

Architecture des ordinateurs

Cours 2

Représentation des nombres entiers relatif

Opérations arithmétiques

Représentation des caractères et chaînes de caractères

Plan du cours 2

- 1 Rappels
- 2 Arithmétique sur entiers naturels
- 3 Représentation des entiers relatifs et arithmétique entière
- 4 Représentation des caractères alphanumériques

Codage des entiers naturels en base 2

Interprétation d'un mot binaire comme un entier naturel (codage entiers naturels)

$$(a_{n-1} \dots a_1 a_0)_b = N_d = (\sum_{i=0}^{n-1} a_i 2^i)_d$$

Intervalle de représentation (codage entiers naturels)

- Sur n bits, on peut représenter les entiers compris dans $[0, 2^n-1]$

Extension d'un mot binaire représentant un entier naturel (codage entiers naturels)

- Extension de n à $p > n$ bits en recopiant les n bits de poids faible et en mettant 0 sur les $p - n$ bits de poids fort :

$$a_{n-1} \dots a_1 a_0 \rightarrow 0 \dots 0 a_{n-1} \dots a_1 a_0$$

1 Rappels

2 Arithmétique sur entiers naturels

- En base 2
- En base 16
- Addition de mots
- **Multiplication et division**

3 Représentation des entiers relatifs et arithmétique entière

4 Représentation des caractères alphanumériques

Addition d'entiers naturels : en décimal

Principe et règles

- Addition des chiffres de même rang, en commençant par la droite
- Quand l'addition de deux chiffres dépasse 10, propagation d'une retenue à gauche
- La retenue entrante (sortante du rang précédent) est à sommer avec les chiffres du rang courant

Exemple

$$\begin{array}{r} 7^1 \quad 5 \\ + \quad 1 \quad 7 \\ \hline 9 \quad 2 \end{array}$$

- Pour les unités : “ $7 + 5 = 12$: je pose 2 et je retiens 1”
- Pour les dizaines : il faut intégrer la retenue
“ $7 + 1 + 1 = 9$: je pose 9 (et je retiens 0)”

Addition en base 2

Principe (1)

Addition des bits de même rang

Règle sur les bits

- $0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$
- $1 + 1 = 2_d = 10_b$ donc “je pose 0 et je retiens 1”

Principe (2)

- Lorsqu’il y a une retenue, il faut l’intégrer dans le calcul sur les bits de gauche
- On a donc non pas 2 mais 3 chiffres à sommer

Règle supplémentaire sur les bits

- Addition de trois 1, et $1 + 1 + 1 = 3_d = 11_b$, soit “je pose 1 et je retiens 1”

Exemples

Premier exemple

$$\begin{array}{rcccccccc} & 1 & 0^1 & 1^1 & 0^1 & 1 & 0 & 1 & 1 \\ + & & & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

- Les 2 opérandes s'écrivent sur 8 et 6 bits respectivement
- Extension implicite du 2ème opérande pour réaliser des opérations sur 2 mots de 8 bits
- Le résultat s'écrit sur 8 bits

Deuxième exemple

$$\begin{array}{rcccc} & 1 & 1^1 & 1 & 0 & 0 \\ + & & & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 \end{array}$$

- Les 2 opérandes s'écrivent sur 4 et 3 bits respectivement
- Extension implicite du 2ème opérande pour réaliser des opérations sur des mots de 4 bits
- Le résultat s'écrit sur 5 bits : il n'est pas représentable sur 4 bits

Addition en base 16

Principe

- Addition réalisée en sommant les chiffres/symboles de même rang
- Propagation d'une retenue lorsque le résultat est strictement supérieur à $F_h = 15$

Exemples

$$\begin{array}{r} \text{D} \\ + \\ \hline \end{array}$$

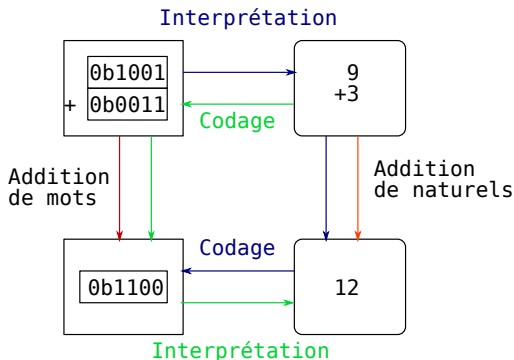
$$\begin{array}{r} \text{1}^1 \text{F} \text{A} \\ + \\ \hline \end{array}$$

$$\begin{array}{r} \\ + \\ \hline \end{array}$$

Addition d'entiers représentés sur des mots binaires

Principe

- On définit l'addition sur des mots binaires pour que l'interprétation du résultat comme entier naturel corresponde à la somme des interprétations des opérandes
- Cela correspond à l'addition binaire (modulo 2^n si les opérandes et résultat sont représentés sur des mots de n bits)



