5. Codage des nombres entiers relatifs

Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d'Informatique

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Trouvé une erreur sur un transparent? Envoyez-moi un message

- *sur Twitter @teachjl ou*
- *par e-mail jonas.latt@unige.ch*

Contenu du cours

Partie I: Introduction

Partie II: Codage de l'information

Partie III: Circuits logiques

Partie IV: Architecture des ordinateurs

Soustractions effectuées **à l'aide d'additions**

	5	~~4~~ 3	~~3~~ 12	~~2~~ 11	11
-	1	2	3	4	5
	4	1	9	7	6

Appris à l'école:
Méthode de compensation

		5	4	3	2	1
+	(-	1	2	3	4	5)

Méthode du complément:
Calcul d'une addition

Nouveau codage: Représentation de «-12345» par un nombre positif

-	1	2	3	4	5
	↓	↓	↓	↓	↓
	8	7	6	5	4

Etape 1: remplacer chaque chiffre x par son complément à 10 (9-x)

+ 1

= 8 7 6 5 5

Etape 2: Calcul +1

		5	4	3	2	1
+		8 [1]	7	6	5	5
	~~1~~	4	1	9	7	6

Codage à taille fixe:
"Ce qui dépasse part à la poubelle"

Débordement

Jonas Lätt

Objectifs de la leçon

1. Codage et décodage des entiers relatifs: la règle du complément à 2.

“Nous savons coder un entier naturel. Il suffit donc de représenter les nombres négatifs par des nombres positifs.”

2. Compréhension et interprétation des bits d’un nombre codé.

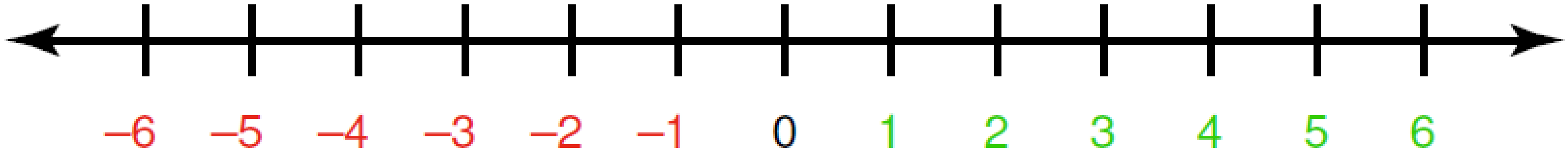
“A partir des 0/1 qui représentent x, comment trouver les 0/1 qui représentent –x ?”

3. Additions et soustractions de nombres vues par l’ordinateur (en représentation interne).

“Comment fera l’ordinateur pour trouver les 0/1 qui représentent x+y, ou x-y ?

PARTIE I:
PRINCIPES DU CODAGE DES NOMBRES RELATIFS

Les nombres entiers relatifs



$$\mathbb{Z} = \{\dots - 2, -1, 0, 1, 2, \dots\}$$

Première idée: Codage en signe-norme

Idée : on code séparément le signe (bit=1 si le signe est négatif) et la norme du nombre

$\text{code}_{\mathbb{N}(4)}$	code signe-norme
15	-7
14	-6
13	-5
12	-4
11	-3
10	-2
9	-1
8	-0
7	7
6	6
5	5
4	4
3	3
2	2
1	1
0	0

Représentation externe

signe négatif

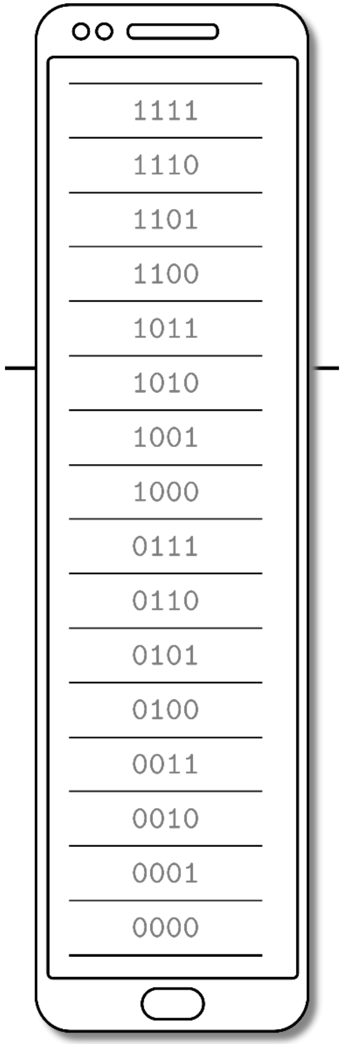
1111
1110
1101
1100
1011
1010
1001
1000
0111
0110
0101
0100
0011
0010
0001
0000

