Architecture des ordinateurs

(X31I050)

Frédéric Goualard

Laboratoire d'Informatique de Nantes-Atlantique, UMR CNRS 6241
Bureau 112, bât. 11
Frederic.Goualard@univ-nantes.fr

Représentation de l'information

Ordinateur numérique *vs.* analogique (1/2)

Ordinateur numérique *vs.* analogique (2/2)

Ordinateur analogique :

- Transformation d'un problème en un autre *analogue* (exemple : addition de nombres $\rightarrow$ addition de résistances) résolvable directement et physiquement sur des *valeurs continues*

Ordinateur numérique :

- Transformation d'un problème en une suite d'instructions sans analogie sur des *valeurs discrètes* puis exécution du code

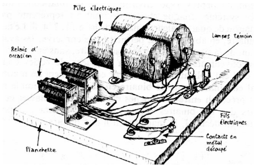
Ordi. numérique : représentation des informations

George Boole, Algèbre booléenne, 1847

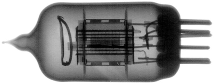
George Stibitz, model K 1937 (« *kitchen* ») :

Claude Shannon, 1937

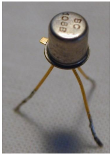
John V. Atanasoff, 1937–1942 (ABC)



Relais électromécanique (Henry, 1835)

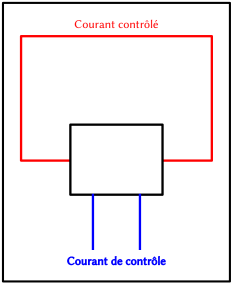


Tube thermionique
"*lampe à vide*" (De Forest, 1907)



Transistor
(Shockley, Bardeen & Brattain, 1947)

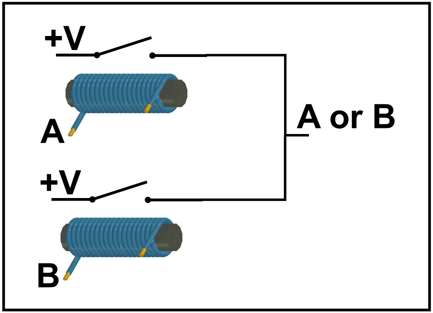
Circuit intégré (Kilby, 1958)

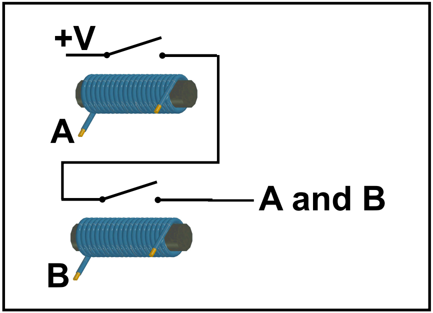


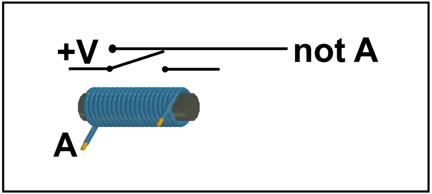
Ordinateur numérique ($\neq$ analogique) : architecture utilisant l'absence ou la présence de courant (relais, tubes ou transistors)

➡ 2 états (0 et 1) : représentation *binaire*

Interrupteurs et logique binaire







Les nombres binaires

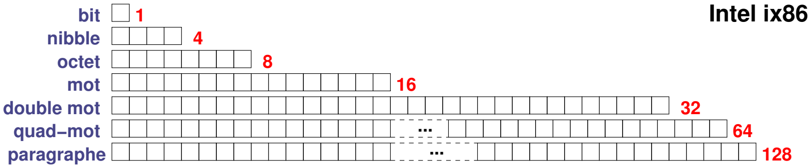
- Deux états internes symbolisés par 0 et 1
- Informations :

Nom	Valeur
Bit	0 ou 1
Octet	00000000 à 11111111

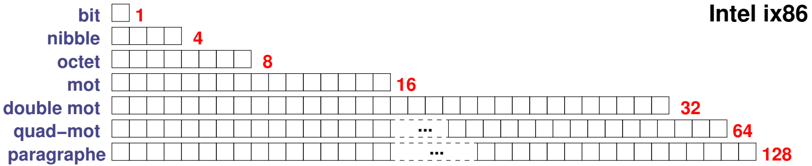
Multiples définis depuis 1998 :

Nom	Notation	Valeur
1 kibibit	1 Kibit	$2^{10} = 1\,024$ bits
1 kilobit	1 kbit	$10^3 = 1\,000$ bits
1 mebioctet	1 MiB/1 Mio	$2^{20} = 1\,048\,576$ octets
1 megaoctet	1 MB/1 Mo	$10^6 = 1000\,000$ octets
1 gibioctet	1 GiB/1 Gio	$2^{30} = 1\,073\,741\,824$ octets
1 gigaoctet	1 GB/1 Go	$10^9 = 1\,000\,000\,000$ octets

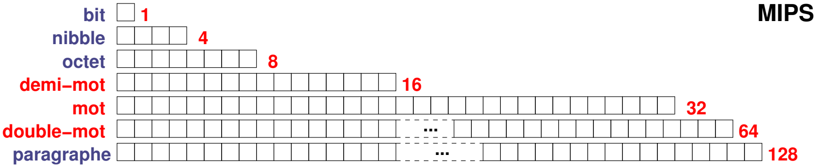
Types de données fondamentaux



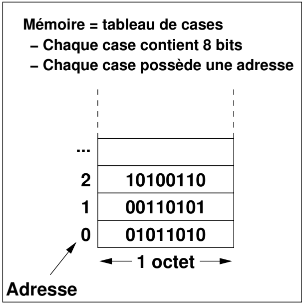
Types de données fondamentaux



ou



La mémoire

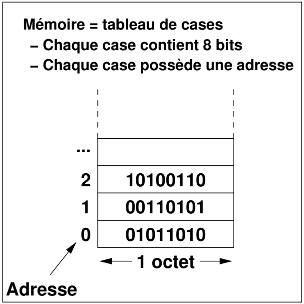


On veut stocker :

- des entiers positifs (12, 534256, ...)
- des entiers négatifs (-56, -435345, ...)
- des caractères ('a', 'Z', '5', '+', ...)
- des chaînes de caractères ("bonjour", ...)
- des réels (12.34, -670.5552, ...)
- des instructions

Mais :
Une case contient uniquement des bits

La mémoire



On veut stocker :