- Flèches bleues et rouge : définition de l'addition de mots binaires
- Flèche orange : ce qu'on veut faire
- Flèches vertes : calcul réalisé

Addition d'entiers représentés sur des mots binaires

Problème : dépassement de capacité

- On dit qu'il y a un **dépassement de capacité** sur entiers naturels quand l'addition de 2 mots binaires ne donnent pas le bon résultat lorsque l'on interprète ceux-ci comme des entiers naturels
- **Exemple** : $0b1001 + 0b1000$ correspond à $9 + 8 = 17$; or 17 n'est pas représentable sur un mot de 4 bits
- Equivalence avec le fait que le résultat appartienne à l'intervalle représentable : lorsque le résultat théorique de l'opération appartient à l'intervalle de représentation, l'addition de mots est toujours correcte
- **Si dépassement** : la retenue sortante est ignorée
- \Rightarrow Correspond à une arithmétique modulaire sur 2^n pour un mot de n bits
- **Exemple** : $0b1001 + 0b1000 = 0b0001 \rightarrow 1 = 17 \% 16$

Multiplication et division

- Opérations de multiplication et division de 2 entiers complexes
⇒ circuits les réalisant pas dans l'ALU

Multiplication et division par une puissance de 2 en binaire

- Multiplier ou diviser par $2^n \equiv$ décalage à gauche ou à droite de n bits
 - $N_d = (a_{p-1} \dots a_1 a_0)_b \Rightarrow 2N_d = (a_{p-1} \dots a_1 a_0 \mathbf{0})_b$
 - $N_d = (a_{p-1} \dots a_1 a_0)_b \Rightarrow (N/2)_d = (\mathbf{0} a_p a_{p-1} \dots a_1)_b$

Multiplication et division par une puissance de B en base B

- Décaler à gauche de 1 (respectivement n) revient à multiplier par la base B (respectivement par B^n)
- Décaler à droite de 1 (respectivement n) à diviser par la base B (respectivement par B^n).

Décalage et multiplication/division

Exemple en base 10

- $1024/10 = 102$: décalage à droite de 1 (division par 10^1)
- $1024/100 = 10$: décalage à droite de 2 (division par 10^2)
- $1024 \times 10 = 10240$: décalage à gauche de 1

Exemple en base 2

- $1100_b \gg 1_d = 110_b = 6_d = 12_d/2_d$
- $110_b \gg 2_d = 1_b = 1_d = 6_d/4_d$
- $11_b \ll 1_d = 110_b = 6_d = 3_d \times 2_d$
- $11_b \ll 2_d = 1100_b = 12_d = 3_d \times 4_d$

Décalage de mots binaires de 4 bits

- $0b1100 \gg 1_d = 0b0110 = 6_d = 12_d/2_d$
- $0b0110 \gg 2_d = 0b0001 = 1_d = 6_d/4_d$
- $0b0011 \ll 1_d = 0b0110 = 6_d = 3_d \times 2_d$
- $0b0011 \ll 2_d = 0b1100 = 12_d = 3_d \times 4_d$

1 Rappels

2 Arithmétique sur entiers naturels

3 Représentation des entiers relatifs et arithmétique entière

- Représentation des entiers relatifs
- Codage entiers relatifs en complément à 2
- Addition (et soustraction) d'entiers relatifs

4 Représentation des caractères alphanumériques

Représentation usuelle des entiers relatifs

(Représentation en module et signe)

Définition

- C'est la représentation courante qui utilise un signe et une valeur absolue : +3, -3, +2, -2, ...
- Un bit code le signe, les autres bits la valeur absolue

Addition/soustraction

- Complexe
- Il faut déterminer le signe pour savoir s'il faut faire une addition ou une soustraction

Autre problème

- Il y a deux représentations pour la valeur 0 : rend la comparaison de deux mots compliquée
- \Rightarrow il faut interdire une des deux valeurs (-0)

Représentation en complément à 1

Complément à 1

- En complément à 1 si $N_b = n_d$ alors $-n_d = \overline{N_b}$.