Repr. interne

MSB: Bit de poids fort
0 → nombre positif
1 → nombre négatif
Les bits restants codent la norme du nombre

Première idée: Codage en signe-norme

$\texttt{code}_{\mathbb{N}(4)}$	Signe-Norme
15	-7
14	-6
13	-5
12	-4
11	-3
10	-2
9	-1
8	-0
7	7
6	6
5	5
4	4
3	3
2	2
1	1
0	0

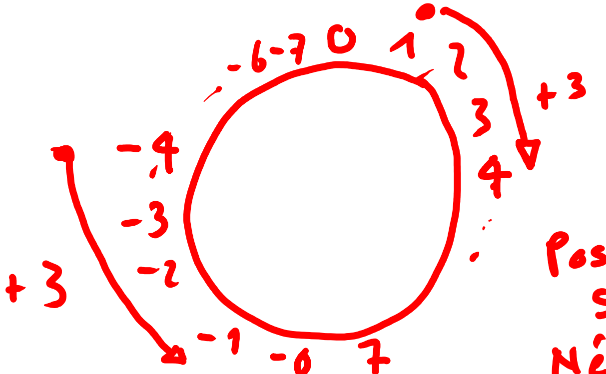


Problèmes …

1. Ambiguïté : il existe 2 manières de représenter le zéro

2. La représentation cyclique des nombres est inconsistente

addition +3

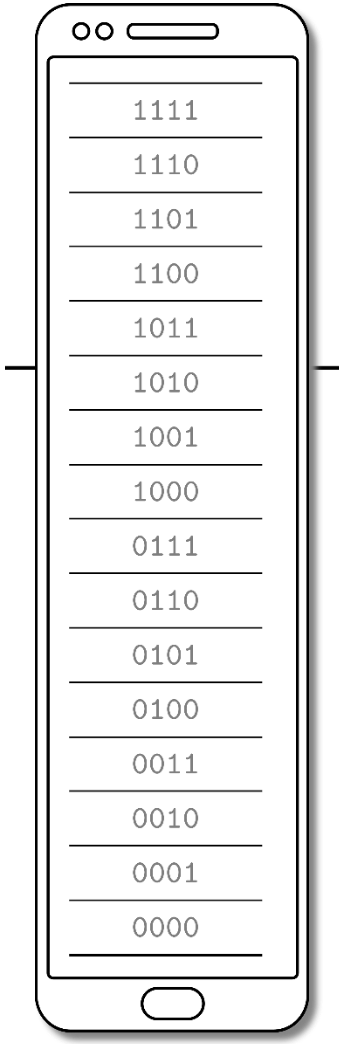


Positifs : dans le sens de l'horloge
Négatifs : sens inverse

Meilleure idée: Codage par complément à 2

Codage par complément $\text{code}_{\mathbb{Z}(N)}$

$\text{code}_{\mathbb{N}(4)}$	Signe-Norme	$\text{code}_{\mathbb{Z}(4)}$
15	-7	-1
14	-6	-2
13	-5	-3
12	-4	-4
11	-3	-5
10	-2	-6
9	-1	-7
8	~~-0~~	-8
7	7	7
6	6	6
5	5	5
4	4	4
3	3	3
2	2	2
1	1	1
0	0	0



1. Pour les nombres naturels 0-7 $(0 - 2^{K-1}-1)$, on fait la même chose qu'en signe-norme, et qu'en $\mathbb{N}(4)$ (en $\mathbb{N}(N)$)
2. On élimine le -0
3. On redresse les nombres négatifs (on les fait avancer dans le même sens que les positifs)

Jonas Lätt

Meilleure idée: Codage par complément à 2

Nombres négatifs
$-2^{k-1} \leq x < 0$

$\text{code}_{\mathbb{N}(4)}$	Signe-Norme	$\text{code}_{\mathbb{Z}(4)}$	
15	-7	-1	1111
14	-6	-2	1110
13	-5	-3	1101
12	-4	-4	1100
11	-3	-5	1011
10	-2	-6	1010
9	-1	-7	1001
8	-0	-8	1000
7	7	7	0111
6	6	6	0110
5	5	5	0101
4	4	4	0100
3	3	3	0011
2	2	2	0010
1	1	1	0001
0	0	0	0000

Nombres positifs
$0 \leq x < 2^{k-1}$

En $\text{code}_{\mathbb{Z}(4)}$, le MSB reste un indicateur de signe

$\text{code}_{\mathbb{Z}(4)}(-8) = \text{code}_{\mathbb{N}(4)}(8)$

$\text{code}_{\mathbb{Z}(4)}(-7) = \text{code}_{\mathbb{N}(4)}(9)$

$\text{code}_{\mathbb{Z}(4)}(-6) = \text{code}_{\mathbb{N}(4)}(10)$

en général, pour les valeurs $x<0$:

$\text{code}_{\mathbb{Z}(4)}(x) = \text{code}_{\mathbb{N}(4)}(x+16)$

en k bits: $x<0$

$\text{code}_{\mathbb{Z}(k)}(x) = \text{code}_{\mathbb{N}(k)}(x+2^k)$

Le sous-ensemble des nombres entiers relatifs $\mathbb{Z}_{(k)}$

Rappel sur $\mathbb{N}_{(k)}$: $\mathbb{N}_{(k)} = \{x \in \mathbb{N} |\ 0 \leq x < 2^k\}$

$\mathbb{Z}_{(k)}$: «On décale $\mathbb{N}_{(k)}$ à gauche» $\mathbb{Z}_{(k)} = \{x \in \mathbb{Z} |\ -2^{k-1} \leq x < 2^{k-1}\}$.

-2^7 2^7-1

Exemple: $\mathbb{Z}_{(8)} = \{-128\ \ldots 127\}$.

Le cours reprend à 14:14

A l'aide de la règle du complément à 2, on peut représenter tous les nombres dans l'intervalle $\mathbb{Z}_{(k)}$ par un *mot* à *k* bits.

PARTIE II:
COMPLÉMENT À 2 – RÈGLES DE CODAGE ET DÉCODAGE

Complément à 2: règle de codage

$\mathtt{code}_{\mathbb{N}_{(4)}}$	Signe-Norme	$\mathtt{code}_{\mathbb{Z}_{(4)}}$	
15	-7	-1	1111
14	-6	-2	1110
13	-5	-3	1101
12	-4	-4	1100
11	-3	-5	1011
10	-2	-6	1010
9	-1	-7	1001
8	-0	-8	1000
7	7	7	0111
6	6	6	0110
5	5	5	0101
4	4	4	0100
3	3	3	0011
2	2	2	0010
1	1	1	0001
0	0	0	0000

Nombres négatifs
$-2^{k-1} \leq x < 0$

Nombres positifs
$0 \leq x < 2^{k-1}$

Codage par complément à 2:

$\mathtt{code}_{\mathbb{Z}_{(k)}}$

$$\mathtt{code}_{\mathbb{Z}_{(k)}} : x \longmapsto \begin{cases} \mathtt{code}_{\mathbb{N}_{(k)}}(x) & \text{si } 0 \leq x < 2^{k-1} \\ \mathtt{code}_{\mathbb{N}_{(k)}}(x + 2^k) & \text{si } -2^{k-1} \leq x < 0 \end{cases}$$

Idée :

- Exprimer le rapport $\mathbb{Z}_{(k)} = \mathbb{N}_{(k)}$
- Appliquer le codage en $\mathbb{N}_{(k)}$

Interprétation du bit de poids fort MSB

Nombres négatifs
$-2^{k-1} \leq x < 0$

code$_{\mathbb{N}(4)}$	Signe-Norme	code$_{\mathbb{Z}(4)}$	
15	-7	-1	1111
14	-6	-2	1110
13	-5	-3	1101
12	-4	-4	1100
11	-3	-5	1011
10	-2	-6	1010
9	-1	-7	1001
8	-0	-8	1000
7	7	7	0111
6	6	6	0110
5	5	5	0101
4	4	4	0100
3	3	3	0011
2	2	2	0010
1	1	1	0001
0	0	0	0000

Nombres positifs
$0 \leq x < 2^{k-1}$

Rappel : le MSB est un indicateur de signe

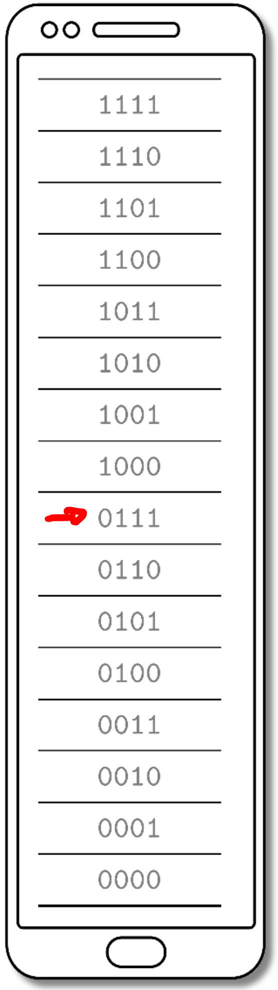
MSB = 1 → Nombre négatif

MSB = 0 → Nombre positif ou nul

Décodage en complément à 2

Nombres négatifs
$-2^{k-1} \leq x < 0$

$\texttt{code}_{\mathbb{N}(4)}$	Signe-Norme	$\texttt{code}_{\mathbb{Z}(4)}$
15	-7	-1
14	-6	-2
13	-5	-3
12	-4	-4
11	-3	-5
10	-2	-6
9	-1	-7
8	-0	-8
7	7	7
6	6	6
5	5	5
4	4	4
3	3	3
2	2	2
1	1	1
0	0	0