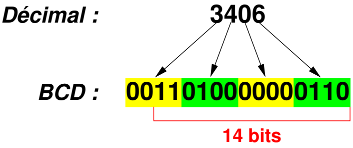
- des entiers positifs (12, 534256, ...)
- des entiers négatifs (-56, -435345, ...)
- des caractères ('a', 'Z', '5', '+', ...)
- des chaînes de caractères ("bonjour", ...)
- des réels (12.34, -670.5552, ...)
- des instructions

Mais :
Une case contient uniquement des bits

➟ tout coder sous forme d'*entiers positifs en binaire*

Représentation « décimale codée binaire »

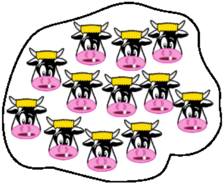
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
xxxx	illégal



- Codage BCD gourmand
  (12 bits seulement en binaire pour coder 3406)
- Opérations arithmétiques compliquées et pas efficaces
- Entrées/sorties facilitées

Représentation usuelle : 12

$$= 1 \times 10^1 + 2 \times 10^0$$



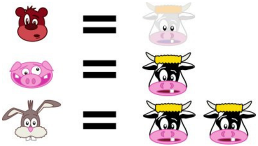
➟ Représentation *positionnelle* :

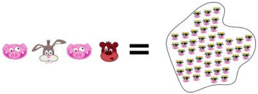
- Choix d'une base b : (ex. : 10, 2, ...)
- Choix de b symboles

Exemples :

Base 2 $(0, 1)$: 1100_2 $(= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 12_{10})$

Base 3 « *Toons* » :





$$1 \times 3^3 + 2 \times 3^2 + 1 \times 3^1 + 0 \times 3^0 = 48$$

Expression d'un nombre *a* en base *b* :

$$\begin{aligned} a_b &= (a_n a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots a_{-m})_b \\ &= a_n b^n + \dots + a_2 b^2 + a_1 b + a_0 + a_{-1} b^{-1} + \dots a_{-m} b^{-m} \end{aligned}$$

a_n : chiffre *le plus significatif*

a_{-m} : chiffre *le moins significatif*

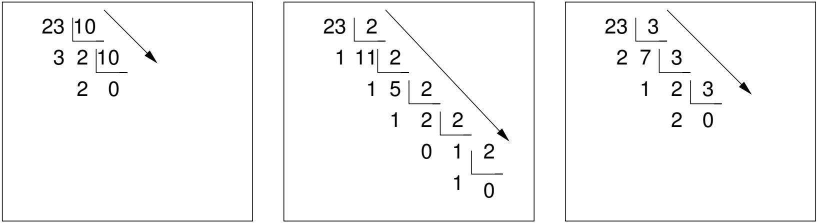
Exemples :

$$\begin{array}{lll} 98_{10} & = 9 * 10^1 + 8 * 10^0 & \\ 101_2 & = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 & = 5_{10} \\ 136_8 & = 1 * 8^2 + 3 * 8^1 + 6 * 8^0 & = 94_{10} \\ 3A_{16} & = 72_8 & = 58_{10} \\ 110.01_2 & = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} & = 6.25_{10} \end{array}$$

Avec $\mathbf{A}_{16} = 10_{10}$, $\mathbf{B}_{16} = 11_{10}$, ...

Passage de 23_{10} en base 10, 2, 3 ?

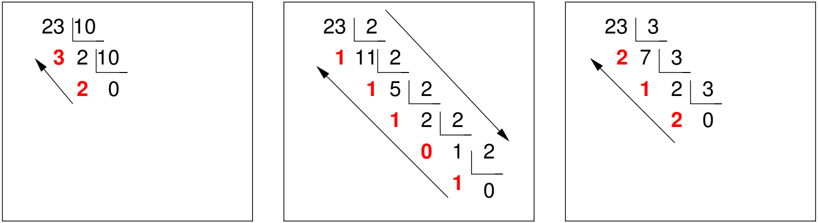
Division par 10, 2, 3 itérées.



Changement de base : exemple

Passage de 23_{10} en base 10, 2, 3 ?

Division par 10, 2, 3 itérées.



Résultat : $(23) = 23_{10} = 10111_2 = 212_3$

Passage de la représentation de u en base r à sa représentation en base R ?

$$\begin{aligned} u &= (x_{k-1}x_{k-2}\dots x_0)_r \\ &= (X_{K-1}X_{K-2}\dots X_0)_R \end{aligned}$$

Représentation de Horner :

$$\begin{aligned} u &= (X_{K-1}R^{K-1} + X_{K-2}R^{K-2} + \cdots + X_0)_r \\ &= (R(\dots R(R \times 0 + X_{K-1}) + X_{K-2}) + X_{K-3})\dots) + X_0)_r \end{aligned}$$

en exprimant toutes les constantes et en faisant toutes les opérations dans la base r.

$\Rightarrow$ Algorithme : divisions itérées par R en base r

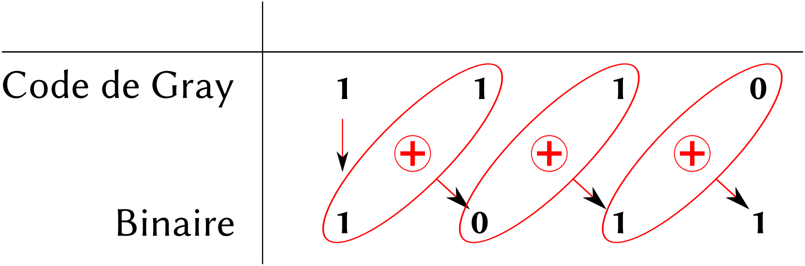
Code de Gray

- Choix d'un ordre pour entiers binaires tel que chaque élément diffère par un seul bit du suivant et du précédent :

Code de Gray BRGC	Ordre naturel	Nombre
000	000	0
001	001	1
011	010	2
010	011	3
110	100	4
111	101	5
101	110	6
100	111	7

- Ordre non unique (*Binary Reflected Gray Code*)
- Utilisation : roue codeuse, tableau de Karnaugh, code de correction d'erreur, ...

