ARCHI – Architecture des ordinateurs

Sylvain Brandel
2025 – 2026
sylvain.brandel@univ-lyon1.fr

Partie 3

CODAGE DES DONNÉES EN MACHINE

Codage des entiers

Codage des nombres rationnels

Information

- (Transistors : plus tard)
- Détection de deux états
  - Haut ≥ réf. haute
  - Bas ≤ réf. basse
- Physique : différence de 1V (ordre de grandeur)

- Convention
  - L'un : 0
  - L'autre : 1

- Représentation binaire
  - bit
  - Par mots de 4 bits : représentation hexadécimale
  - Mot de 8 bits : octet (Byte)

    → Mb : Mega bit MB : Mega Byte

Codage des entiers naturels
Notation positionnelle

- $\beta \in \mathbb{N}$, $\beta > 1$: base
- Représentation positionnelle en base β de n ∈ $\mathbb{N}$:

$$(x_{p-1}x_{p-2} \dots x_1x_0)_\beta := \sum_{i=0}^{p-1} x_i\beta^i$$

- $x_i \in \{0, 1, \dots, \beta - 1\}$: chiffres de l'écriture de *n* en base β
  - β = 2 : chiffres 0 et 1
  - β = 10 : chiffres de 0 à 9
  - β = 16 : chiffres de 0 à F
- *p* : nombre de chiffres nécessaires pour écrire n

- Ex : $(5134)_{10} = 5.10^3 + 1.10^2 + 3.10^1 + 4.10^0$

Codage des entiers naturels
Changement de base

- Avec notation positionnelle
- $\beta \in \mathbb{N}$, $\beta > 1$: base de départ, $\gamma \in \mathbb{N}$, $\gamma > 1$: base d'arrivée
- Toujours possible :
  - Conversion x_i et β vers écriture en base γ
  - Calcul de $\sum_{i=0}^{p-1} x_i \beta^i$ avec opérations en base γ
- Ecriture de *n* en base γ : calcul dans la base d'arrivée γ

- Ex : Conversion binaire vers décimal de n = $(10100)_2$:
  - Ajout des puissances de 2 correspondant aux bits non nuls

Chiffre	1	0	1	0	0
Position	4	3	2	1	0
Poids	2^4	0	2^2	0	0

  - Donc $(10100)_2 = 2^4 + 2^2 = 20$
- Ex : Conversion décimal vers binaire de n = $(95)_{10}$ et $(423)_{10}$

Codage des entiers naturels
Changement de base

- Avec divisions euclidiennes successives
- Reste de la division euclidienne de n par β :
  - Chiffre de poids faible dans l'écriture de n en base β

$$n = x_{p-1}\beta^{p-1} + x_{p-2}\beta^{p-2} + \ldots + x_2\beta^2 + x_1\beta^1 + x_0$$
$$= \underbrace{(x_{p-1}\beta^{p-2} + x_{p-2}\beta^{p-3} + \ldots + x_2\beta^1 + x_1)\beta}_{\text{quotient}} + \underbrace{x_0}_{\text{reste}}$$

avec $0 \leq x_0 \leq \beta$

- En d'autres termes, n mod β = x_0
  - Les chiffres de n sont obtenus par divisions euclidiennes successives
  - Arrêt au premier quotient nul. Chiffres de poids faible d'abord !
- Ex : Conversion décimal vers binaire de $(95)_{10}$ et $(423)_{10}$
- Ex : Conversion décimal vers octal de $(3452)_{10}$

Codage des entiers naturels

Changement de base

- Entre bases 2, 8, 16 : conversions directes
- $8 = 2^3$ → chiffre octal : entier sur trois bits

- $(x_8x_7x_6x_5x_4x_3x_2x_1x_0)_2 = \underbrace{(x_8x_7x_6)_2}_{y_2}8^2 + \underbrace{(x_5x_4x_3)_2}_{y_1}8^1 + \underbrace{(x_2x_1x_0)_2}_{y_0}8^0 = (y_2y_1y_0)_8$

- Ex : Conversion octal vers binaire de $(34521)_8$
- Ex : Conversion hexadécimal vers binaire de $(9A6E)_{16}$