Nombres positifs
$0 \leq x < 2^{k-1}$

Codage par complément à 2:

$$\texttt{code}_{\mathbb{Z}(k)}$$

$$\texttt{code}_{\mathbb{Z}(k)} : x \longmapsto \begin{cases} \texttt{code}_{\mathbb{N}(k)}(x) & \text{si } 0 \leq x < 2^{k-1} \\ \texttt{code}_{\mathbb{N}(k)}(x + 2^k) & \text{si } -2^{k-1} \leq x < 0 \end{cases}$$

séquence 1/0

$$\text{decode}_{\mathbb{Z}(k)} : X \longmapsto \begin{cases} \text{decode}_{\mathbb{N}(k)}(x) & \text{si MSB} = 0 \\ \text{decode}_{\mathbb{N}(k)}(x) - 2^k & \text{si MSB} = 1 \end{cases}$$

Jonas Lätt

Codage et Décodage: Exemples

Plaçons-nous dans l'espace $\mathbb{Z}_{(8)}$, dans lequel les nombres de -128 … 127 sont codés sur des mots d'un octet (8 bits).

$\text{code}_{\mathbb{Z}_{(8)}}(80) =$ $\text{code}_{\mathbb{N}_{(8)}}(80) = 0101\ 0000$

(80 = 64 + 16)

(176 = $\underline{128 + 32 + 16}$

176 − 128 = 48

48 − 32 = 16

$\text{code}_{\mathbb{Z}_{(8)}}(-80) =$ $\text{code}_{\mathbb{N}_{(8)}}(-80+256) = \text{code}_{\mathbb{N}_{(8)}}(176)$

$= 1011\ 0000$

négatif

$\text{decode}_{\mathbb{Z}_{(8)}}(1010\ 0101) =$ $\text{decode}_{\mathbb{N}_{(8)}}(1010\ 0101) - 256$

$= 1 + 4 + 32 + \underbrace{128 - 256}_{-128}$ $= 1 + 4 + 32 - 128 = 37 - 128 = -91$

C'est comme si le MSB avait un poids négatif
Son poids vaut −128 au lieu de 128. En général,
Son poids vaut -2^{k-1} au lieu de 2^{k-1}

en 8 bits, si le nbre est négatif,
le −128 est toujours là
en k bits ça sera -2^{k-1}

Décodage en $\mathbb{Z}_{(k)}$

Interprétation: du décodage en $\mathbb{Z}_{(k)}$: le bit de poids élevé possède un poids négatif:

Décodage en $\mathbb{N}_{(k)}$: $x = [a_0 2^0 + a_1 2^1 + \cdots + a_{N-2} 2^{k-2} + \underline{a_{N-1} 2^{k-1}}$

Décodage en $\mathbb{Z}_{(k)}$: $x = a_0 2^0 + a_1 2^1 + \cdots + a_{N-2} 2^{k-2} \underline{- a_{N-1} 2^{k-1}}$

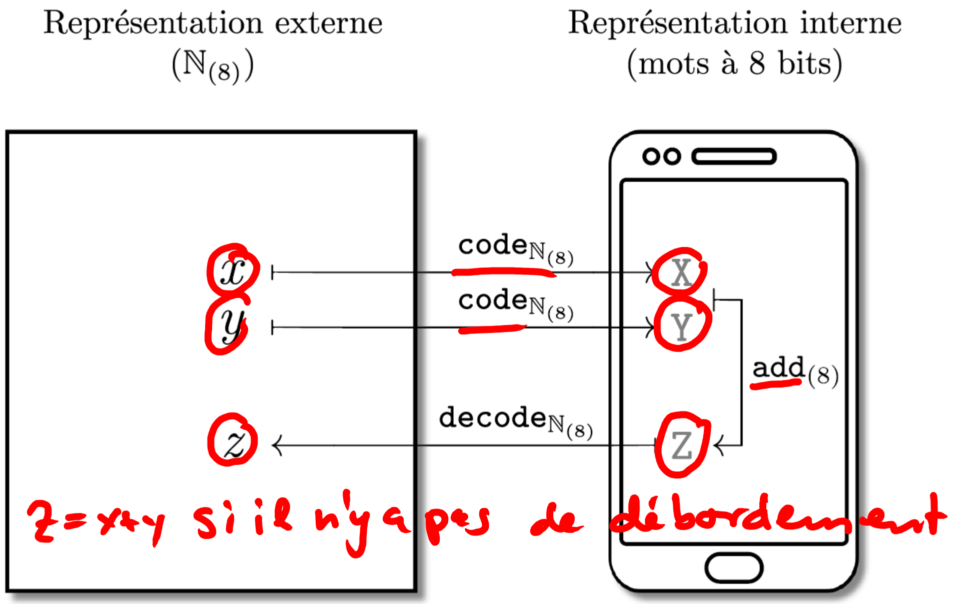
Il s'agit d'une manière intéressante d'interpréter, et donner une signification, à la règle de codage par complément à 2.