Conversion Gray/binaire



$\oplus$	0	1
0	0	1
1	1	0

Conversion binaire/Gray

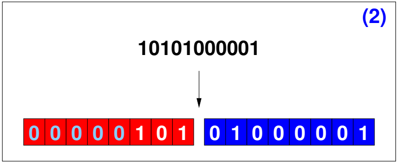
Binaire	**1**	$\oplus$	**0**	$\oplus$	**1**	$\oplus$	**1**
Code de Gray	**1**		**1**		**1**		**0**

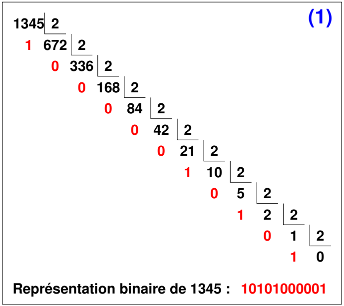
$\oplus$	0	1
0	0	1
1	1	0

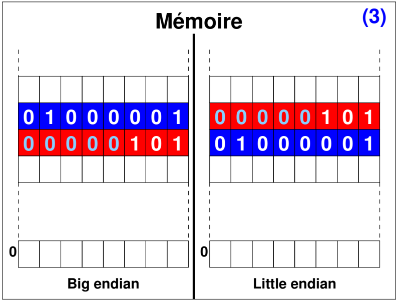
Codage des entiers positifs (non-signés)

Stockage d'un entier positif en mémoire :

1. Représentation du nombre en binaire
2. Découpage de la représentation binaire en octets
3. Stockage de chaque octet consécutivement



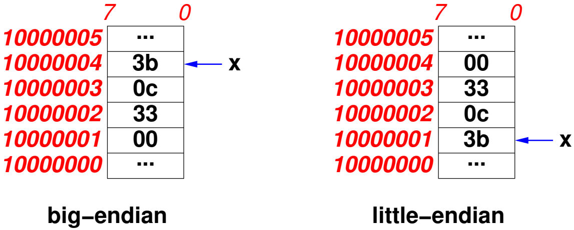




Big endian vs. little endian

- Architecture ix86 : adressage par octet *little-endian*
- Stockage d'infos sur plus d'un octet :
  - msb (*Most Significant Byte*) à l'adresse la plus petite
    - ➡ big-endian
  - lsb (*Least Significant Byte*) à l'adresse la plus petite
    - ➡ little-endian

unsigned long int x = 3345467;
{ => x = 0x00330c3b; }



Les entiers positifs (non-signés)

Entiers représentables sur 1 octet :

Base 2	Base 10	Base 16
11111111	255	FF
11111110	254	FE
⋮	⋮	
00010001	17	11
00010000	16	10
00001111	15	0F
00001110	14	0E
00001101	13	0D
00001100	12	0C
00001011	11	0B
00001010	10	0A
00001001	9	09
00001000	8	08
00000111	7	07
00000110	6	06
00000101	5	05
00000100	4	04
00000011	3	03
00000010	2	02
00000001	1	01
00000000	0	00

Les entiers positifs (non-signés)

Entiers représentables sur 1 octet :

Base 2	Base 10	Base 16
11111111	255	FF
11111110	254	FE
⋮	⋮	
00010001	17	11
00010000	16	10
00001111	15	0F
00001110	14	0E
00001101	13	0D
00001100	12	0C
00001011	11	0B
00001010	10	0A
00001001	9	09
00001000	8	08
00000111	7	07
00000110	6	06
00000101	5	05
00000100	4	04
00000011	3	03
00000010	2	02
00000001	1	01
00000000	0	00

Passage de base 2 à base 16 et inversement :

Base 2 : 1000101001

0 0 1 0 | 0 0 1 0 | 1 0 0 1

Base 16 : 2 2 9

B2	B16
1111	F
1110	E
1101	D
1100	C
1011	B
1010	A
1001	9
1000	8
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

Les entiers signés

Entiers non-signés : ensemble d'entiers positifs

Entiers signés : ensemble d'entiers positifs et négatifs

Comment représenter des entiers négatifs ?

Les entiers signés

Entiers non-signés : ensemble d'entiers positifs

Entiers signés : ensemble d'entiers positifs et négatifs

Comment représenter des entiers négatifs ?

Convention de recodage des chaînes de bits

- Magnitude signée
- Complément à 1
- Complément à 2
- Biaisée

Représentation par magnitude signée

Dans un entier de k bits, le bit de poids fort code le signe :

- $0 =$ positif
- $1 =$ négatif

Exemples (sur 8 bits) :

- $+25_{10} = 00011001_2$
- $-25_{10} = 10011001_2$

Inconvénient = deux représentations pour 0 :

- $+0_{10}$=00000000_2
- -0_{10}=10000000_2

Sur 8 bits : -127..+127

Représentation par magnitude signée

Dans un entier de k bits, le bit de poids fort code le signe :

- $0 =$ positif
- $1 =$ négatif

Exemples (sur 8 bits) :

- $+25_{10} = 00011001_2$
- $-25_{10} = 10011001_2$

Inconvénient = deux représenta

- $+0_{10}=00000000_2$
- $-0_{10}=10000000_2$

Sur 8 bits : -127..+127

chaîne de bits	non signé	magnitude signée
1111	15	–7
1110	14	–6
1101	13	–5
1100	12	–4
1011	11	–3
1010	10	–2
1001	9	–1
1000	8	–0
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Complément à 1

Le bit de poids fort correspond au signe :

- 0 = positif
- 1 = négatif

Un nombre négatif s'obtient en complémentant bit à bit sa valeur absolue avec 1 (cf. complément à 9 de la *Pascaline*)

Exemple : représentation de -25_{10} sur 8 bits :

- $25_{10} = 00011001_2$
- D'où : $\begin{array}{cr} & 11111111 \\ - & 00011001 \\ \hline = & 11100110 \end{array}$

Deux représentations pour 0 : 00000000_2 et 11111111_2

Nombres représentés sur 8 bits : -127..+127

Complément à 1

Le bit de poids fort correspond au signe :

- 0 = positif
- 1 = négatif

Un nombre négatif s'obtient en compl
absolue avec 1 (cf. complément à 9 d
Exemple : représentation de -25_{10} su

- $25_{10} = 00011001_2$
- D'où : $\begin{array}{rr} & 11111111 \\ - & 00011001 \\ \hline = & 11100110 \end{array}$

Deux représentations pour 0 : 000000
Nombres représentés sur 8 bits : -127.

chaîne de bits	non signé	complément à 1
1111	15	–0
1110	14	–1
1101	13	–2
1100	12	–3
1011	11	–4
1010	10	–5
1001	9	–6
1000	8	–7
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Complément à 2

Le bit de poids fort indique le signe :

- 0 = positif
- 1 = négatif

Un nombre négatif s'obtient en ajoutant 1 au complément à 1 de sa valeur absolue (et inversement).

Exemple : représentation de -25_{10} ?

