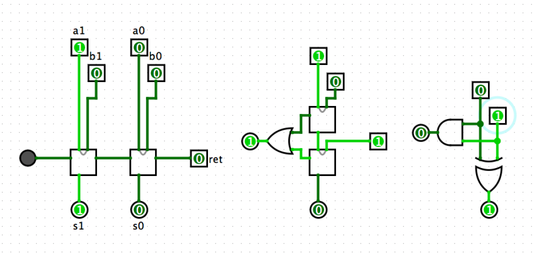
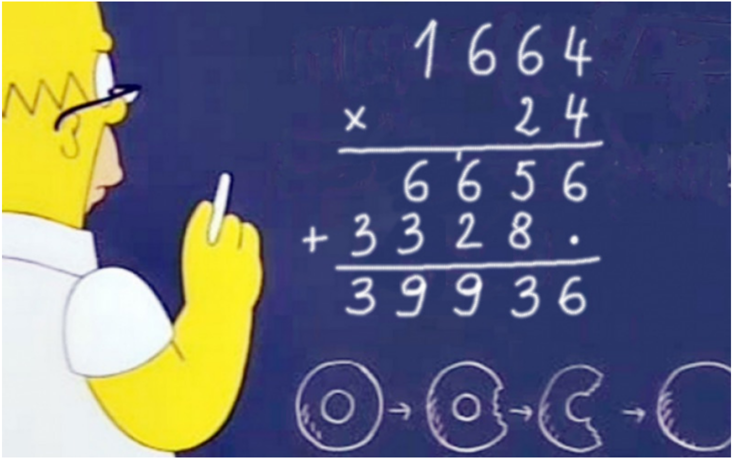
Principes de fonctionnement des machines binaires

2022–2023

Matthieu Picantin

- numération et arithmétique
- numération et arithmétique en machine
- numérisation et codage (texte, images)
- compression, cryptographie, contrôle derreur
- logique et calcul propositionnel
- circuits numériques




1664
× 24
6656
+ 3328 .
39936

Addition posée

- on s'appuie sur la *table d'addition* en base b
- on dispose les nombres en colonnes: chiffres des unités, chiffres des b-aines, chiffres des b^2-aines, etc
- on effectue la somme des chiffres de la colonne la plus à droite:
  on *pose* son chiffre des unités
  on *reporte* la retenue sur la ou les colonnes à gauche
- on recommence sur la colonne immédiatement à gauche, etc

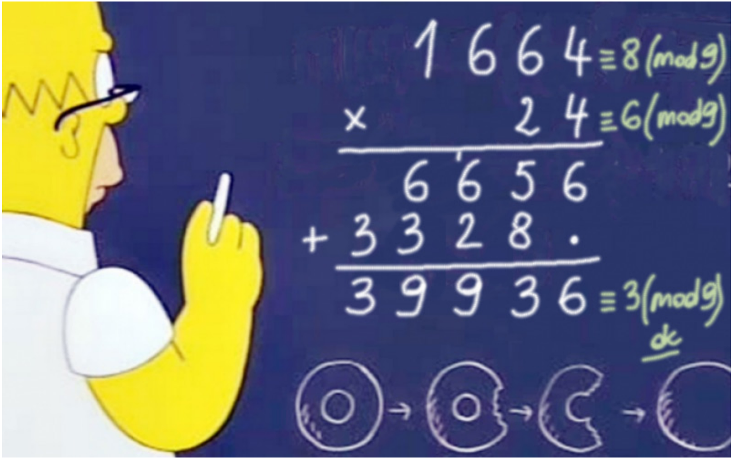
Multiplication posée

- on s'appuie sur les *tables d'addition et de multiplication* en base b
- on dispose les deux nombres $(a_p \cdots a_0)_b$ et $(c_q \cdots c_0)_b$ en colonnes: chiffres des unités, chiffres des b-aines, chiffres des b^2-aines, etc
- on effectue le produit de $(a_p \cdots a_0)_b$ par le nombre $c_k b^k$ pour $0 \leq k \leq q$
- on conclut avec l'addition posée de ces $q + 1$ produits intermédiaires