PARTIE III:
ADDITION DE NOMBRES RELATIFS

Addition en $\mathbb{Z}_{(k)}$

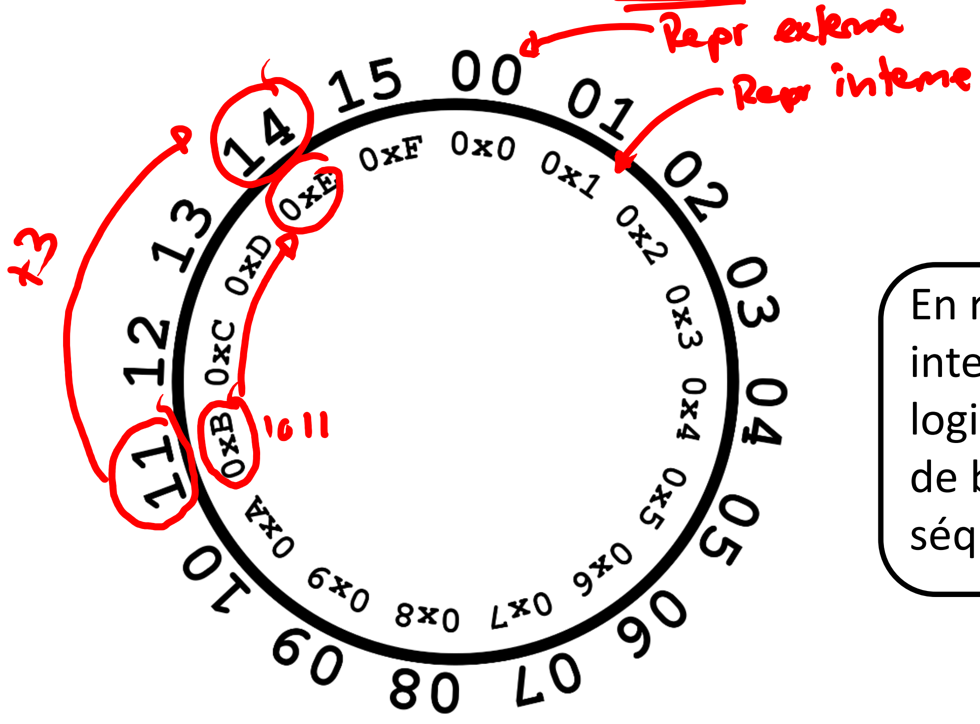
Idée

- On n'informe pas le processeur de l'ordinateur si les données représentées sont en $\mathbb{N}_{(k)}$ ou en $\mathbb{Z}_{(k)}$.
- Le même circuit additionneur s'utilise dans les deux cas.
- Tour de magie: ça marche en $\mathbb{N}_{(k)}$ et en $\mathbb{Z}_{(k)}$!

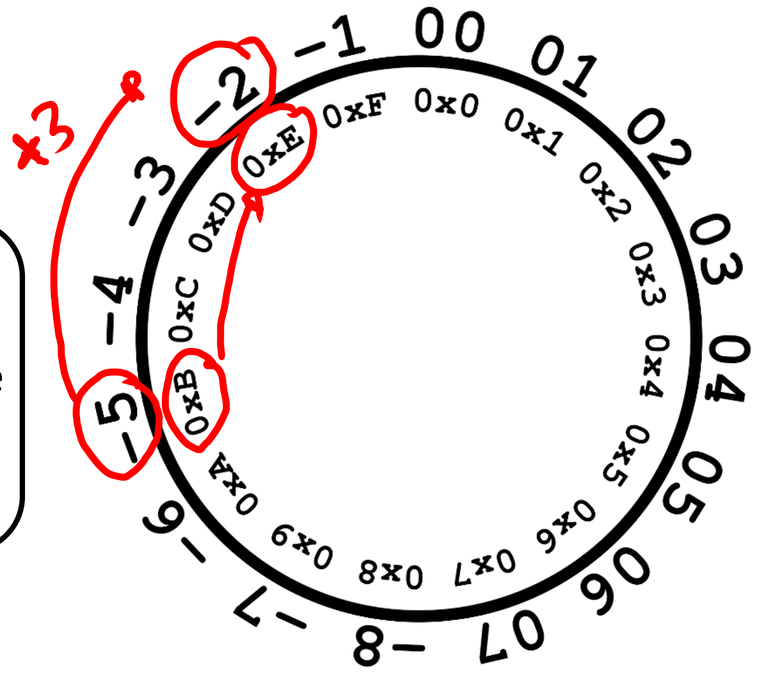


Equivalence de l'addition / soustraction entre $\mathbb{N}_{(4)}$ et $\mathbb{Z}_{(4)}$

Entiers naturels $\mathbb{N}_{(4)}$



Entiers relatifs $\mathbb{Z}_{(4)}$ (complément à 2)



En représentation interne, une fonction logique lit une séquence de bits et produit une séquence de bits.