Conséquences

- L'opposé en binaire est simplement le complément
- Il y a deux zéros (que des 1 et que des 0) à gérer
- Les opérations d'addition et de soustraction sont encore plus complexes

Représentation en complément à 2

En complément à 2

- Un seul zéro
- Les traitements à appliquer pour réaliser une addition sont identiques que les nombres soient positifs ou négatifs

Utilisation

- C'est la représentation adoptée dans tous les processeurs pour représenter les entiers relatifs
- Appelée dans la suite représentation ou codage entiers relatifs en complément à 2 ou simplement entiers relatifs

Représentation entiers relatifs

Codage

En complément à deux, tout nombre est représenté par deux parties :

- un terme négatif constant ou nul
- une correction positive

Interprétation de $a_{n-1} \dots a_1 a_0 b$

$$a_{n-1} \dots a_1 a_0 b = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

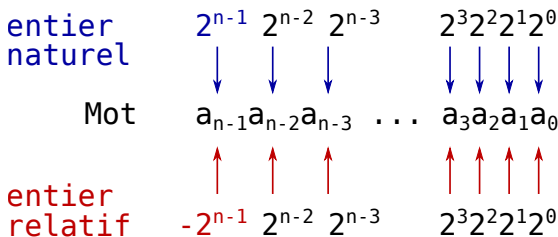
$$a_{n-1} \dots a_1 a_0 b = -a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_1 2 + a_0$$

Codage entiers relatifs

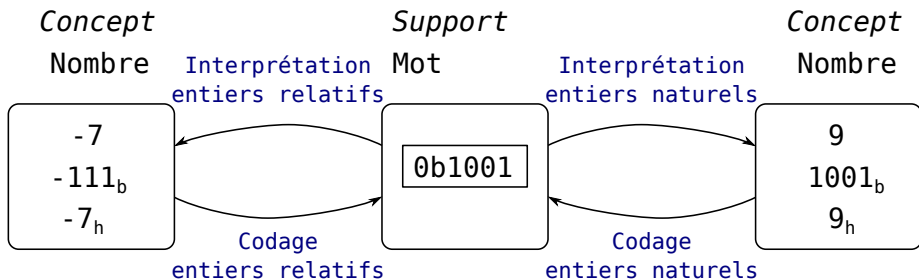
$$(a_{n-1} \dots a_1 a_0)_b = (-a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_12 + a_0)_d$$

- $-a_{n-1}2^{n-1}$ valeur négative constante ou nulle $\Rightarrow a_{n-1}$ appelé **bit de signe** : sa valeur indique le signe de l'entier
- $\sum_{i=0}^{n-2} a_i 2^i$ correspond à la correction positive

- Différence des interprétations entiers naturels et relatifs



Codage entiers relatifs



Exemple

- Sur 4 bits, le mot `0b1001` s'interprète comme la valeur 9 selon le codage **entiers naturels**, et $9 = 1001_b$
- Sur 4 bits, le mot `0b1001` s'interprète comme la valeur -7 selon le codage **entiers relatifs**

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2a_1a_0$, son **interprétation en complément à 2** est :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2a_1a_0$, son interprétation en complément à 2 est :
 $-a_2 \times 2^2 + a_1 \times 2 + a_0$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2a_1a_0$, son **interprétation en complément à 2** est :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2 a_1 a_0$, son interprétation en complément à 2 est :
 $-a_2 \times 2^2 + a_1 \times 2 + a_0$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2 a_1 a_0$, son interprétation en complément à 2 est :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2 a_1 a_0$, son interprétation en complément à 2 est :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Exemples

Exemple sur 3 bits

Soit un mot de 3 bits $a_2 a_1 a_0$, son interprétation en complément à 2 est :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = 0b000 = 0_d$
- $N = 0b101 = -2^2 + 1 = -3$
- $N = 0b110 = -2^2 + 2 = -2$
- $N = 0b111 = -2^2 + 2 + 1 = -1$
- $N = 0b011 = 2 + 1 = 3$
- Repérer
 - Entier positif \Leftrightarrow le bit de signe a_2 vaut 0
 - Entier négatif \Leftrightarrow le bit de signe a_2 vaut 1