Entiers naturels
Représentation *machine*

- Nombre de bits p fixé pour chaque format de codage (8, 16, 32, 64)
- Si résultat sur plus de p bits :
  - Obtenu : p bits de poids faible du résultat exact
  - Drapeau de dépassement de capacité de l'UAL
- Les calculs continuent !
- m codé sur q ≥ p bits

$$m = \sum_{i=0}^{q-1} m_i 2^i = \underbrace{\left(\sum_{i=p}^{q-1} m_i\right) 2^p}_{\text{quotient de m par } 2^p} + \underbrace{\sum_{i=0}^{p-1} m_i 2^i}_{\text{reste}}$$

- Opération arithmétique sur entiers naturels
  - Résultat m placé sur p bits
  - Du coup résultat obtenu : m mod 2^p

Codage des entiers relatifs

- Mots de n bits → différents états w = $b_{n-1} \dots b_2 b_1 b_0$

- Non signés :
  - Notation positionnelle : $[\![w]\!] = \sum_{i=0}^{n-1} b_i 2^i$

- Signés :
  - Signe + valeur absolue : $[\![w]\!] = (-1)b^{n-1} \sum_{i=0}^{n-2} b_i 2^i$
  - Complément à 1 : $[\![v]\!] = -[\![w]\!] = \overline{b_{n-1}} \dots \overline{b_2}\ \overline{b_1}\ \overline{b_0}$
  - Complément à 2 : $[\![w]\!] = -b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$
  - Biais N : $[\![w]\!] = \sum_{i=0}^{n-1} b_i 2^i - N$

Codage des entiers relatifs
Complément à 2

- n entier relatif, $-2^{p-1} \leq n \leq 2^{p-1} - 1$, à coder sur p bits
- Notation en complément à 2 sur p bits

$$n = \left(c_{p-1}c_{p-2} \ \ldots \ c_1c_0\right)_{\overline{2}}$$

$$\left(c_{p-1}c_{p-2} \ \ldots \ c_1c_0\right)_{\overline{2}} := -c_{p-1}2^{p-1} + \sum_{i=0}^{p-2} c_i 2^i$$

- Propriétés :
  - $n \geq 0$ ssi $c_{p-1} = 0$
  - $n \leq 0$ ssi $c_{p-1} = 1$

Codage des entiers relatifs
Complément à 2

- Interprétation
  - Considérer le bit le plus à gauche de poids négatif (-2^{p-1})

- Ex : p = 8, valeur décimale codée par $(10000011)_{\overline{2}}$

$$(10000011)_{\overline{2}} = -128 + 3 = (-125)_{10}$$

- Ex : p = 8, coder $(-120)_{10}$ en complément à 2

$$(-120)_{10} = -128 + 8 = (10001000)_{\overline{2}}$$

Entiers relatifs
Complément à 2

- $m = (c_m)_{\overline{2}}$ et $n = (c_n)_{\overline{2}}$ en complément à 2 sur p bits

- Addition
  - Sans dépassement : codage de m + n = codage de $(c_m + c_n) \bmod 2^p$ en tant qu'entier naturel
  - Avec dépassement : résultat faux
  - Dépassement
    - m et n de signes opposés : dépassement impossible
    - m et n de même signe : dépassement ssi signe résultat ≠ signe de n

- Opposé
  - Sans dépassement : codage de –n en complément à 2 sur p bits

$$(\overline{c_{p-1}} \ \ldots \ \overline{c_1} \ \overline{c_0})_2 + 1 \ mod \ 2^p$$

comme entier naturel

- Ex : p = 7, $(36)_{10} = (0100100)_{\overline{2}}$ $(-36)_{10} = (1011100)_{\overline{2}}$

Rationnels

- Nombre de la forme $\frac{p}{q}$ avec p ∈ $\mathbb{Z}$ et q ∈ $\mathbb{N}$ − {0}
- Format de longueur fixe là où écriture binaire potentiellement infinie

