6. Codage des nombres réels

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

1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. Codage des nombres entiers naturels
5. Codage des nombres entiers relatifs
6. **Codage des nombres réels**
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
11. Circuits combinatoires importants
12. Principes de logique séquentielle
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. Réalisation des composants
16. Code machine et langage assembleur
17. Réalisation d'un processeur
18. Performance et micro-architecture
19. Du processeur au système

Les nombres réels: Introduction (base 10)

- Ecriture des nombres réels par une notation en virgule:

  $103.795 = 1*10^{2} + 0*10^{1} + 3*10^{0} + 7*10^{-1} + 9*10^{-2} + 5*10^{-3}$

- Le même nombre peut être écrit en notation scientifique:

  $103.795 = 1.03795 * 10^{2}$

Les nombres réels: Introduction (autres bases)

- Toutes les observations faites en base 10 restent valables en base 2

  $10010.0011_{(2)} = 2^4 + 2^1 + 2^{-3} + 2^{-4}$

- De même, en notation mantisse-exposant:

  $10010.0011_{(2)} = 1.00100011_{(2)} * 2^4 = 1.00100011_{(2)} * 2^{100(2)}$

- En n'importe quelle base:

$$N = (-1)^S * M * B^E$$

Codage de nombres réels

Représentation en virgule flottante.

- Notation mantisse-exposant en base 2.
- On code séparément signe, mantisse et exposant.
- Le terme "virgule flottante" vient du fait qu'on peut déplacer la virgule en changeant la valeur de l'exposant.
- **Le standard IEEE 754**, représente le signe S, l’exposant biaisé EB et la partie fractionnaire F de la mantisse.

Le standard IEEE 754

Un nombre réel général, en base binaire

$0.00101_{(2)} * 2^{0}$

est récrit comme sous forme normalisée.

$1.01_{(2)} * 2^{-3}$

IEEE 754: Codage de la mantisse

Exemple: $1.01_{(2)} * 2^{-3}$

La mantisse M a toujours la forme
M=1.XXXXXX

où "XXXXX" représente la partie fractionnaire F.
On ne représente pas le «1»!

Dans notre exemple, F = 01

IEEE 754: Codage de l'exposant

Exemple: $1.01_{(2)} * 2^{-3}$

L'exposant E peut être positif (pour des nombres > 1 en norme) ou négatif (pour des nombres < 1 en norme)

Notre exemple: E = -3

- Une idée: coder E par la règle du complément à 2.
- Le standard IEEE 754 prend une voie différente: codage par nombre biaisé.
- On code l'exposant biaisé EB qui est un nombre positif.
  Pour obtenir E, il faut soustraire le biais:
  E = EB - biais

La norme IEEE 754

- Codage en **simple precision:** 4 octets (32 bits). 8 bits pour l'exposant biaisé, avec un biais de 127.
- Codage en **double précision:** 8 octets (64 bits). 11 bits pour l'exposant biaisé, avec un biais de 1023.

Précision	Mémoire nécessitée pour un nombre	Signe	Exposant biaisé	Partie fractionnaire de la mantisse	Valeur	Chiffres décimaux signific.
Simple	32 bits	1 bit	8 bits	23 bits	$N = (-1)^S * M * 2^{EB-127}$	Env. 7
Double	64 bits	1 bit	11 bits	52 bits	$N = (-1)^S * M * 2^{EB-1023}$	Env. 16

Exemple: simple précision

Représentons le nombre -5.5 en virgule flottante, simple précision:

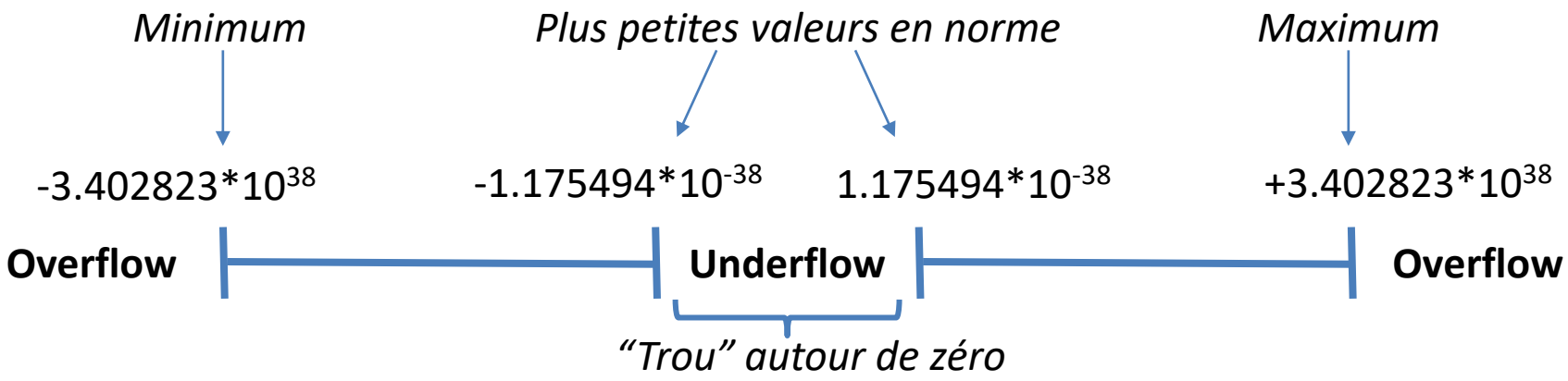
$$N = (-1)^S * M * 2^{EB-127}$$

Limitations de l’exposant

- L’exposant peut en principe adopter des valeurs entre -127 et +128 en simple précision, et entre -1023 et +1024 en double précision. En pratique, IEEE 754 n’utilise que les exposants -126 et +127 en simple précision, et entre -1022 et +1023 en double précision.
- Si l’exposant passe en-dessous de sa valeur minimale, le résultat ne peut pas être codé par la représentation en virgule flottante. On parle alors d’une situation de **underflow**.
- Si l’exposant dépasse sa valeur maximale, le résultat ne peut pas être codé non plus. On parle alors d’un **overflow.**