Intervalle de représentation

Intervalle de représentation sur n bits

- Sur n bits, en complément à 2, l'intervalle de représentation est :

$$[-2^{n-1}, 2^{n-1} - 1]$$

- Si $a_{n-1} = 0 \Rightarrow$ entier naturel sur $n - 1$ bits
 \Rightarrow dans l'intervalle $[0, 2^{n-1} - 1]$ (cf. cours 1)
- Si $a_{n-1} = 1 : N = -2^{n-1} + \text{corr}$ avec corr la correction
et $\text{corr} \in [0, 2^{n-1} - 1]$
 - si $\text{corr} = 0 \Rightarrow$ c'est le plus petit entier négatif possible, il vaut -2^{n-1}
 - si $\text{corr} = 2^{n-1} - 1$ valeur entière maximale \Rightarrow c'est le plus grand entier négatif possible, il vaut -1
- Ainsi, l'ensemble des entiers relatifs représentés en complément à 2 sur n bits est bien $[-2^{n-1}, 2^{n-1} - 1]$

Détermination de l'opposé d'un nombre

Opposé d'un entier en complément à 2

- L'opposé de N_b représenté en complément à 2 est $\overline{N_b} + 1$
- $\overline{N_b} + 1$ est appelé le *complément à 2* (Cà2) de N
- Il ne faut pas confondre le calcul du complément à 2 d'un nombre avec la représentation en complément à 2

Détermination/preuve

- Pour tout bit a_i , $a_i + \overline{a_i} = 1$
$$\begin{aligned} N_b + \overline{N_b} &= \begin{array}{r} a_{n-1} \dots a_1 a_0 \\ + \overline{a_{n-1} \dots a_1 a_0} \\ \hline 1 \dots 1 1_b \end{array} \\ &= -1_d \end{aligned}$$
- $-N_b = 1 + \overline{N_b}$

Remarques

- L'intervalle de définition des nombres représentables en complément à 2 sur n bits n'est pas symétrique
- On ne peut pas déterminer l'opposé de -2^{n-1} sur n bits.

Détermination de l'opposé d'un nombre

Opposé d'un entier en complément à 2

- L'opposé de N_b représenté en complément à 2 est $\overline{N_b} + 1$
- $\overline{N_b} + 1$ est appelé le *complément à 2* (Cà2) de N
- Il ne faut pas confondre le calcul du complément à 2 d'un nombre avec la représentation en complément à 2

Détermination/preuve

- Pour tout bit a_i , $a_i + \overline{a_i} = 1$

$$\begin{aligned} N_b + \overline{N_b} &= \begin{array}{c} a_{n-1} \dots a_1 a_0 \\ + \overline{a_{n-1} \dots a_1 a_0} \\ \hline 1 \dots 1 \ 1_b \end{array} \\ &= -1_d \end{aligned}$$

- $-N_b = 1 + \overline{N_b}$

Remarques

- L'intervalle de définition des nombres représentables en complément à 2 sur n bits n'est pas symétrique
- On ne peut pas déterminer l'opposé de -2^{n-1} sur n bits.

Détermination de l'opposé d'un nombre

Opposé d'un entier en complément à 2

- L'opposé de N_b représenté en complément à 2 est $\overline{N_b} + 1$
- $\overline{N_b} + 1$ est appelé le *complément à 2* (Cà2) de N
- Il ne faut pas confondre le calcul du complément à 2 d'un nombre avec la représentation en complément à 2

Détermination/preuve

- Pour tout bit a_i , $a_i + \overline{a_i} = 1$
$$\begin{aligned} N_b + \overline{N_b} &= \begin{array}{r} a_{n-1} \dots a_1 a_0 \\ + \overline{a_{n-1} \dots a_1 a_0} \\ \hline 1 \dots 1 1_b \end{array} \\ &= -1_d \end{aligned}$$

- $-N_b = 1 + \overline{N_b}$

Remarques

- L'intervalle de définition des nombres représentables en complément à 2 sur n bits n'est pas symétrique
- On ne peut pas déterminer l'opposé de -2^{n-1} sur n bits.