- $+25_{10}$=00011001_2
- Complément à 1 de $+25_{10}$ = 11100110_2
- Ajout de 1 : 11100110_2+1 = 11100111_2

Une seule représentation pour 0 : 00000000_2

Nombres représentés sur 8 bits : -128..+127

Complément à 2

Le bit de poids fort indique le s

- 0 = positif
- 1 = négatif

Un nombre négatif s'obtient en ... de sa valeur absolue (et inversement)

Exemple : représentation de -2

- $+25_{10}$=00011001_2
- Complément à 1 de $+25_{10}$
- Ajout de 1 : 11100110_2+1

Une seule représentation pour 0

Nombres représentés sur 8 bits

chaîne de bits	non signé	complément à 2
1111	15	–1
1110	14	–2
1101	13	–3
1100	12	–4
1011	11	–5
1010	10	–6
1001	9	–7
1000	8	–8
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Représentation biaisée

Représentation des nombres négatifs par ajout d'un biais les rendant positifs.
Le biais est ajouté aussi aux nombres positifs

Exemple : codage sur 8 bits avec un biais de 127

- -12_{10} est codé par $-12 + 127 = 115$ i.e. 01110011_2
- 30_{10} est codé par $30 + 127 = 157$ i.e. 10011101_2

Nombres représentés sur 8 bits avec biais de 127 : -127..128

Représentation biaisée

Représentation des nombres nég
rendant positifs.
Le biais est ajouté aussi aux no

Exemple : codage sur 8 bits ave

- -12_{10} est codé par $-12 +$
- 30_{10} est codé par $30 + 127$

Nombres représentés sur 8 bits

chaîne de bits	non signé	codage avec biais de 7
1111	15	8
1110	14	7
1101	13	6
1100	12	5
1011	11	4
1010	10	3
1001	9	2
1000	8	1
0111	7	0
0110	6	–1
0101	5	–2
0100	4	–3
0011	3	–4
0010	2	–5
0001	1	–6
0000	0	–7

Les nombres négatifs : résumé

binaire	décimal	signe + magnitude	complément à 1	complément à 2	représentation biaisée
0000	0	0	0	0	–7
0001	1	1	1	1	–6
0010	2	2	2	2	–5
0011	3	3	3	3	–4
0100	4	4	4	4	–3
0101	5	5	5	5	–2
0110	6	6	6	6	–1
0111	7	7	7	7	0
1000	8	–0	–7	–8	1
1001	9	–1	–6	–7	2
1010	10	–2	–5	–6	3
1011	11	–3	–4	–5	4
1100	12	–4	–3	–4	5
1101	13	–5	–2	–3	6
1110	14	–6	–1	–2	7
1111	15	–7	–0	–1	8

(biais = 7)

Les nombres négatifs : résumé

binaire	décimal	signe + magnitude	complément à 1	complément à 2	représentation biaisée
0000	0	0	0	0	–7
0001	1	1	1	1	–6
0010	2	2	2	2	–5
0011	3	3	3	3	–4
0100	4	4	4	4	–3
0101	5	5	5	5	–2
0110	6	6	6	6	–1
0111	7	7	7	7	0
1000	8	–0	–7	–8	1
1001	9	–1	–6	–7	2
1010	10	–2	–5	–6	3
1011	11	–3	–4	–5	4
1100	12	–4	–3	–4	5
1101	13	–5	–2	–3	6
1110	14	–6	–1	–2	7
1111	15	–7	–0	–1	8

(biais = 7)

Quelle est la valeur de la chaîne de bits : 1010 ?

Représentation des caractères et chaînes

Plusieurs formats pour représenter des caractères (imprimables et de contrôle) sous forme binaire :

- EBCDIC (*Extended Binary-Coded Decimal Interchange Code*)
  - Représentation sur 8 bits (256 caractères possibles)
  - Utilisé autrefois sur les mainframes IBM
- ASCII (*American Standard Code for Information Interchange*)
  - Représentation sur 7 bits (pas d'accents)
  - ASCII étendu : sur 8 bits mais pas de normalisation
- Unicode : encodage sur 8–48 bits (UTF-8) pour représenter tous les caractères de toutes les langues

Chaîne de caractères : suite de caractères stockés consécutivement en mémoire (terminateurs ?)

Exemple : table ASCII restreinte

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Source : http://upload.wikimedia.org/wikipedia/commons/1/1b/ASCII-Table-wide.svg

Opérations $+$, $-$, $\times$, $\div$ sur :

- Nombres non-signés
- Nombres signés *en complément à 2*

Le calcul se fait indépendamment de l'interprétation des chaînes de bits

Addition binaire entière

L'addition se fait classiquement avec les règles :

$$
\begin{aligned}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 0 \quad \text{avec retenue de } 1
\end{aligned}
$$

Exemples :

$$
\begin{array}{rcccccccccc}
 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & & \mathbf{26} \\
+ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & & \mathbf{12} \\
 & & & \color{red}{1} & \color{red}{1} & & & & & & \\
\hline
 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & & \mathbf{38}
\end{array}
\qquad
\begin{array}{ccccccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & & \mathbf{19} \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & & \mathbf{62} \\
 & \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{1} & & & & \\
\hline
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & & \mathbf{81}
\end{array}
$$

Résultat sur 9 bits :

- Non signé : dépassement de capacité
- Signé : pas de signification

Soustraction binaire entière

La soustraction suit les règles suivantes :

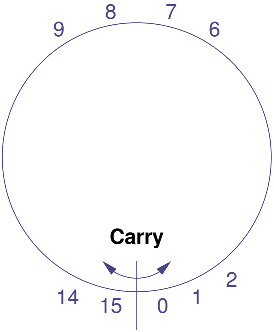
$$
\begin{array}{ll}
0 - 0 = 0 & \\
0 - 1 = 1 & \text{et on prend 1 à gauche} \\
1 - 0 = 1 & \\
1 - 1 = 0 &
\end{array}
$$

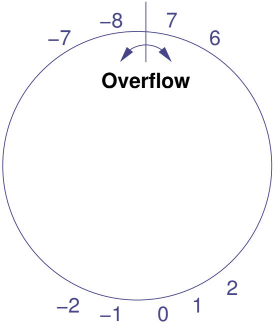
Exemples :

$$
\begin{array}{rccccccccl}
 & 0 & 0 & 1 & {}^{1}0 & 0 & 1 & 0 & 1 & \mathbf{37} \\
- & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{17} \\
- & & & 1 & & & & & & \\
\hline
 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & \mathbf{20}
\end{array}
\qquad
\begin{array}{rccccccccl}
 & 0 & 0 & 1 & {}^{1}1 & {}^{1}0 & {}^{1}0 & 1 & 1 & \mathbf{51} \\
- & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \mathbf{22} \\
- & & & 1 & 1 & 1 & & & & \\
\hline
 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & \mathbf{29}
\end{array}
$$

On peut aussi faire une addition avec le complément à 2 du deuxième opérande.