  → Approximation
- Tout x ∈ $\mathbb{Q}$ positif décomposé en
  - Partie entière $\lfloor x \rfloor \in \mathbb{N}$ telle que $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$
  - Partie fractionnaire $\{x\} = x - \lfloor x \rfloor$ avec $0 \leq \{x\} < 1$

- Notation positionnelle pour l'écriture de $\{x\}$: s'il existe $q \in \mathbb{N}$ tq

$$\{x\} = (0, x_{-1} \dots x_{-q})_\beta = \sum_{i=0}^{q} x_{-i}\beta^{-i}$$

- Alors $(x_{p-1} \dots x_1 x_0 , x_{-1} \dots x_{-q})_\beta$ écriture de x en base β
- Écriture en base β pas forcément finie mais forcément périodique
- Ex : β = 10, écriture de 13/7 en notation partie entière - fractionnaire

13/7 = $(1,\underline{857142})_{10}$

Rationnels
Changement de base

- 0 ≤ x < 1 écrit en base 10
- Décimal vers binaire

$$x = (0, x_{-1} \dots x_{-q})_2$$

  - Or $2 \times x = (x_{-1}, x_{-2} \dots x_{-q})_2$, donc $x_{-1} = \lfloor 2 \times x \rfloor$

- Multiplications successives par 2

  → extraction bits écriture binaire de x

- Ex : Convertir 1/10 = $(0,1)_{10}$ en écriture binaire

$(0,1)_{10} = (0,0\underline{0011})_2$

Rationnels
Changement de base

- $0 \leq x < 1$ écrit en base 2
- Binaire vers décimal

- Multiplications successives par $(10)_{10} = (1010)_2$

  → en calculant en binaire : chiffres décimaux de x

- Fastidieux à la main
- Si pas trop de bits, il suffit d'additionner les poids de ces bits :
  - $2^{-1} = 0,5$ $2^{-2} = 0,25$ $2^{-3} = 0,125$ $2^{-4} = 0,0625$

- Ex : Convertir $(0,1011)_2$ en écriture décimale

$$0,5 + 0,125 + 0,0625 = 0,6875$$

Rationnels
Représentation en machine ?

- On n'a pas réellement les rationnels

- Un nombre flottant normalisé x est
  - Soit zéro
  - Soit un rationnel de la forme $x = (-1)^s \times \underbrace{(1, b_1 \ \ldots \ b_{p-1})_2}_{p \text{ bits de précision } (b_0 = 1)} \times 2^e$
    - s ∈ {0,1} : signe
    - $(1, b_1 \ldots b_{p-1})_2$: mantisse fractionnaire
    - e ∈ $\mathbb{Z}$: exposant tel que $e_{min} \leq e \leq e_{max}$

- On parle de nombre à virgule flottante, ou nombre flottant, ou flottant

- Le 1, en tête garantit l'unicité de la représentation

- Types `float` et `double` en C

Nombres à virgule flottante

- On ne les a pas tous !
- P. ex $(0,1)_{10}$ pas dans l'exemple suivant …
- Ex : flottants ≥ 0

  avec 3 bits de précision,

  $e_{min} = -1$, $e_{max} = 2$

  $-1 \leq e \leq 2$

e	Binaire	Décimal
-	0	0
e = -1	$(1,00)_2 \times 2^{-1}$ $(1,01)_2 \times 2^{-1}$ $(1,10)_2 \times 2^{-1}$ $(1,11)_2 \times 2^{-1}$	0,5 0,625 0,75 0,875
e = 0	$(1,00)_2 \times 2^{0}$ $(1,01)_2 \times 2^{0}$ $(1,10)_2 \times 2^{0}$ $(1,11)_2 \times 2^{0}$	1 1,25 1,5 1,75
e = 1	$(1,00)_2 \times 2^{1}$ $(1,01)_2 \times 2^{1}$ $(1,10)_2 \times 2^{1}$ $(1,11)_2 \times 2^{1}$	2 2,5 3 3,5
e = 2	$(1,00)_2 \times 2^{2}$ $(1,01)_2 \times 2^{2}$ $(1,10)_2 \times 2^{2}$ $(1,11)_2 \times 2^{2}$	4 5 6 7