Détermination de l'opposé d'un nombre

Opposé d'un entier en complément à 2

- L'opposé de N_b représenté en complément à 2 est $\overline{N_b} + 1$
- $\overline{N_b} + 1$ est appelé le *complément à 2* (Cà2) de N
- Il ne faut pas confondre le calcul du complément à 2 d'un nombre avec la représentation en complément à 2

Détermination/preuve

- Pour tout bit a_i , $a_i + \overline{a_i} = 1$
$$\begin{aligned} N_b + \overline{N_b} &= \begin{array}{r} a_{n-1} \dots a_1 a_0 \\ + \overline{a_{n-1} \dots a_1 a_0} \\ \hline 1 \dots 1 1_b \end{array} \\ &= -1_d \end{aligned}$$
- $-N_b = 1 + \overline{N_b}$

Remarques

- L'intervalle de définition des nombres représentables en complément à 2 sur n bits n'est pas symétrique
- On ne peut pas déterminer l'opposé de -2^{n-1} sur n bits.

Extension de la représentation d'un entier relatif

Extension de p à n bits

- Soit un mot de p bits contenant une valeur v , interprété selon le codage entier relatif par le nombre d
- Pour que le mot v' de n bits ($n > p$) encode également le nombre d , il faut :
 - Copier les bits de poids faible de v dans v'
 - Mettre les bits de poids fort restant à la valeur du bit de poids fort de v
- Si $N = a_{p-1}a_{p-2}\dots a_1a_0$ sur p bits, alors sur n bits
 $N = a_{p-1}\dots(n-p \text{ fois})\dots a_{p-1}a_{p-1}a_{p-2}\dots a_1a_0$

Exemple

$$N = 1000_b = 11000_b = 111000_b = 1111000_b$$

Extension de la représentation d'un entier relatif

Preuve : extension de p à $p + 1$ bit

- N_p représentant N codé sur p bits ($a_{p-1}a_{p-2}\dots a_1a_0$) :

$$N_p = -a_{p-1}2^{p-1} + \sum_{i=0}^{p-2} a_i 2^i$$

- N_{p+1} correspondant à l'extension de N_p sur $p + 1$ bit :

$$\begin{aligned} N_{p+1} &= -a_{p-1}2^p + \sum_{i=0}^{p-1} a_i 2^i \\ &= -a_{p-1}2^p + a_{p-1}2^{p-1} + \sum_{i=0}^{p-2} a_i 2^i \end{aligned}$$

- Factorisation des premiers termes par a_{p-1} :

$$N_{p+1} = a_{p-1}(-2^p + 2^{p-1}) + \sum_{i=0}^{p-2} a_i 2^i$$

- Comme $-2^p + 2^{p-1} = -2^{p-1}$ on obtient :

$$\begin{aligned} N_{p+1} &= a_{p-1}(-2^{p-1}) + \sum_{i=0}^{p-2} a_i 2^i \\ &= N_p \quad \square \end{aligned}$$

Exemples d'extension en entier relatif

Extension de 4 à 8 bits

- $N_1 = 0b1001 = 0b11111001$
- $N_2 = 0b0110 = 0b00000110$

Extension de 16 à 32 bits en hexadécimal

- $N_3 = 0x1110 = 0x00001110$
- $N_4 = 0x90B2 = 0xFFFF90B2$

Exemples d'extension en entier relatif

Extension de 4 à 8 bits

- $N_1 = 0b1001 = 0b11111001$
- $N_2 = 0b0110 = 0b00000110$

Extension de 16 à 32 bits en hexadécimal

- $N_3 = 0x1110 = 0x00001110$
- $N_4 = 0x90B2 = 0xFFFF90B2$

Exemples d'extension en entier relatif

Extension de 4 à 8 bits

- $N_1 = 0b1001 = 0b11111001$
- $N_2 = 0b0110 = 0b00000110$

Extension de 16 à 32 bits en hexadécimal

- $N_3 = 0x1110 = 0x00001110$
- $N_4 = 0x90B2 = 0xFFFF90B2$

Exemples d'extension en entier relatif

Extension de 4 à 8 bits

- $N_1 = 0b1001 = 0b11111001$
- $N_2 = 0b0110 = 0b00000110$

Extension de 16 à 32 bits en hexadécimal

- $N_3 = 0x1110 = 0x00001110$
- $N_4 = 0x90B2 = 0xFFFF90B2$

Exemples d'extension en entier relatif

Extension de 4 à 8 bits

- $N_1 = 0b1001 = 0b11111001$
- $N_2 = 0b0110 = 0b00000110$

Extension de 16 à 32 bits en hexadécimal

- $N_3 = 0x1110 = 0x00001110$
- $N_4 = 0x90B2 = 0xFFFF90B2$

Addition et soustraction d'entiers relatifs

Propriété

- L'addition sur les mots binaires peut être utilisée pour sommer des entiers relatifs
 - Cela revient à sommer les parties négatives et à sommer les corrections positives
- Seule la condition de débordement change
- Propriété du codage des relatifs en complément à 2