La "preuve" par 9 pour un calcul en base 10

- n'est pas une *preuve* de correction, mais un moyen simple de vérification (avec un certain indice de confiance)
- utilise les propriétés de l'arithmétique modulaire
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* 9 (obtenus en faisant la somme de leurs chiffres, etc)

$$\sum_{k=0}^{p} a_k 10^k \equiv \sum_{k=0}^{p} a_k \pmod 9$$


1 6 6 4 ≡ 8 (mod 9)
× 2 4 ≡ 6 (mod 9)
6 6 5 6
+ 3 3 2 8 .
3 9 9 3 6 ≡ 3 (mod 9)
ok

La "preuve" par 11 pour un calcul en base 10

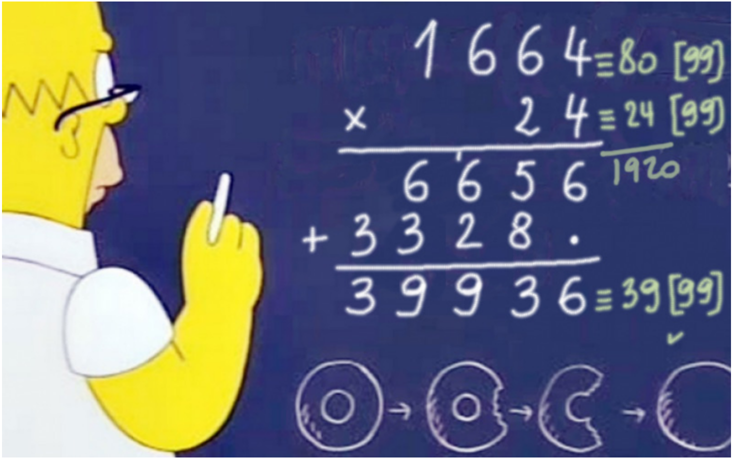
- est un moyen simple de vérification (avec certain indice de confiance)
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* 11 (obtenus en faisant la somme alternée de leurs chiffres, etc)

$$\sum_{k=0}^{p} a_k 10^k \equiv \sum_{k=0}^{p} (-1)^k a_k \pmod{11}$$

La "preuve" par 99 pour un calcul en base 10

- est un moyen simple de vérification (avec meilleur indice de confiance)
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* 99 (obtenus en faisant la somme des blocs de deux chiffres, etc)

$$\sum_{k=0}^{p} a_k 10^k \equiv \sum_{k=0}^{p/2} a_{2k+1} a_{2k} \pmod{99}$$


1 6 6 4 ≡ 80 [99]
× 2 4 ≡ 24 [99]
1920
6 6 5 6
+ 3 3 2 8 .
3 9 9 3 6 ≡ 39 [99]

La "preuve" par $b-1$ pour un calcul en base b

- n'est pas une *preuve* de correction, mais un moyen simple de vérification (avec un certain indice de confiance)
- utilise les propriétés de l'arithmétique modulaire
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* $b-1$ (obtenus en faisant la somme de leurs chiffres, etc)

$$\sum_{k=0}^{p} a_k b^k \equiv \sum_{k=0}^{p} a_k \pmod{b-1}$$

La "preuve" par $b+1$ pour un calcul en base b

- est un moyen simple de vérification (avec certain indice de confiance)
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* $b+1$ (obtenus en faisant la somme alternée de leurs chiffres, etc)

$$\sum_{k=0}^{p} a_k b^k \equiv \sum_{k=0}^{p} (-1)^k a_k \pmod{b+1}$$

La "preuve" par b^2-1 pour un calcul en base b

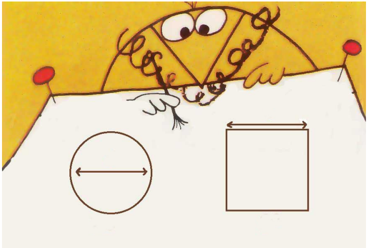
- est un moyen simple de vérification (avec meilleur indice de confiance)
- consiste à "refaire" le calcul mais en considérant les nombres *modulo* b^2-1 (obtenus en faisant la somme des blocs de deux chiffres, etc)