Carry vs. Overflow





Non signé		Signé		Non signé		Signé	
1110	14	1110	–2	0111	7	0111	7
+ 1011	11	+ 1011	–5	+ 0001	1	+ 0001	1
11001	25>15	$\cancel{1}$1001	–7	1000	8	1000	–8

Détection :
1 bit de plus en sortie

Détection :
Opérandes de même signe
Résultat de signe différent

Multiplication binaire entière

La multiplication suit les règles suivantes :

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Exemple :

```
      0 0 1 0 1 0 0 1      41
    x 0 0 0 0 0 1 1 0       6
      ---------------
      0 0 0 0 0 0 0 0
    0 0 1 0 1 0 0 1
  0 0 1 0 1 0 0 1
  -------------------
  0 0 1 1 1 1 0 1 1 0     246
```

On peut aussi faire des additions itérées

Division binaire entière

Division obtenue par itération de soustractions jusqu'à ce que le résultat de la soustraction soit inférieur au diviseur :

- Quotient = nombre de soustractions
- Reste = résultat de la dernière soustraction

Exemple : division de 7 par 3

$$\underbrace{\underbrace{\begin{array}{rrr} & 0\,0\,0\,0\,0\,1\,1\,1 & \mathbf{7} \\ - & 0\,0\,0\,0\,0\,0\,1\,1 & \mathbf{3} \\ \hline & 0\,0\,0\,0\,0\,1\,0\,0 & \mathbf{4} \end{array}}_{1} \quad \underbrace{\begin{array}{rrr} & 0\,0\,0\,0\,0\,1^{1}0^{1}0 & \mathbf{4} \\ - & 0\,0\,0\,0\,0\,0\,1\,1 & \mathbf{3} \\ - & 1\,1 & \\ \hline & \mathbf{0\,0\,0\,0\,0\,0\,0\,1} & \mathbf{1} \end{array}}_{2}}_{\mathbf{2}}$$

Résultat : quotient = 2 et reste = 1

On peut aussi faire comme une division classique en décimal

Calcul sur les réels (1)

- Infinité de nombres entiers
  - ➡ *Mais* représentation correcte dans un intervalle
- Infinité de nombres réels
  - Impossibilité de représentation correcte même d'un petit intervalle :

$$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}\ \exists \mathbf{c} \in \mathbb{R} \quad \text{t.q.} \quad \mathbf{a} \leq \mathbf{c} \leq \mathbf{b}$$

$\Rightarrow$ Représentation d'un sous-ensemble de $\mathbb{Q}$

Calcul sur les réels (2)

- Nombres en virgule fixe :

$$110.011 = 1 \times 2^{2} + 1 \times 2^{1} + 0 \times 2^{0} + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 6.375$$

- Nombres en virgule flottante (notation scientifique) :

$$101010.10 = 1.0101010 \times 2^{5}$$
$$0.00100011 = 1.00011 \times 2^{-3}$$

➟ Usage de l'arithmétique en virgule flottante majoritaire

Passage d'un nombre réel de base 10 vers base 2 en virgule fixe :

- Partie entière : comme pour les entiers
- Partie décimale : multiplications itérées par 2

Exemple : conversion de 14.375_{10} en base 2 ?

- $14_{10} = 1110_2$ (divisions itérées par 2)
- $0.375_{10} = ???_2$

$$\begin{aligned}
0.375 \quad &\times 2 \quad &= \mathbf{0} + 0.75 \\
0.75 \quad &\times 2 \quad &= \mathbf{1} + 0.5 \\
0.5 \quad &\times 2 \quad &= \mathbf{1} + 0.0
\end{aligned}$$

Résultat : $14.375_{10} = 1110.\mathbf{011}_2$

Vers la norme IEEE 754

1936 : machines *Z* de K. Zuse

60's–70's : Design des FPUs = anarchie totale

- Portabilité nulle
- Peu de propriétés ($a = b \not\Leftrightarrow a - b = 0$)

1976 : création du i8087 par Intel (*best arithmetic*)

- Rapport Kahan–Coonen–Stone

1985 : Rapport K-C-S devient la norme *IEEE 754*

Aujourd'hui : norme implantée de fait sur tous les ordinateurs (sauf certains Cray).

Représentation des flottants

Nombre flottant x en binaire :

- un bit de signe s
- un exposant E
- un *signifiant* m

$$\mathbf{x} = (-\mathbf{1})^{\mathbf{s}} \times \mathbf{m} \cdot \mathbf{2}^{\mathbf{E}}$$

Représentations équivalentes :

(a) $\mathbf{0.000011}1010 \cdot 2^0$
(b) $\mathbf{0.000000}111010 \cdot 2^2$
(c) $\mathbf{1.110100} \cdot 2^{-5}$

Taille de signifiant fixée $\Rightarrow$ forme **c** plus précise

Représentation IEEE 754 (1)

- Représentation *normalisée* (forme **c**) ;
- Toujours un 1 avant la virgule $\Rightarrow$ pas codé (*hidden bit*)

$$\mathbf{m} = \mathbf{1.00101} \quad \longrightarrow \quad \mathbf{f} = \mathbf{00101}$$

- exposants négatifs et positifs : codage par biais :

$$\mathbf{E} \quad \longrightarrow \quad \mathbf{e} = \mathbf{E} + \text{biais}$$

Intérêt : comparaison lexicographique

$$\left[\begin{array}{l} \textbf{single (1,8,23)} \\ \textbf{double (1,11,52)} \\ \textbf{ix87 reg. (1,15,64)} \end{array}\right]$$

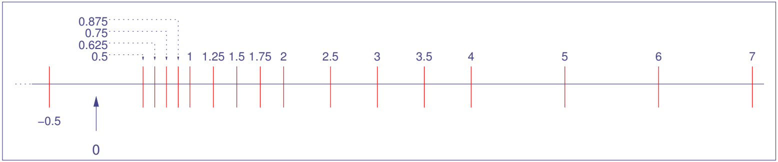
s	e	f
1	10101010101	101010101010...10100101
Signe	*Exposant biaisé*	*Partie fractionnaire*

Représentation IEEE 754 (2)

