$$

$$0,0 * 2 = 0,0$$

$$0,0 * 2 = \dots$$

$$(100,125)_{10} = (1100100,0010\dots)_2$$

# Passage base b base 10

Algorithme : développer :

$$\underline{x} = \sum_{i=-p}^{-1} \underline{c_i b^i}$$

Exemple :  $(0,2345)_8 = (?)_{10}$

$$(0.2345)_8 = ( (0)_{10} * 8^0 + (2)_{10} * 8^{-1} + (3)_{10} * 8^{-2} + (4)_{10} * 8^{-3} + (5)_{10} * 8^{-4} )_{10}$$

ou utiliser l'algorithme de Horner

## 2.3 Les entiers relatifs en binaire

Format fixe sur  $n$  bits ( $a_{n-1} \dots a_0$ )

Nombre fini d'entiers relatifs  $x$  dans  $[m, M]$

Les entiers trop grands ou trop petits ont leur partie haute troquée  
c'est un **dépassement de capacité** de la représentation

**Addition en base 2**

		1			
	5	0	1	0	1
+	6	0	1	1	0
	---	-----			
	11	1	0	1	1



# Représentation en signe et valeur absolue

Le bit le plus significatif  $a_{n-1}$  représente le signe de l'entier :

$a_{n-1} = 0$  si le nombre est positif

$a_{n-1} = 1$  si le nombre est négatif

La valeur absolue est représentée par les  $n-1$  bits ( $a_{n-2} \dots a_0$ )

Le calcul d'une somme n'est pas immédiat

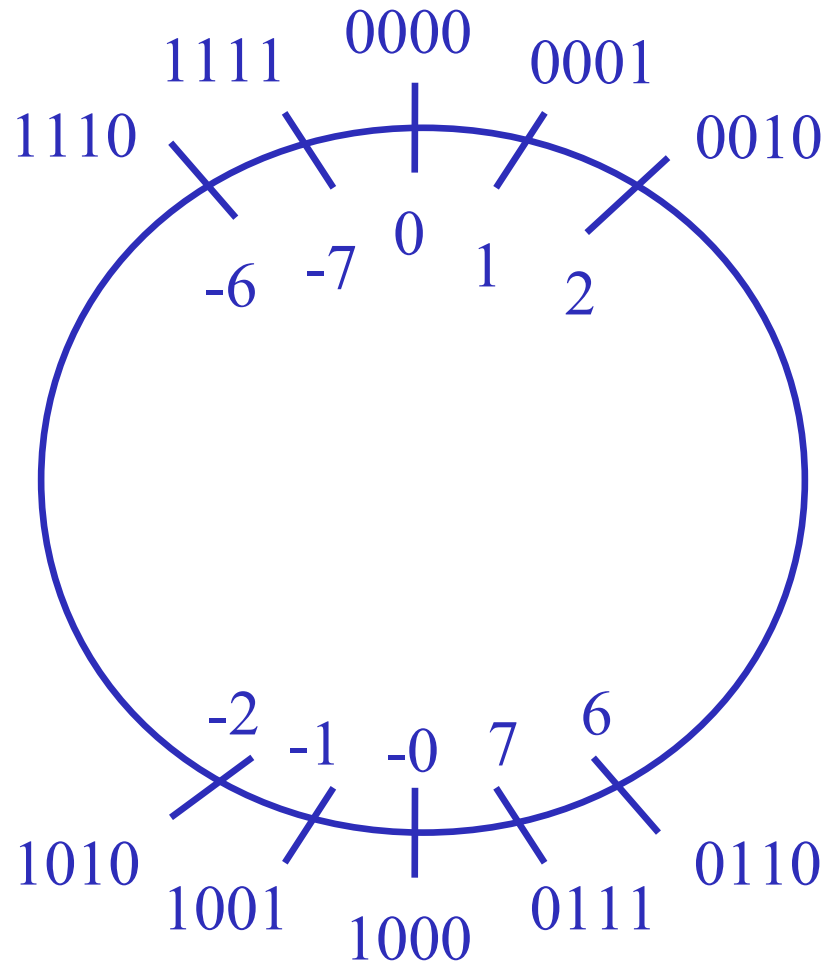
le signe dépend du signe et de la valeur absolue des nombres

la valeur absolue est obtenue par addition ou soustraction

Deux représentations pour zéro : -0 et +0

# Signe et valeur absolue

Ex. sur 4 bits



# Représentation en complément restreint

Entier positif représenté par sa valeur absolue en binaire  $a_{n-1} = 0$

Entier négatif représenté par le complément restreint de la représentation de sa valeur absolue