Débordements et nombres spéciaux

Exemple en simple précision:



IEEE 754: Limitation de la mantisse

- La mantisse elle aussi est codée sur un nombre de bits fixe.
- Problème: précision limitée (nombre de chiffres après la virgule).
- Problématique pour les nombres irrationnels (qui nécessiteraient une infinité de nombres après la virgule)
- Problématique pour tout autre nombre qui requiert une précision plus élevée.

IEEE 754: Limitation de la mantisse

Pour quels nombres l'expansion fractionnaire est-elle limitée ? Pour quels nombres illimités ?

Nombre irrationnels: infinie.

Nombres rationnels: ça dépend.
(c'est différent pour la base décimale et la base binaire)

IEEE 754: Limitation de la mantisse

- En règle générale, tout nombre rationnel s'exprime comme une fraction sous forme numérateur/dénominateur.
- Un tel nombre possède une série fractionnaire finie en base 2 si et seulement si le dénominateur est une puissance de 2.

Exemple: 5.5 peut être représenté exactement (voir exemple précédent), car 5.5 = 11/2: le dénominateur vaut 2.

Exemple: le nombre 0.1 ne peut pas être représenté exactement car 0.1=1/10, et le dénominateur vaut 10=2*5.

Exemple: codage de 0.1 en simple précision

- Exemple: 0.1 en virgule flottante.

  E=-4, M=1.100 1100 1100 1100…

- En simple précision, on a 24 chiffres pour la mantisse:

  M=1.100 1100 1100 1100 1100 1101

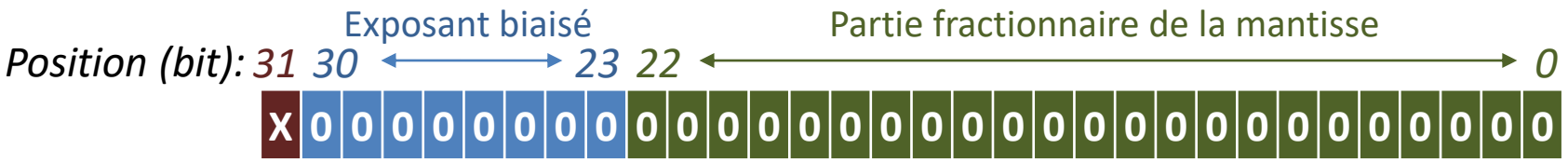
  En base décimale, cela vaut

  $2^{-4}(2^{0}+2^{-1}+2^{-4}+2^{-5}+\ldots) = 0.100000001490116119384765625$

  exactement.

Codage du zéro

- Avec les règles énoncées précédemment, on ne peut pas représenter le zéro exactement. En effet, à cause du bit implicite de la mantisse, la mantisse vaut 1, même si on met à zéro tous les bits dans le nombre réel.
- Pour remédier à ce problème, on *définit* le nombre en virgule flottante dont tous les bits sont nuls comme étant le zéro.
- Plus précisément, on a deux codages possibles pour le zéro, en fonction de la valeur du signe: le zéro positif, et le zéro négatif.



Pour conclure les chapitres sur le codage de nombres:

Comparaison entre
- Les codages de nombres
- Les types de données en programmation

Types de donnée dans le langage C

Les variables et leurs types de donnée sont une abstraction offerte par les langages de programmation pour accéder aux différents codages de l'information. Nous listons ci-dessous certains types de donnée du langage de programmation C.

Nom	Alias(*)	Type	Taille (octets)
int8_t	**char**	$\mathbb{Z}_{(8)}$	1
uint8_t	**unsigned char**	$\mathbb{N}_{(8)}$	1
int16_t	**short**	$\mathbb{Z}_{(16)}$	2
uint16_t	**unsigned short**	$\mathbb{N}_{(16)}$	2
int32_t	**int** ou **long**	$\mathbb{Z}_{(32)}$	4
uint32_t	**unsigned int** ou **unsigned long**	$\mathbb{N}_{(32)}$	4
int64_t	**long** ou **long long**	$\mathbb{Z}_{(64)}$	8
uint64_t	**unsigned long** ou **unsigned long long**	$\mathbb{N}_{(64)}$	8
	float	IEEE 754 single	4
	double	IEEE 754 double	8

Les alias sont d'autres noms, communément utilisés dans le langage C. Leur signification exacte dépend souvent du processeur et du système d'exploitation. Par exemple, le type **long** utilise 4 octets sur la plupart des processeurs 32-bit, 8 octets sur les processeurs Intel 64-bit sous Linux ou Mac OS/X, mais 4 octets sur les processeurs Intel 64-bit sous Windows.

Jonas Lätt