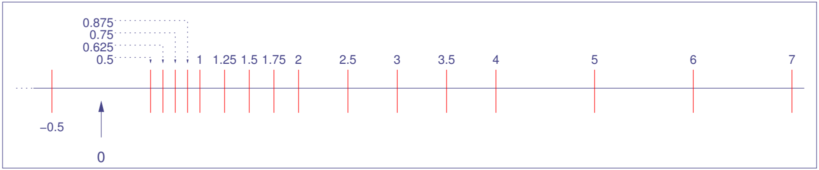
Exemple : format `tiny` sur 5 bits $(1, 2, 2)$ de biais 1 :
Nombres positifs représentables :

$$\mathbf{0\ 00\ 00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{-1}} = \mathbf{0.5}$$
$$\mathbf{0\ 00\ 01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{-1}} = \mathbf{0.625}$$
$$\mathbf{0\ 00\ 10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{-1}} = \mathbf{0.75}$$
$$\mathbf{0\ 00\ 11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{-1}} = \mathbf{0.875}$$
$$\mathbf{0\ 01\ 00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{0}} = \mathbf{1}$$
$$\mathbf{0\ 01\ 01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{0}} = \mathbf{1.25}$$
$$\mathbf{0\ 01\ 10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{0}} = \mathbf{1.5}$$
$$\mathbf{0\ 01\ 11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{0}} = \mathbf{1.75}$$

$$\mathbf{0\ 10\ 00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{1}} = \mathbf{2}$$
$$\mathbf{0\ 10\ 01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{1}} = \mathbf{2.5}$$
$$\mathbf{0\ 10\ 10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{1}} = \mathbf{3}$$
$$\mathbf{0\ 10\ 11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{1}} = \mathbf{3.5}$$
$$\mathbf{0\ 11\ 00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{2}} = \mathbf{4}$$
$$\mathbf{0\ 11\ 01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{2}} = \mathbf{5}$$
$$\mathbf{0\ 11\ 10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{2}} = \mathbf{6}$$
$$\mathbf{0\ 11\ 11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{2}} = \mathbf{7}$$

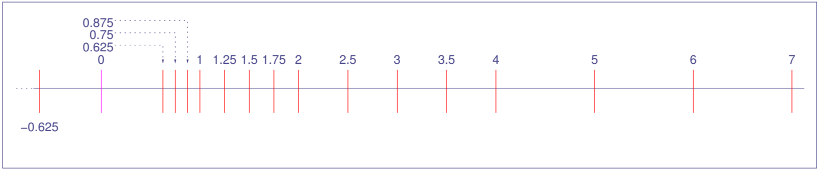


Représentation IEEE 754 (3)



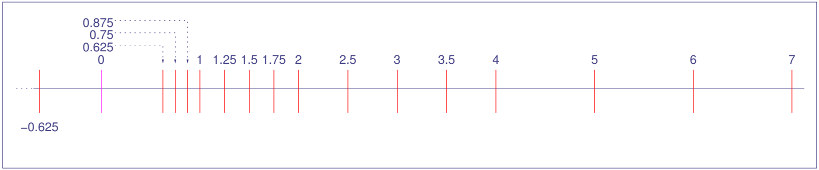
- Pas de codage pour 0

Représentation IEEE 754 (3)



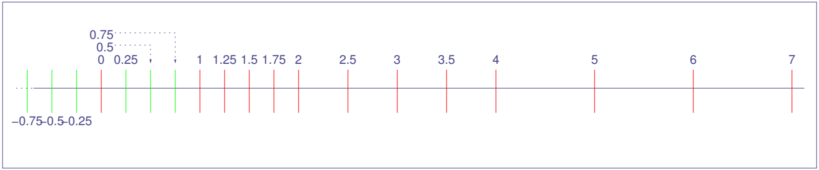
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)

Représentation IEEE 754 (3)



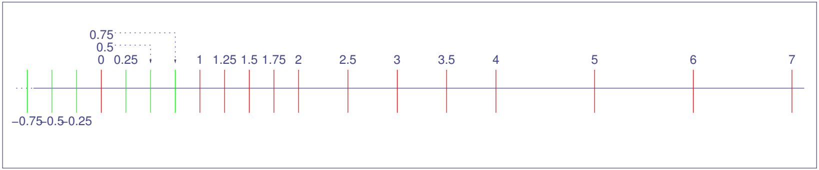
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0

Représentation IEEE 754 (3)



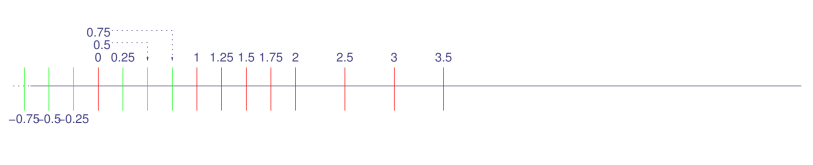
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - ➟ Plus de *hidden bit* à 1

Représentation IEEE 754 (3)



- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - ➟ Plus de *hidden bit* à 1
- Notions d'infinis mathématiques et de résultat indéfini :

Représentation IEEE 754 (3)



- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - ➟ Plus de *hidden bit* à 1
- Notions d'infinis mathématiques et de résultat indéfini :
  - Réserver $e = 3$

Représentation IEEE 754 (4)

Interprétation des bits :

$$\begin{cases} e = 3, & f \neq 0 & : v = \text{NaN} \\ e = 3, & f = 0 & : v = (-1)^s \times \infty \\ 0 < e < 3 & & : v = (-1)^s \times (1.f) \cdot 2^{e-1} \\ e = 0, & f \neq 0 & : v = (-1)^s \times (0.f) \cdot 2^0 \\ e = 0, & f = 0 & : v = (-1)^s \times 0 \end{cases}$$

00000	0.5
00001	0.625
00010	0.75
00011	0.875
00100	1
00101	1.25
00110	1.5
00111	1.75
01000	2
01001	2.5
01010	3
01011	3.5
01100	4
01101	5
01110	6
01111	7

$\longrightarrow$

00000	0
00001	0.25
00010	0.5
00011	0.75
00100	1
00101	1.25
00110	1.5
00111	1.75
01000	2
01001	2.5
01010	3
01011	3.5
01100	+∞
01101	NaN
01110	NaN
01111	NaN

NaNs et infinis

- *Not a Number* : cas indéfinis (*non ordonnés*)
- Infinis : $\begin{cases} 1/+\infty & = +0 \\ 1/-\infty & = -0 \\ 1/-0 & = -\infty \\ 1/0 & = +\infty \end{cases}$

$x+y$	NaN	$-\infty$	0	$+\infty$
NaN	✓	✓	✓	✓
$-\infty$	✓			✓
0	✓			
$+\infty$	✓	✓		

$x-y$	NaN	$-\infty$	0	$+\infty$
NaN	✓	✓	✓	✓
$-\infty$	✓	✓		
0	✓			
$+\infty$	✓			✓