$$-x = \text{CR}(x)$$

Le complément restreint se calcule en complémentant chaque bit

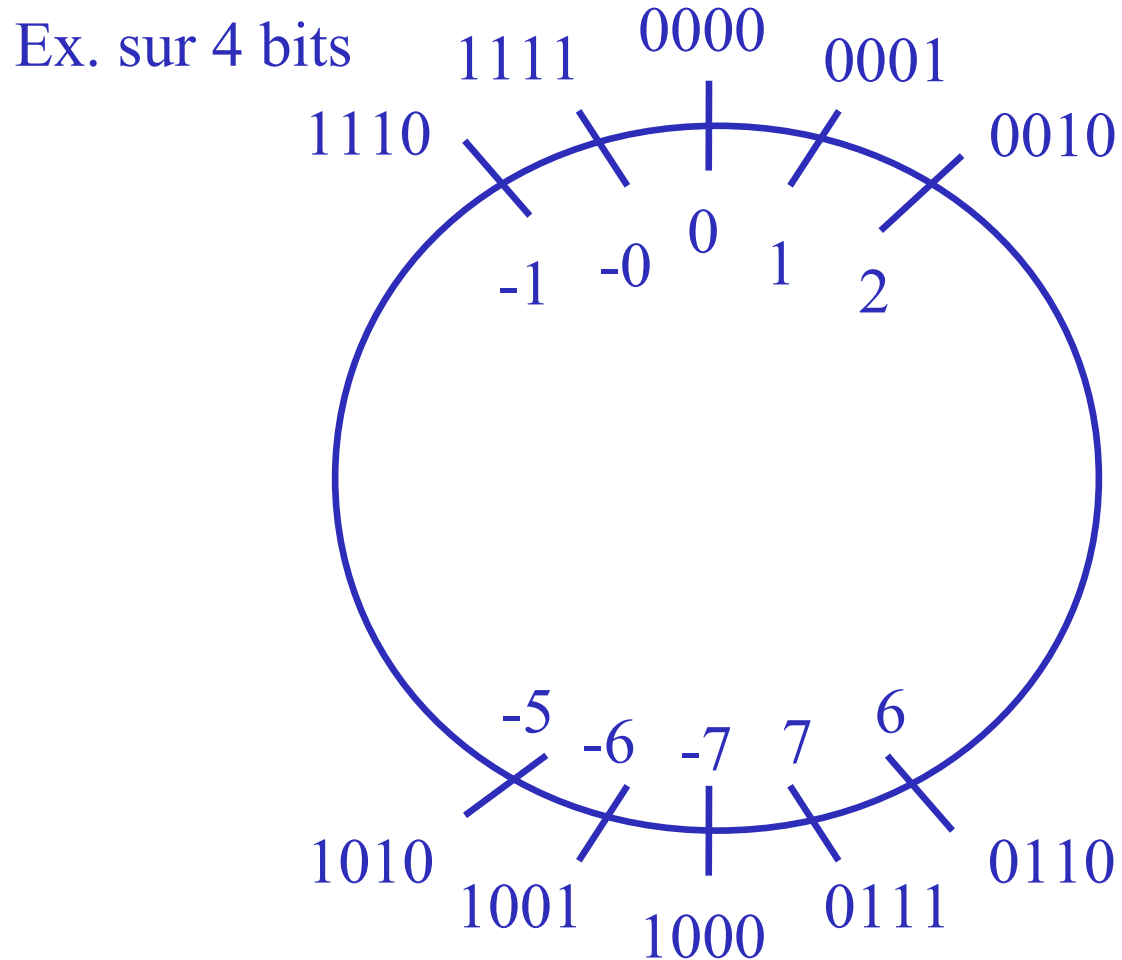
$$\begin{array}{ll} \underline{x} : & (x_{n-1} \dots x_0) \\ \underline{-x} : & (\overline{x_{n-1}} \dots \overline{x_0}) \\ 3 : & (0\ 0\ 1\ 1) \\ -3 : & (1\ 1\ 0\ 0) \end{array}$$