11 + 3 = 14

0xB -> 0xE

-5 + 3 = -2

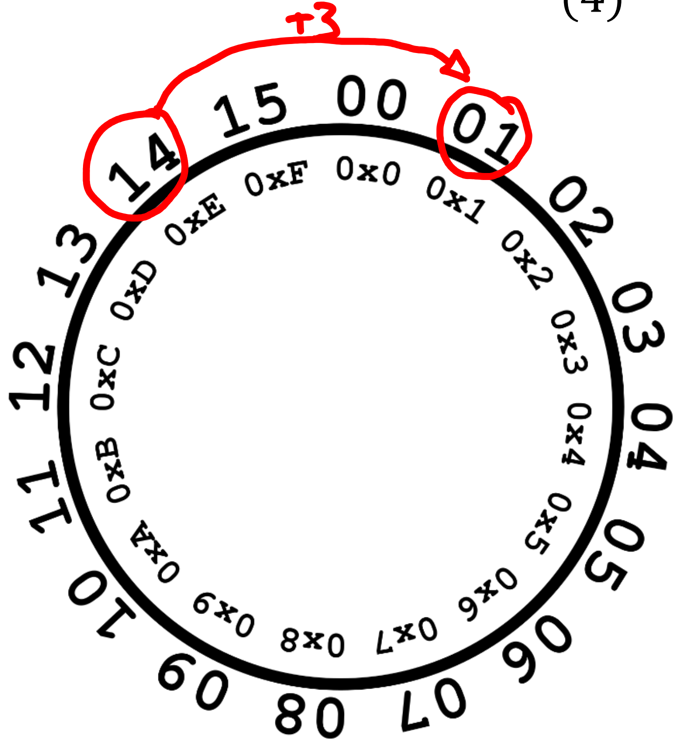
L'opération effectuée peut s'interpréter comme une addition en $\mathbb{N}_{(4)}$ …

… ou alors, comme une addition en $\mathbb{Z}_{(4)}$.

Jonas Lätt

Equivalence de l'addition / soustraction entre $\mathbb{N}_{(4)}$ et $\mathbb{Z}_{(4)}$

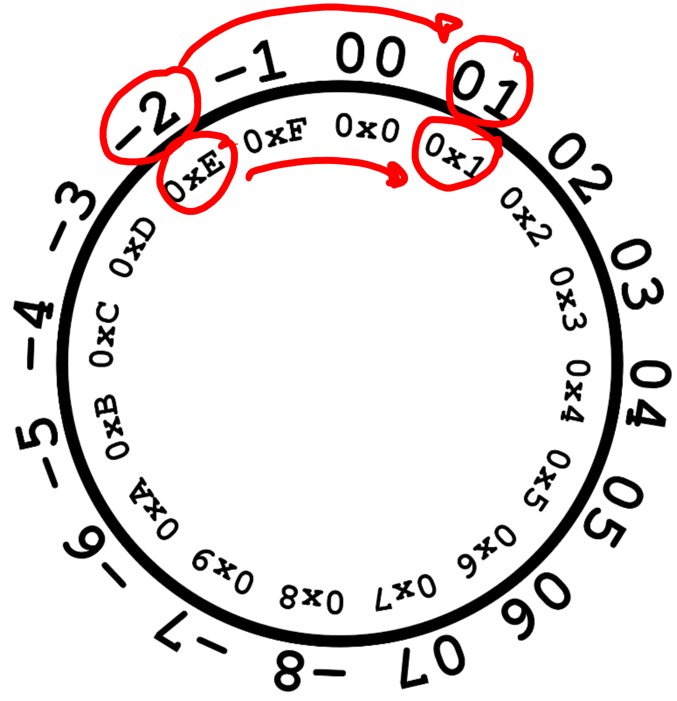
Entiers naturels $\mathbb{N}_{(4)}$



14 + 3 = 17 (overflow) -> 1

$\boxed{\text{0xE -> 0x1}}$

Entiers relatifs $\mathbb{Z}_{(4)}$ (complément à 2)