Addition et soustraction d'entiers relatifs

Soustraction

- On réaliser un circuit soustracteur, mais en réalité pas besoin
- On tire partie du fait que $A - B = A + (-B)$, avec $-B = \text{Cà2}(B)$
- \Rightarrow On calcule $-B$, et on réutilise l'additionneur comme pour l'addition des entiers relatifs

Exemples d'addition et soustraction

Exemples

$$\begin{array}{r} 0 \ 1 \ 1 \ 0 \\ + \ 0 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

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

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

Exemples d'addition et soustraction

Exemples

$$\begin{array}{r} 0 \ 1 \ 1 \ 0 \\ + \ 0 \ 0 \ 1 \ 1 \\ \hline 1 \ 0 \ 0 \ 1 \end{array}$$

naturels : OK
relatifs : KO

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

mais également :

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \\ + \ 0 \ 1 \ 0 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \end{array}$$

relatifs : OK

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

mais également :

$$\begin{array}{r} 0 \ 0 \ 1 \ 0 \\ + \ 0 \ 1 \ 0 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \end{array}$$

relatifs : OK

Dépassement de capacité

Résultat représentable

- Le résultat d'une addition/soustraction de 2 mots de n bits est dit **représentable** sur n bits si l'interprétation des bits $n - 1$ à 0 dans la même représentation représente bien le résultat théorique de l'opération.
- Si ce n'est pas le cas : **dépassement de capacité**
- **Remarque** : le fait qu'il y ait un dépassement ou non dépend de l'interprétation des mots

Addition de A et B et dépassement de capacité en entiers relatifs

Il y a 4 cas à analyser en fonction du signe de A et B

- $A < 0$ et $B > 0$: signes différents, A négatif
- $A > 0$ et $B < 0$: signes différents, B négatif
- $A > 0$ et $B > 0$: même signe et positifs
- $A < 0$ et $B < 0$: même signe et négatifs

Détermination d'un dépassement de capacité

A et B de signe différent, A négatif

On a

- $-2^{n-1} \leq A \leq -1$
- $0 \leq B \leq 2^{n-1} - 1$
- $-2^{n-1} \leq A + B \leq 2^{n-1} - 2$

Le résultat est toujours représentable, il n'y a pas de dépassement de capacité

$$\begin{array}{rcl} & 1a_{n-2} \dots a_1 a_0 \\ + & 0b_{n-2} \dots b_1 b_0 \\ = & s_{n-1} s_{n-2} \dots s_1 s_0 \end{array}$$

- Si l'addition des corrections positives \geq à 2^{n-1} on a $c_{out_{n-2}} = 1$ et donc forcément $c_{out_{n-1}} = 1$
- Si l'addition des corrections positives est $<$ à 2^{n-1} on a $c_{out_{n-2}} = 0$ et donc forcément $c_{out_{n-1}} = 0$
- On a toujours : résultat correct et $c_{out_{n-1}} = c_{out_{n-2}}$

Détermination d'un dépassement de capacité

A et B de même signe et négatif

$$\begin{array}{r} 1a_{n-2}\dots a_1a_0 \\ + \quad 1b_{n-2}\dots b_1b_0 \\ = \quad s_{n-1}s_{n-2}\dots s_1s_0 \end{array}$$

- On a toujours $c_{out_{n-1}} = 1$
- Si l'addition des corrections positives est $< 2^{n-1}$ on a $c_{out_{n-2}} = 0$ et le résultat est positif, et donc incorrect
- Si l'addition des corrections positives est $\geq 2^{n-1}$ on a $c_{out_{n-2}} = 1$, le résultat est négatif ($s_{n-1} = 1$), et donc correct
- On a donc résultat correct et $c_{out_{n-1}} = c_{out_{n-2}}$ ou alors résultat incorrect et $c_{out_{n-1}} \neq c_{out_{n-2}}$