Propriété :  $x = \text{CR}(\text{CR}(x))$

Ajuster le résultat de l'addition suivant la retenue à cause de -0

**OVERFLOW** : nombres de même signe et somme de signe opposé

# Complément restreint



# Représentation en complément vrai

## complément à deux

Entier positif représenté par sa valeur absolue en binaire  $a_{n-1} = 0$

Entier négatif représenté par le complément vrai  
de la représentation de sa valeur absolue

$$-x = CV(x) = CR(x) + 1$$

Le complément vrai se calcule en ajoutant 1 au complément restreint  
le bit  $a_n$  est ignoré

$$\begin{array}{rcl} 3 : & (0011) & \\ & (1100) = CR(3) & \\ & +1 & \end{array}$$

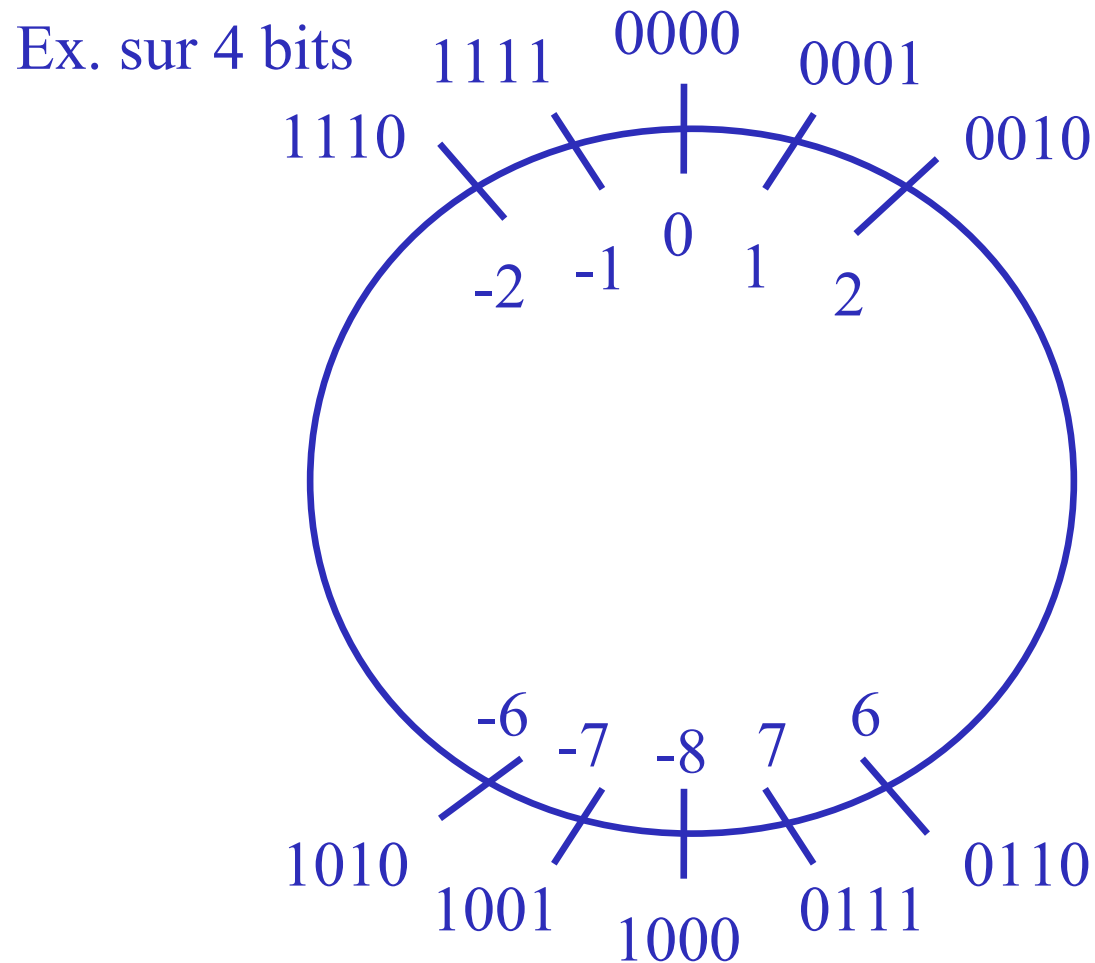
$$-3 : (1101) = CV(3)$$

$$\begin{array}{rcl} 0 & (0000) & \\ & (1111) = CR(0) & \\ & +1 & \end{array}$$

$$0 \quad 1(0000) = CV(0)$$

Propriété :  $x = CV(CV(x))$

# Complément vrai (à deux)



# Représentation en excès à $2^{n-1}$

Entier relatif  $x$  représenté par  $(x + 2^{n-1}) \bmod 2^n$

Zéro est représenté par  $2^{n-1}$  en binaire

Corriger le résultat de l'addition en retranchant  $2^{n-1}$   
par complémentation du bit de signe  $a_{n-1}$