Nombres à virgule flottante

- Ex : $(0,1)_{10}$ avec 8 bits de précision

- $(0,1)_{10} = (0,0\underline{0011})_2$
  $= (0,000110011001100\underline{0011})_2$
  $= (1,10011001\underline{0011})_2 \times 2^{-4}$

  8 bits de précision

- On doit tronquer à 7 bits de mantisse

  Donc $(1,1001100)_2 \times 2^{-4} \leq (0,1)_{10} \leq (1,1001101)_2 \times 2^{-4}$

- Il faut faire un choix → arrondi
  - Le milieu de cet intervalle est $(1,10011001)_2 \times 2^{-4}$
  - Ici on choisit d'arrondir au plus proche
  - $(0,1)_{10}$ est supérieur au milieu de l'intervalle, donc

L'approximation $(0,1)_{10} \approx (1,1001101)_2 \times 2^{-4}$

Nombres à virgule flottante – norme IEEE-754

- Codage en précision p :

c =

1 bit	k bits	p-1 bits
Signe	Code de l'exposant : c_e	Code de la mantisse : c_m

- Pour un flottant normal, la valeur codée est

$$f(c) = (-1)^{s(c)} \times m(c) \times 2^{e(c)}$$

  - s(c) : signe du flottant
  - $m(c) = (1,c_m)_2$ (le 1, est implicite)
  - $e(c) = (c_e)_2 - (2^{k-1} - 1)$

Nombres à virgule flottante – norme IEEE-754

- Codage en précision p :

c =

1 bit	k bits	p-1 bits
Signe	Code de l'exposant : c_e	Code de la mantisse : c_m

- $(c_e)_2 = 0$ et $(c_m)_2 = 0$: nombre représenté est zéro (2 codages)
- $(c_e)_2 = 2^k - 1$: valeur exceptionnelle (+Inf, -Inf, NaN)
- $1 \leq (c_e)_2 \leq 2^k - 2$: nombre représenté normal, alors

$$m(c) = (1,c_m)_2 \quad \text{et} \quad e(c) = (c_e)_2 - \underbrace{(2^{k-1} - 1)}_{\text{biais}}$$

Nombres à virgule flottante – norme IEEE-754

- Simple précision
  - codé sur 32 bits
  - type `float` en C)

31	30 … 23	22 … 0
Signe	c_e	c_m

  - Précision p = 24 bits, k = 8, biais = 127, e_{min} = -126, e_{max} = 127
  - $[\![w]\!] = (-1)^{b_{31}} \times (1, b_{22} \ \ldots \ b_1 b_0)_2 \times 2^{(\sum_{i=23}^{30} b_i 2^{(i-23)-127})}$
  - Valeurs représentables$[\ \pm 2^{-126}, (2 \ - 2^{-23}) \times 2^{127}\ ]$

- Double précision
  - codé sur 64 bits
  - type `double` en C

63	62 … 52	51 … 0
Signe	c_e	c_m

  - Précision p = 53 bits, k = 11, biais = 1023, e_{min} = -1022, e_{max} = 1023
  - $[\![w]\!] = (-1)^{b_{63}} \times (1, b_{51} \ \ldots \ b_1 b_0)_2 \times 2^{(\sum_{i=52}^{62} b_i 2^{(i-52)-1023})}$
  - Valeurs représentables$[\ \pm 2^{-1022}, (2 \ - 2^{-52}) \times 2^{1023}\ ]$

Nombres à virgule flottante – norme IEEE-754

- Rationnel ou réel : représentation arrondie en général
- 4 modes d'arrondis
  - Arrondi au plus proche : RN(c)
    - Si c équidistant de deux flottants consécutifs, on prend celui dont la mantisse termine par 0
  - Arrondi vers $-\infty$ (RD), $+\infty$ (RU), 0 (RZ)

- La norme impose l'arrondi correct pour $+$, $-$, $\times$, $\div$, $\sqrt{}$:
  - Résultat calculé = résultat exact arrondi selon le mode d'arrondi courant
  - Mode d'arrondi par défaut : arrondi au plus proche en général

Les float NE SONT PAS les réels