Détermination d'un dépassement de capacité

A et B de même signe et positif

$$\begin{array}{r} 0a_{n-2} \dots a_1 a_0 \\ + \quad 0b_{n-2} \dots b_1 b_0 \\ = \quad s_{n-1} s_{n-2} \dots s_1 s_0 \end{array}$$

- On a toujours $c_{out_{n-1}} = 0$
- Si l'addition des corrections positives est $< 2^{n-1}$ on a $c_{out_{n-2}} = 0$, le résultat est positif et correct : il reste dans l'intervalle de représentation des valeurs positives
- Si l'addition des corrections positives est $\geq 2^{n-1}$ on a $c_{out_{n-2}} = 1$, le résultat n'est pas dans l'intervalle de représentation des valeurs positives, le résultat sera négatif ($s_{n-1} = 1$), et donc incorrect
- On a donc résultat correct et $c_{out_{n-1}} = c_{out_{n-2}}$ ou alors résultat incorrect et $c_{out_{n-1}} \neq c_{out_{n-2}}$

Détermination d'un dépassement de capacité lors d'une addition

Synthèse de l'analyse des différents cas

Le résultat d'une addition sur entiers relatifs est correct ssi $c_{out_{n-1}} = c_{out_{n-2}}$

Détection de dépassement

Il y a dépassement de capacité sur entiers relatifs ssi $c_{out_{n-1}} \neq c_{out_{n-2}}$.

Attention

La détection d'un dépassement de capacité lors de l'addition sur entiers relatifs est différente de celle lors de l'addition d'entiers naturels.

Détermination d'un dépassement de capacité lors d'une addition

Synthèse de l'analyse des différents cas

Le résultat d'une addition sur entiers relatifs est correct ssi $c_{out_{n-1}} = c_{out_{n-2}}$

Détection de dépassement

Il y a dépassement de capacité sur entiers relatifs ssi $c_{out_{n-1}} \neq c_{out_{n-2}}$.

Attention

La détection d'un dépassement de capacité lors de l'addition sur entiers relatifs est différente de celle lors de l'addition d'entiers naturels.

Détermination d'un dépassement de capacité lors d'une addition

Synthèse de l'analyse des différents cas

Le résultat d'une addition sur entiers relatifs est correct ssi $c_{out_{n-1}} = c_{out_{n-2}}$

Détection de dépassement

Il y a dépassement de capacité sur entiers relatifs ssi $c_{out_{n-1}} \neq c_{out_{n-2}}$.

Attention

La détection d'un dépassement de capacité lors de l'addition sur entiers relatifs est différente de celle lors de l'addition d'entiers naturels.

Exemples d'addition et illustration des différents cas

Exemples

$$\begin{array}{r} 0 \ 1 \ 1 \ 0 \\ + \ 0 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \\ + \ 0 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \\ + \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \\ + \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 0 \ 1 \ 1 \ 0 \\ + \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 0 \\ + \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

Exemples d'addition avec les différents cas

Exemples

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Résultat négatif !

$$C_{out_{n-1}} \neq C_{out_{n-2}}$$

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Résultat positif !

$$C_{out_{n-1}} = C_{out_{n-2}}$$

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Résultat négatif !

$$C_{out_{n-1}} = C_{out_{n-2}}$$

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Résultat positif !

$$C_{out_{n-1}} \neq C_{out_{n-2}}$$

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Addition de nombres de signes différents

$$C_{out_{n-1}} = C_{out_{n-2}}$$

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

Addition de nombres de signes différents

$$C_{out_{n-1}} = C_{out_{n-2}}$$

Soustractions d'entiers non signés

Soustraction d'entiers non signés

- A priori, pour faire une soustraction d'entiers non signés sur n bits, si on ne veut pas rajouter un soustracteur^a, il faut un additionneur $n + 1$ bits :
 - Sur $n + 1$ bits en complément à 2, on peut représenter les valeurs de -2^n à $2^n - 1$
 - \Rightarrow inclut l'intervalle $[0; 2^n - 1]$ des entiers naturels sur n bits (avec le bit de signe = 0)
 - On peut ainsi calculer l'opposé des nombres et faire l'addition $A + (-B) = A - B$
- ...ou pas : exemple avec $15 - 10$

a. On peut réaliser un circuit réalisant la soustraction de mot binaire de n bits à partir d'un soustracteur 1 bit – on réalise alors la soustraction comme à l'école mais pour la base 2 et non 10