## *Exercices :*

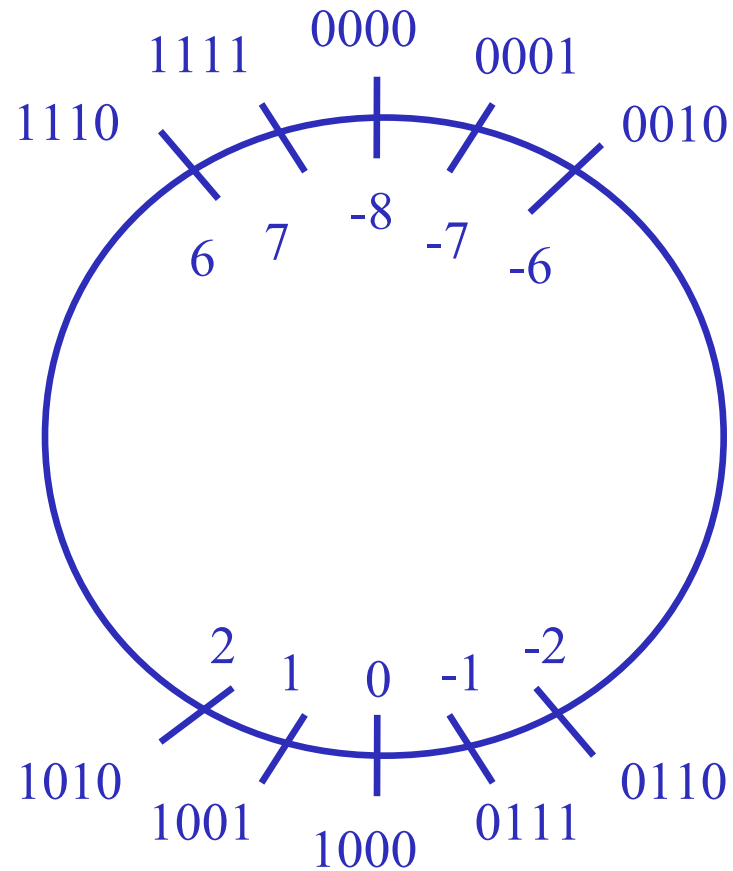
1- Pour le format fixe  $n=4$  calculer

$2+3$ ,  $3+(-2)$ ,  $(-5)+(-2)$ ,  $6+3$ ,  $(-4)+3$ ,  $(-5)+(-3)$ ,  $(-5)+5$ ,  $(-4)+(-5)$   
avec les quatre représentations.

2- Déterminer le plus grand nombre représentable et le plus petit  
pour chaque représentation

# Excès à $2^{n-1}$

Excès à 8 sur 4 bits





## 2.4 Les réels en virgule fixe

Le réel  $x$  est multiplié par un facteur constant  $2^p$  où  $p > 0$

$$\underline{x} = s \sum_{i=-p}^{n-1} \underline{c_i} 2^i \quad ; \quad x \cdot 2^p = s (c_{n-1}c_{n-2}\dots c_1c_0c_{-1}c_{-2}\dots c_{-p+1}c_{-p})_2 ; s = \text{sgn}(x)$$

On obtient un entier relatif sur  $n+p$  bits, la virgule n'apparaît plus  
Seule la connaissance du facteur  $2^p$  permet de retrouver sa position

**Placer correctement le 1** lors du calcul du complément vrai

On a les mêmes problèmes qu'avec les entiers relatifs :

erreur de chute, erreur de dépassement de capacité

Overflow : erreur de dépassement arithmétique

# Représentation en virgule fixe

**Représenter -109,125 sur 12 bits dont 4 pour la partie fractionnaire**

On représente en base 2 le réel 109,125

$$(109,125)_{10} \rightarrow (01101101,0010)_2$$

On le multiplie par le facteur constant de  $2^4$

$$(01101101,0010)_2 2^4 = (011011010010)_2$$

On utilise la représentation en complément vrai

$$\begin{aligned} \text{CR}(011011010010) + 1 &= 100100101101 + 1 \\ &= 100100101110 \end{aligned}$$