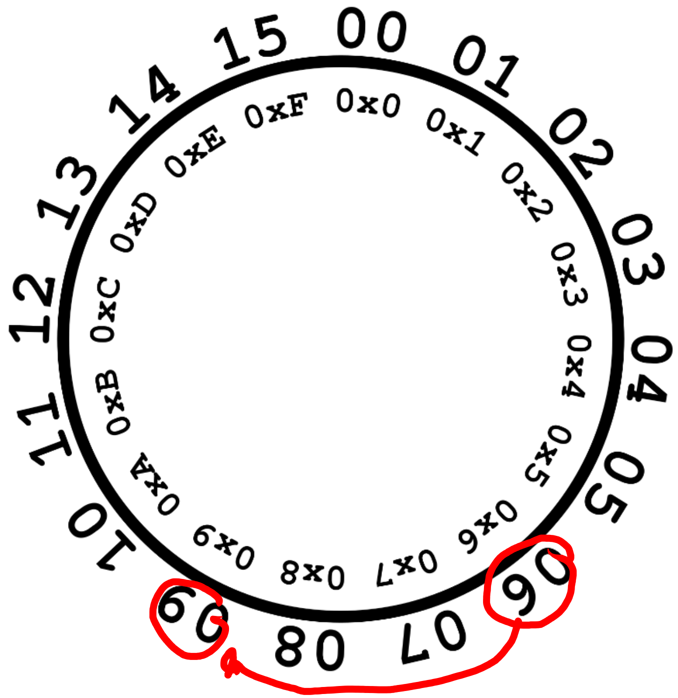


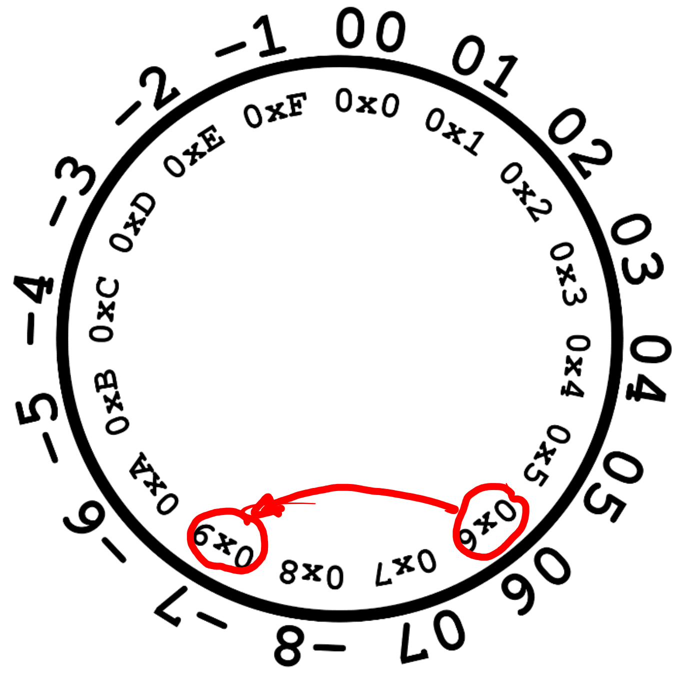
-5 + 3 = -2

Equivalence de l'addition / soustraction entre $\mathbb{N}_{(4)}$ et $\mathbb{Z}_{(4)}$

Entiers naturels $\mathbb{N}_{(4)}$

Entiers relatifs $\mathbb{Z}_{(4)}$ (complément à 2)





6 + 3 = 9

$\boxed{\text{0x6 -> 0x9}}$

6 + 3 = ? Débordement en $\mathbb{Z}_{(4)}$

6 + 3 = 9 (Overflow) -> -7

→ produit la valeur $-7 = 9 - 16$

lorsque $a+b \geq 2^{k-1}$ le résultat est $a+b-2^k$

On définit une règle de débordement en $\mathbb{Z}_{(4)}$

Règles de débordement en $\mathbb{Z}_{(k)}$

$$\text{Addition en } \mathbb{Z}_{(\text{k})} : \quad x, y \longmapsto \begin{cases} \underline{x+y+2^k} & \text{si } x+y < -2^{k-1} \\ \underline{x+y-2^k} & \text{si } x+y \geq 2^{k-1} \\ x+y & \text{sinon} \end{cases}$$

Avantages du complément à 2

1. Le circuit **add** peut effectuer des additions pour des valeurs en $\mathbb{N}_{(k)}$ tout comme en $\mathbb{Z}_{(k)}$: le circuit n'a même pas besoin de savoir lequel des deux codages est utilisé.
2. A voir dans la prochaine partie: le circuit **add** permet d'effectuer non seulement des additions, mais aussi des soustractions, en $\mathbb{N}_{(k)}$ tout comme en $\mathbb{Z}_{(k)}$.

PARTIE IV:
SOUSTRACTION DE NOMBRES ENTIERS

Rappel: algorithme de soustraction de nombres

- Technique classique: méthode de compensation.
- Exemple: La soustraction z = x - y = 4321 – 1234.