Conclusion

- Pour faire une soustraction d'entiers non signés sur n bits, on peut utiliser le même additionneur/soustracteur n bits que pour les signés en faisant le calcul $A - B = A + C_2(B)$
- La condition de débordement n'est pas la même que pour les relatifs...^a

a. Question : comment détecter un dépassement de capacité lors d'une soustraction sur 2 entiers non signés ?

- 1 Rappels
- 2 Arithmétique sur entiers naturels
- 3 Représentation des entiers relatifs et arithmétique entière
- 4 Représentation des caractères alphanumériques
 - Codage ASCII
 - Codage de chaînes

Représentation des caractères alphanumériques

Utilité

Les caractères alphanumériques sont utilisés pour les mots en langage naturel/écrits par utilisateur (texte, programme, commande, email,...)

Codage des caractères

- Les caractères sur un clavier sont désignés par leur position
- Une table de codage associe à chaque position (éventuellement en combinaison avec d'autres touches) un caractère
- Ce caractère lorsqu'il est rangé en mémoire ou lorsqu'il est manipulé par un programme est codé suivant un codage
- Il existe plusieurs codages différents (UTF8, ASCII, etc.)

Codage ASCII

Dans ce codage, un caractère est codé sur 1 octet, une table donne le codage Seul 7 des 8 bits sont utilisés, le bit de poids fort vaut toujours 0

Hex	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	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
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



Quartet de poids fort

Extension du codage ASCII

Extension du codage ASCII

- Des extensions de codage incorporant le 8ème bit de poids fort permettent d'avoir un sous ensemble commun (caractères codés de 00_h à $7F_h$) puis des caractères spécifiques à une langue (codé de 80_h à FF_h)
- L'extension pour les caractères français (accents, cédille) est latin-1 (ISO 8859-1)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8																
9																
A		;	ç	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Extension iso-latin

Codage de chaînes

Codage d'une chaîne de caractères

- Une chaîne de caractère est composée de plusieurs lettres et symboles
- Codage d'une chaîne = codage de chacune des lettres et symboles la composant + caractère de fin de chaîne : `'\0'` (code 0x00)

Codage de chaînes en ASCII

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0').
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0').

Attention

$$123_{10} = 64 + 32 + 16 + 8 + 2 + 1$$

Donc sur 8 bits, $123_{10} = 0b01111011 = 0x7B$, et sur 32 bits, $123_{10} = 0x0000007B$

$$123_{10} \neq \text{"123"}$$

Codage de chaînes

Codage d'une chaîne de caractères

- Une chaîne de caractère est composée de plusieurs lettres et symboles
- Codage d'une chaîne = codage de chacune des lettres et symboles la composant + caractère de fin de chaîne : `'\0'` (code 0x00)

Codage de chaînes en ASCII

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0').
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0').

Attention

$$123_d = 64 + 32 + 16 + 8 + 2 + 1$$

Donc sur 8 bits, $123_d = 0b01111011 = 0x7B$, et sur 32 bits, $123_d = 0x0000007B$

$$123_d \neq \text{"123"}$$

Codage de chaînes

Codage d'une chaîne de caractères

- Une chaîne de caractère est composée de plusieurs lettres et symboles
- Codage d'une chaîne = codage de chacune des lettres et symboles la composant + caractère de fin de chaîne : `'\0'` (code 0x00)

Codage de chaînes en ASCII

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0').
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0').

Attention

$$123_d = 64 + 32 + 16 + 8 + 2 + 1$$

Donc sur 8 bits, $123_d = 0b01111011 = 0x7B$, et sur 32 bits, $123_d = 0x0000007B$

$123_d \neq \text{"123"}$

Conclusion

On a vu :

- addition (et soustraction) sur entiers naturels
- addition sur des mots binaires (taille bornée), dépassement de capacité
- représentation des entiers relatifs en complément à 2
- addition et soustraction sur entiers relatifs, dépassement de capacité
- représentation des caractères alphanumériques, codage ASCII

Nota bene :

- Tous les supports + infos sont sur Moodle