Donc -109,125  $\rightarrow$  100100101110

# Représentation en virgule flottante

Inconvénients des représentations en virgule fixe

- l'erreur due au débordement
- l'erreur de chute cause un défaut de précision

Les représentations en virgule flottante représentent les réels  $x$  avec le même nombre de chiffres significatifs, la mantisse **0,m** est **normalisée**

$$x = (-1)^s \times 0,m \times 2^e \text{ avec } 0,1 \leq 0,m < 1$$

# Virgule flottante REAL

En Turbo Pascal le format REAL utilise 48 bits (6 octets)

$$m = 0,1F$$

$$C = e + 128$$

1

39

8

S	F	C
---	---	---

si  $C = 0$  et  $F = 0$      $X = 0$

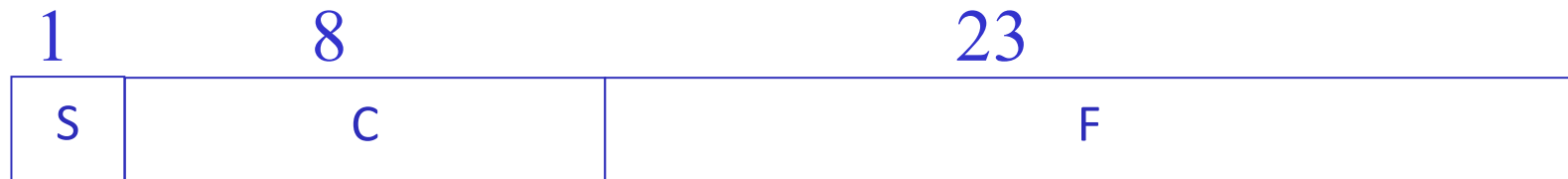
sinon     $X = (-1)^S \times 0,1F \times 2^{(C-128)}$

Exemple :  $X = 5$

# Virgule flottante SINGLE (IEEE)

Le format SINGLE utilise 32 bits (4 octets)

$$C = e + 128$$



$$0 < C < 255$$

$$X = (-1)^s \times 0.1F \times 2^{(C-126)}$$

$$C = 0 \text{ et } F \neq 0$$

$$X = (-1)^s \times 0.F \times 2^{-126}$$

$$C = 0 \text{ et } F = 0$$

$$X = (-1)^s 0$$

$$C = 255 \text{ et } F = 0$$

$$X = (-1)^s \infty$$

$$C = 255 \text{ et } F \neq 0$$

NaN (Not a Number)

# Virgule flottante DOUBLE (IEEE)

Le format DOUBLE utilise 64 bits (8 octets)

$$C = e + 1022$$



$$0 < C < 2047$$

$$X = (-1)^s \times 0.1F \times 2^{(C-1022)}$$

$$C = 0 \text{ et } F \neq 0$$

$$X = (-1)^s \times 0.F \times 2^{-1022}$$

$$C = 0 \text{ et } F = 0$$

$$X = (-1)^s 0$$

$$C = 2047 \text{ et } F = 0$$

$$X = (-1)^s \infty$$

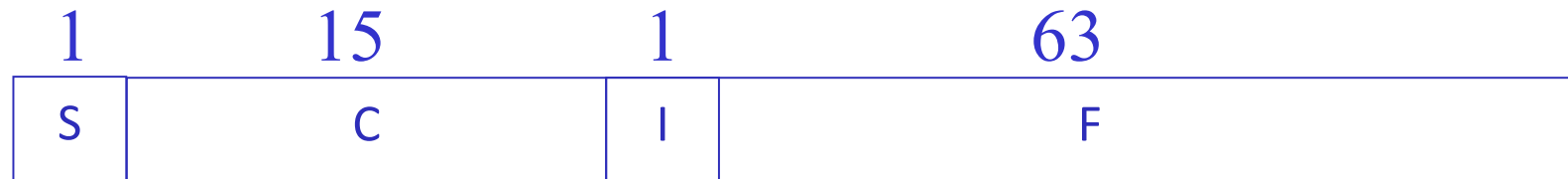
$$C = 2047 \text{ et } F \neq 0$$

$$\text{NaN}$$

# Virgule flottante EXTENDED (IEEE)

Le format EXTENDED utilise 80 bits (10 octets)

$$C = e + 16382 \quad I = 0 \text{ ou } 1 \quad 0, m = 0, IF$$



$$0 < C < 32767$$

$$X = (-1)^S \times 0.IF \times 2^{(C-16382)}$$

$$C = 32767 \text{ et } F = 0$$

$$X = (-1)^S \infty$$

$$C = 32767 \text{ et } F \neq 0$$

$$\text{NaN}$$