$\sqrt{x}$	
NaN	✓
$-\infty$	
$x<0$	✓
0	
$+\infty$	

$x \times y$	NaN	$-\infty$	0	$+\infty$
NaN	✓	✓	✓	✓
$-\infty$	✓		✓	
0	✓	✓		✓
$+\infty$	✓		✓	

x/y	NaN	$-\infty$	0	$+\infty$
NaN	✓	✓	✓	✓
$-\infty$	✓	✓		✓
0	✓		✓	
$+\infty$	✓	✓		✓

NaNs (2)

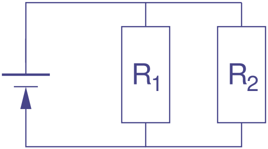
Cas spéciaux pour les opérateurs non définis par la norme IEEE 754 :

x	$\exp(x)$	$\log(x)$	$\sin(x)$	$\cos(x)$	$\tan(x)$	$\text{acos}(x)$	$\text{asin}(x)$	$\text{atan}(x)$
NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
$-\infty$	0.0	*NaN*	*NaN*	*NaN*	*NaN*			
$x < -745.13$	0.0							
$x < -1$						*NaN*	*NaN*	
$x < 0$		*NaN*						
0		$-\infty$						
$x > 1$						*NaN*	*NaN*	
$x > 709.78$	$+\infty$							
$+\infty$	$+\infty$	$+\infty$	*NaN*	*NaN*	*NaN*			

Source : librairie `fdlibm` ***sur les*** `doubles`

Usage des infinis

Calculer la résistance totale du circuit :



Formule :

$$T = \frac{1}{1/R_1 + 1/R_2}$$

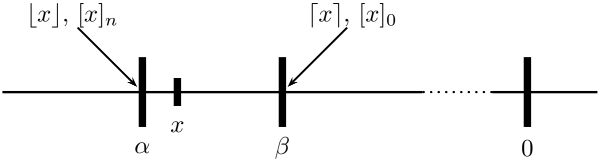
Cas où l'une des résistances est nulle ?

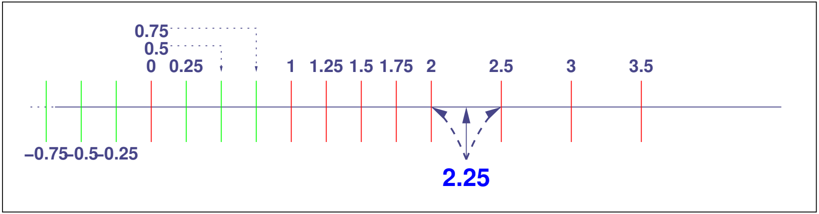
➟ Résistance totale nulle

- Ensemble des flottants $\mathbb{F}$ non clos pour les opérations de base
- conversion exacte décimal/binaire pas toujours possible

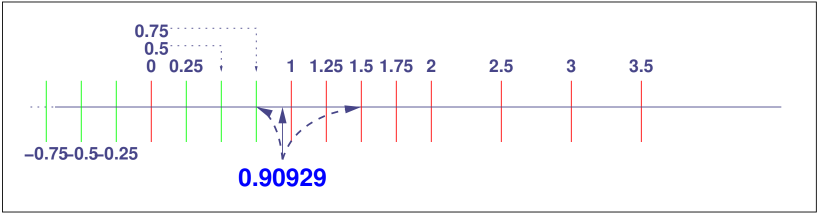
$\Rightarrow$ Fonctions d'arrondi :

$$\begin{cases} \lfloor x \rfloor & \text{arrondi vers } -\infty \\ \lceil x \rceil & \text{arrondi vers } +\infty \\ [x]_0 & \text{arrondi vers } 0 \\ [x]_n & \text{arrondi au plus proche pair} \end{cases}$$





- Opérations $\{+, -, \times, /, \sqrt{}\}$:
  arrondi correct suivant le mode courant
  (erreur < 1 *ulp.*) ;
  - Exemple : $2.0 + 0.25 = 2.25$ arrondi à 2 ou 2.5



- Opérations $\{+, -, \times, /, \sqrt{}\}$:
  arrondi correct suivant le mode courant
  (erreur < 1 *ulp.*) ;
  - Exemple : $2.0 + 0.25 = 2.25$ arrondi à 2 ou 2.5
- fonctions transcendantales : ***aucune garantie***
  - Exemple : $\sin(2.0) = 0.90929$ arrondi vers 0.75 ou 1.5

Addition en nombres flottants

- Addition possible si et seulement si les opérandes ont même exposant
- Exposants différents ⇒ décalage du nombre de *plus petit* exposant

Exemple :

$$\begin{array}{r} 10.375 \\ +\ 6.34375 \\ \hline 16.71875 \end{array} \dashrightarrow \begin{array}{r} 1.0100110 \times 2^3 \\ +\ 1.1001011 \times 2^2 \\ \hline \end{array} \dashrightarrow \begin{array}{r} 1.0100110 \times 2^3 \\ +\ 0.1100101 \times 2^3 \\ \hline 10.0001011 \times 2^3 \end{array}$$

$$16.6875 = 1.0000101 \times 2^4$$

Attention : bit de garde

Soustraction en nombres flottants

- Soustraction possible si et seulement si les opérandes ont même exposant
- Exposants différents $\Rightarrow$ décalage du nombre de *plus petit* exposant

Exemple :

$$
\begin{array}{r}
16.75 \\
-\ 15.9375 \\
\hline
0.8125
\end{array}
\quad \dashrightarrow \quad
\begin{array}{r}
1.0000110 \text{ x } 2^4 \\
-\ 1.1111111 \text{ x } 2^3 \\
\hline
\end{array}
\quad \dashrightarrow \quad
\begin{array}{r}
1.0000110 \text{ x } 2^4 \\
-\ 0.1111111 \text{ x } 2^4 \\
\hline
0.0000111 \text{ x } 2^4
\end{array}
$$

$$0.875$$

Attention : bit de garde

Multiplication en nombres flottants

- Multiplication des signifiants et ajout des exposants

Exemple :

$$\begin{array}{lr} & 10.375 \\ \times & 2.5 \\ \hline & 25.9375 \end{array} \qquad \begin{array}{lr} & 1.0100110 \times 2^3 \\ \times & 1.0100000 \times 2^1 \\ \hline & 10100110 \\ & 10100110 \\ \hline & 1.10011111000000 \times 2^4 \end{array}$$

$$25.875 \longleftarrow 1.1001111 \times 2^4$$

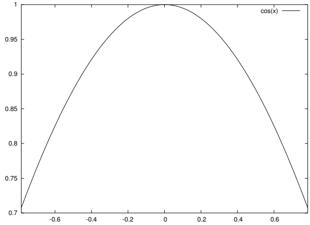
Division en nombres flottants

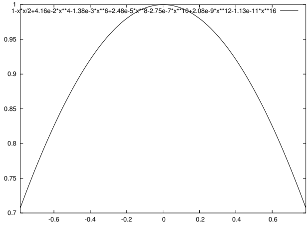
- Procédure plus compliquée
- Parfois implémentée par utilisation de la méthode de Newton