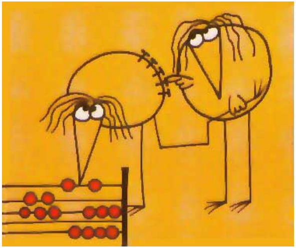
$$\sum_{k=0}^{p} a_k b^k \equiv \sum_{k=0}^{p/2} a_{2k+1} a_{2k} \pmod{b^2-1}$$

Numération positionnelle en base $b > 1$

- exactement b chiffres, disons $\{0, 1, \cdots, b-1\}$
- $(a_p \cdots a_0{,}a_{-1}a_{-2}\cdots a_{-q})_b$ représente le nombre réel $\sum_{k=-q}^{p} a_k b^k$, soit

$$a_p \times b^p + \cdots + a_2 \times b^2 + a_1 \times b + a_0 + a_{-1} \times \frac{1}{b} + a_{-2} \times \frac{1}{b^2} + \cdots + a_{-q} \times \frac{1}{b^q}$$





Comment convertir une écriture en base b $\quad (a_p \cdots a_0{,}a_{-1}a_{-2}\cdots)_b$

vers une écriture en base d? $\quad (c_q \cdots c_0{,}c_{-1}c_{-2}\cdots)_d$

Méthode (par divisions successives) pour la partie entière

- on convertit la partie entière $(a_p \cdots a_0)_b = (c_q \cdots c_0)_d$

Méthode par multiplications successives pour la partie fractionnaire

- on multiplie $(0{,}a_{-1} \cdots a_{-t})_b$ par d (calcul en base b toujours)
- on collecte la partie entière c_{-1} de ce produit
- on recommence avec sa partie fractionnaire
- on s'arrête quand $\quad (0{,}c_{-1} \cdots c_{-r})_d$
  le produit est nul: on renvoie la suite finie des chiffres collectés
  ou déjà obtenu: on renvoie la suite ultimement périodique
  $$(0{,}\underbrace{c_{-1} \cdots c_{-r}}_{\text{prépériode}}(\underbrace{c_{-r-1} \cdots c_{-r-s}}_{\text{période}})^{\omega})_d$$

Numération positionnelle en base $b > 1$

- exactement b chiffres, disons $\{0, 1, \cdots, b-1\}$
- $(a_p \cdots a_0{,}a_{-1} \cdots a_{-q})_b$ représente le nombre réel $\sum_{k=-q}^{p} a_k b^k$
- $(a_p \cdots a_0{,}a_{-1} a_{-2} \cdots)_b$ représente le nombre réel $\sum_{k=-\infty}^{p} a_k b^k$

Non-unicité des représentations des réels

- certains nombres réels admettent plusieurs représentations dans une même base

$$(1)_{10} = (0,9999\cdots)_{10}$$

$$(1)_{d+1} = (0,dddd\cdots)_{d+1}$$

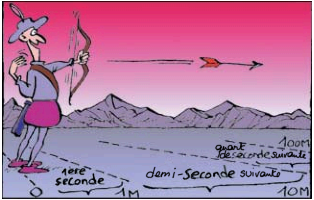
Les nombres b-adiques pour une base $b > 1$

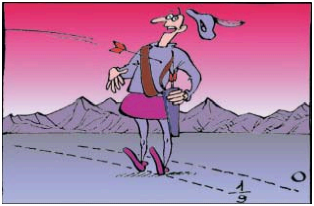
- exactement b chiffres, disons $\{0, 1, \cdots, b-1\}$
- $(\cdots a_2 a_1 a_0, a_{-1} a_{-2} \cdots a_{-q})_b$ représente le nombre b-adique $\sum_{k=-q}^{+\infty} a_k b^k$
- tout nombre b-adique admet un opposé b-adique (ou *complément*)

$$(\cdots 001)_{10} + (\cdots 999)_{10} = 0$$

$$(\cdots 001)_2 + (\cdots 111)_2 = 0$$

$$(\cdots 001)_{d+1} + (\cdots dddd)_{d+1} = 0$$





(si) on manipule rarement les très très très graaaaaaaaaands nombres
(si) on manipule rarement les nombres avec une graaaaande précision