	4	3	2	1
-	1	2	3	4

Cet algorithme peut être réalisé dans un ordinateur, mais mène à un circuit complexe. On cherche une solution techniquement plus simple pour l’ordinateur.

Soustraction de nombres entiers

Le circuit add peut être réutilisé pour la soustraction de nombres (on peut utiliser **add** pour réaliser le circuit de soustraction **sub**)!

- Si les arguments de sub sont des représentations de nombres relatifs: Pour tout argument Y qui représente un nombre y, on peut trouver une séquence binaire Y_O qui représente la valeur opposée $-y$.
- La soustraction $z = x - y$ peut s'écrire $z = x + (-y)$: Il est possible d'effectuer la soustraction à l'aide du circuit additionneur **add**.
- Si les arguments de **sub** représentent des nombres entiers naturels, il suffit de les *interpréter* comme des entiers relatifs, et l'argument ci-dessus reste valable. Comme nous l'avons vu au chapitre précédent, le circuit **add** ne fait pas la difference entre entiers naturels et relatifs.

Comment calculer la valeur opposée de y?

Y est la representation de y.

Comment trouver Y_O , qui est la représentation de $-y$, en manipulant les bits de Y? Tout ça doit se faire au niveau d'un circuit, donc en représentation interne.

Codage en complément à 2

Nombres négatifs
$-2^{k-1} \leq x < 0$

$\texttt{code}_{\mathbb{N}(4)}$	Signe-Norme	$\texttt{code}_{\mathbb{Z}(4)}$	
15	-7	-1	1111
14	-6	-2	1110
13	-5	-3	1101
12	-4	-4	1100
11	-3	-5	1011
10	-2	-6	1010
9	-1	-7	1001
8	-0	-8	1000
7	7	7	0111
6	6	6	0110
5	5	5	0101
4	4	4	0100
3	3	3	0011
2	2	2	0010
1	1	1	0001
0	0	0	0000

Nombres positifs
$0 \leq x < 2^{k-1}$

"Passage d'une valeur x vers son opposé":

- Calculer $16 - x$
- Calculer $2^k - x$

Comment calculer la valeur opposée de y?

Y est la representation de y.

Comment trouver $\mathrm{Y_O}$, qui est la représentation de $-y$, en manipulant les bits de Y?

En représentation externe $\mathbb{N}_{(k)}$, c'est facile:

- On cherche la valeur decode$_{\mathbb{N}_k}(\mathrm{Y_O}) = 2^k - y$

Mais en représentation interne ??

Mais en représentation interne? Astuce:

$$\boxed{\text{decode}_{\mathbb{N}_k}(\text{Y}_\text{O}) = 2^k - y = 1 + (2^k - 1) - y}$$

Astuce: représentation de -y

Entrée: Codage de y, Y

Sortie: Codage de $-y$, Y_O

$11111111_{(2)}$

On aligne k fois le "1".

Astuce: $2^k - y = 1 + (2^k - 1) - y$

$$
\begin{array}{r}
\;1\;1\;1\;1\;1\;1\;1\;1 \\
-\;0\;1\;1\;0\;1\;0\;0\;1 \\
\hline
1\;0\;0\;1\;0\;1\;1\;0
\end{array}
$$

Il n'y a pas de retenue!

Il suffit d'inverser tous les bits de x.

Calcul du complément à 2

Ce procédé en deux étapes s'appelle le *calcul du complément à 2*:

1. Inversion de tous les bits
2. Addition de la valeur 1 au résultat

Complément à 2: représentation de -x

Règle:

- On inverse tous les bits de x.
- On ajoute 1 au résultat.

Exemple. Entrée: La valeur 3 codée en $\mathbb{Z}_{(4)}$.

0011 — Le codage de la valeur 3

1100 — Inversion des bits

1101 — +1

1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

Le circuit C2: Calcul du complément à 2

Le circuit $C2_{(k)}$

- Inverse tous les bits
- Rajoute la valeur 1 au résultat

Cela permet de soustraire deux nombres, $z = x - y$, à l'aide du circuit add:

$$\mathtt{Z} = \mathtt{add}_{(k)}(\mathtt{X}, \mathtt{C2}(\mathtt{Y}))$$

Terminologie: Complément à 2

Le terme “complément à 2” s’utilise de différentes manières:

- Un nombre relatif est codé selon la **règle du complément à 2** lorsqu’on applique le codage $\texttt{code}_{\mathbb{Z}(k)}$
- On **calcule le complément à 2** d’un séquence de bits en appliquant l’opérateur $C2_{(k)}$: inversion des bits, calcul +1.
- La **méthode du complément** est une méthode qui permet de soustraire deux nombres en n’effectuant que des additions.