- Fonctions cos, sin, exp, ... :
  - Utilisation d'approximations polynomiales
  - Implémentation dans une librairie (e.g. en C)
  - Implémentation codée dans l'*unité arithmétique flottante*

Exemple : Pour $x \in [-\frac{\pi}{4}, \frac{\pi}{4}]$:

$$\cos x \approx 1 - \frac{x^2}{2} + 4.16 \cdot 10^{-2} x^4 - 1.38 \cdot 10^{-3} x^6$$
$$+ 2.48 \cdot 10^{-5} x^8 - 2.75 \cdot 10^{-7} x^{10} + 2.08 \cdot 10^{-9} x^{12} - 1.13 \cdot 10^{-11} x^{16}$$





Erreurs

- *Arrondi*. $$\left[\begin{array}{l} x \to \widetilde{x} = x(1+\delta), \quad \delta \le \varepsilon/2 \\ x + y \to x \oplus y = (x+y)(1+\delta) \end{array}\right.$$
- *Absorption*.
  Addition opérandes de magnitudes différentes :
  $$1.345 \cdot 10^5 + 1.45 \cdot 10^1 = 1.345 \cdot 10^5$$
  Alignement $\Rightarrow$ chiffres significatifs de $1.45 \cdot 10^1$ éliminés.
- *Cancellation bénigne*. Voir arrondis.
- *Cancellation catastrophique*.
  Soustraction opérandes entachés d'erreurs :
  $$a = 1.22, b = 3.34, c = 2.28, b^2 - 4ac = 11.2 - 11.1 = .1 \neq .0292$$

Propriétés

$$x \oplus y = y \oplus x \qquad \text{(commutativité de l'addition)}$$

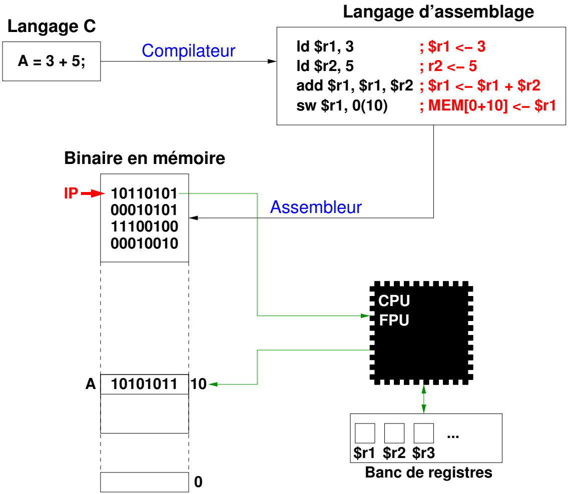
$$x \otimes y = y \otimes x \qquad \text{(commutativité de la multiplication)}$$

$$x \oplus (y \oplus z) \neq (x \oplus y) \oplus z \qquad \text{(non associativité)}$$

$$x \otimes (y \oplus z) \neq x \otimes y \oplus x \otimes z \qquad \text{(non distributivité)}$$

$\Rightarrow$ ordre des calculs pas indifférent. (e.g. $\sum_{i=1}^{n} x_i$)

Codage des instructions machine (1)



Codage des instructions machine (2)

Deux types d'architecture :

	RISC *Reduced Instruction Set Computer*	CISC *Complex Instruction Set Computer*
Nb. d'instr.	Peu	Beaucoup
Taille d'instr.	Fixe	Variable
Arité des instr.	Fixe	Variable
Adressage	Peu	Beaucoup

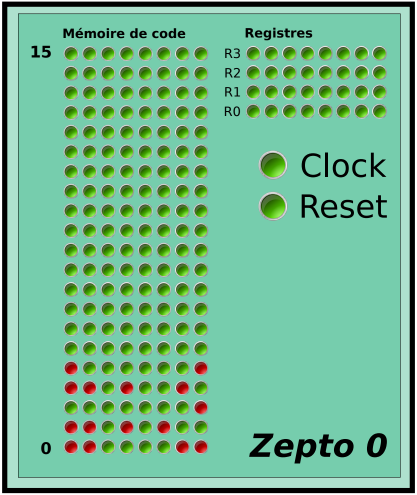
- RISC (Sparc, MIPS) : instructions simples et rapides
  - Optimisations faciles
  - Programmes complexes longs
- CISC (ix86) : instructions plus ou moins complexes
  - Programmes plus petits
  - Optimisation plus difficiles

Types de jeux d'instructions

$A \leftarrow 3 + 5$?

Machine à pile	Machine à accumulateur	Machine à registres
PUSH X $top(pile) \leftarrow X$	LOAD X $acc \leftarrow X$	LOAD R_i, X $R_i \leftarrow X$
POP X $X \leftarrow top(pile)$	STORE X $X \leftarrow acc$	STORE R_i, X $X \leftarrow R_i$
ADD $POP\ t_2$; $POP\ t_1$ $PUSH\ t_1 + t_2$	ADD X $acc \leftarrow acc + X$	ADD R_i, R_j, R_k $R_i \leftarrow R_j + R_k$

push 3 push 5 add pop A	load 3 add 5 store A	load R1, 3 load R2, 5 add R3, R1, R2 store R3, A

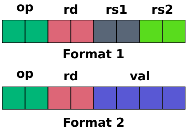


Zepto 0

- 4 registres de 8 bits
- Mémoire de code de 16 octets
- 4 instructions : `add`, `sub`, `mul`, `load`

Format d'instructions

Instr	op
add	00
sub	01
mul	10
load	11



Exemple de programme

Calcul de (3+4)*2

```
load R0, 3        # R0 <- 3
load R1, 4        # R1 <- 4
add R0, R0, R1    # R0 <- R0 + R1
load R1, 2        # R1 <- 2
mul R0, R0, R1    # R0 <- R0 x R1
```

Résultat dans R0

Codage du programme

1	1	0	0	0	0	1	1
1	1	0	1	0	1	0	0
0	0	0	0	0	0	0	1
1	1	0	1	0	0	1	0
1	0	0	0	0	0	0	1

Circuit du Zepto-0