Choix de fixer la taille des représentations

- architectures communes 32 ou 64 bits: arithmétique sur des nombres représentés sur 32 ou 64 bits
  binary digit

$$2^{32} \sim 4,3 \times 10^9 \qquad 2^{64} \sim 18,4 \times 10^{18}$$

- une infinité de choix pour représenter des entiers!

Un choix parmi les $2^{32}!$ choix pour représenter les entiers de 0 à $2^{32} - 1$

mots sur $\{0, 1\}$ dans l'ordre lexicographique	base 2	base 10
00000000 00000000 00000000 00000000	0	0
00000000 00000000 00000000 00000001	1	1
00000000 00000000 00000000 00000010	10	2
00000000 00000000 00000000 00000011	11	3
...		...
11111111 11111111 11111111 11111110		4 294 967 294
11111111 11111111 11111111 11111111		4 294 967 295

représentation non-signée

Un parmi les $2^{32}!$ choix pour représenter les entiers de $-2^{31}+1$ à $2^{31}-1$

mots sur $\{0,1\}$	base 10
11111111 11111111 11111111 11111111	$-2\,147\,483\,647$
11111111 11111111 11111111 11111110	$-2\,147\,483\,646$
...	...
10000000 00000000 00000000 00000010	-2
10000000 00000000 00000000 00000001	-1
10000000 00000000 00000000 00000000	-0
00000000 00000000 00000000 00000000	0
00000000 00000000 00000000 00000001	1
00000000 00000000 00000000 00000010	2
...	...
01111111 11111111 11111111 11111110	$2\,147\,483\,646$
01111111 11111111 11111111 11111111	$2\,147\,483\,647$

bit de signe

inadapté au calcul

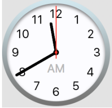
Un parmi les $2^{32}!$ choix pour représenter les entiers de -2^{31} à $2^{31}-1$

mots sur $\{0,1\}$	base 10
10000000 00000000 00000000 00000000	− 2 147 483 648
10000000 00000000 00000000 00000001	− 2 147 483 647
...	
11111111 11111111 11111111 11111101	− 3
11111111 11111111 11111111 11111110	− 2
11111111 11111111 11111111 11111111	− 1
00000000 00000000 00000000 00000000	0
00000000 00000000 00000000 00000001	1
00000000 00000000 00000000 00000010	2
...	...
01111111 11111111 11111111 11111110	2 147 483 646
01111111 11111111 11111111 11111111	2 147 483 647

bit de signe

complément à 2

midi **moins** vingt
twenty **to** noon



midi (**et**) vingt
twenty **past** noon

Un parmi les $2^{32}!$ choix pour représenter les entiers de -2^{31} à $2^{31}-1$

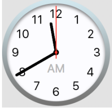
mots sur $\{0,1\}$	base 10
10000000 00000000 00000000 00000000	−2 147 483 648
10000000 00000000 00000000 00000001	−2 147 483 647
...	
11111111 11111111 11111111 11111101	−3
11111111 11111111 11111111 11111110	−2
11111111 11111111 11111111 11111111	−1
00000000 00000000 00000000 00000000	0
00000000 00000000 00000000 00000001	1
00000000 00000000 00000000 00000010	2
...	...
01111111 11111111 11111111 11111110	2 147 483 646
01111111 11111111 11111111 11111111	2 147 483 647

bit de signe

complément à 2

1. inverser tous les bits
2. ajouter 1

midi **moins** vingt
twenty **to** noon



midi (**et**) vingt
twenty **past** noon