Architecture des Ordinateurs Avancée (L3)

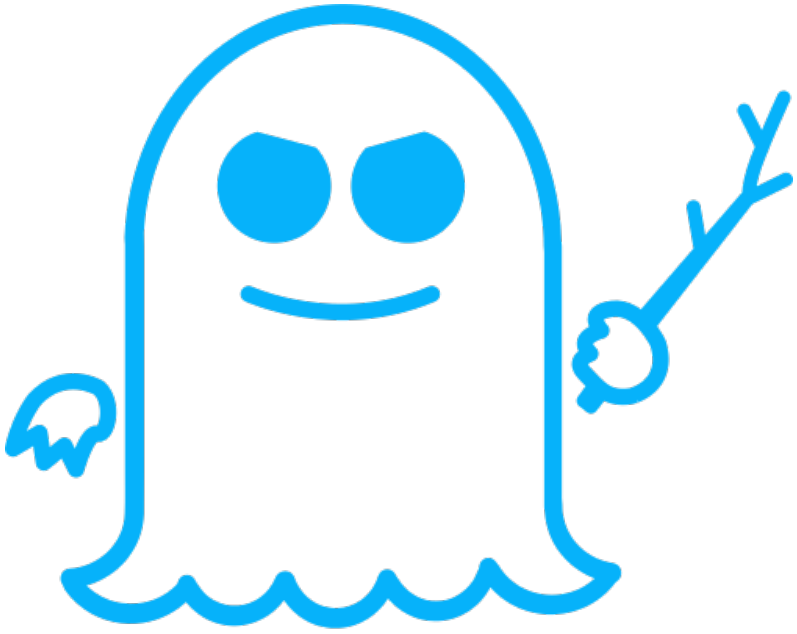
Cours 1 - Encodage des nombres

Carine Pivoteau [1]

1. En collaboration avec X. Goaoc

Une petite faille informatique...

Janvier 2018 :



SPECTRE

MELTDOWN

Spectre

source : `https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)`

Spectre is a vulnerability that affects modern microprocessors that perform branch prediction.[1][2][3] On most processors, the speculative execution resulting from a branch misprediction may leave observable side effects that may reveal private data to attackers. For example, if the pattern of memory accesses performed by such speculative execution depends on private data, the resulting state of the data cache constitutes a side channel through which an attacker may be able to extract information about the private data using a timing attack.[4][5][6]

Two Common Vulnerabilities and Exposures IDs related to Spectre, CVE-2017-5753 (bounds check bypass, Spectre-V1, Spectre 1.0) and CVE-2017-5715 (branch target injection, Spectre-V2), have been issued.[7] JIT engines used for JavaScript were found to be vulnerable. A website can read data

Impact [edit]

As of 2018, almost every computer system is affected by Spectre, including desktops, laptops, and mobile devices. Specifically, Spectre has been shown to work on Intel, AMD, ARM-based, and IBM processors.[56][57][58] Intel responded to the reported security vulnerabilities with an official statement.[59] AMD originally acknowledged vulnerability to one of the Spectre variants (GPZ variant 1), but stated that vulnerability to another (GPZ variant 2) had not been demonstrated on AMD processors, claiming it posed a "near zero risk of exploitation" due to differences in AMD architecture. In an update nine days later, AMD said that "GPZ Variant 2…is applicable to AMD processors" and defined upcoming steps to mitigate the threat. Several sources took AMD's news of the vulnerability to GPZ variant 2 as a change from AMD's prior claim, though AMD maintained that their position had not changed.[60][61][62]

Meltdown

source : `https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)`

Overview [edit]

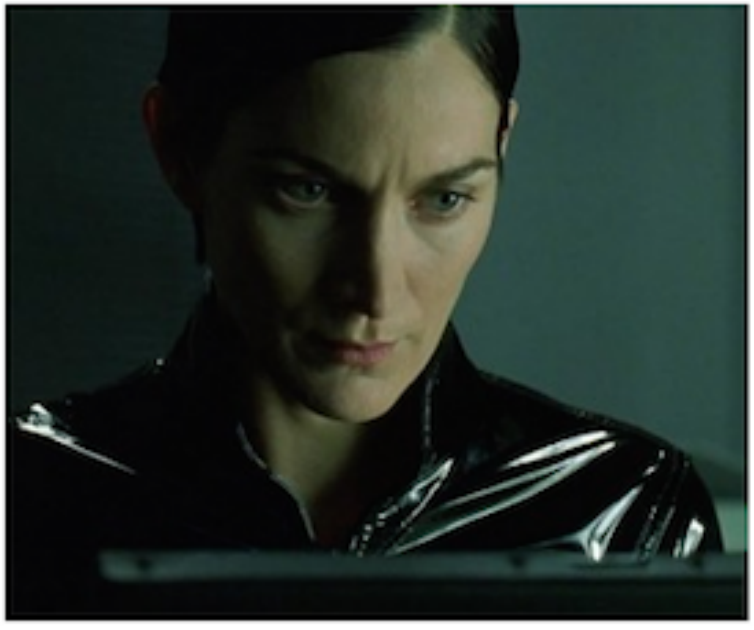
Meltdown exploits a race condition, inherent in the design of many modern CPUs. This occurs between memory access and privilege checking during instruction processing. Additionally, combined with a cache side-channel attack, this vulnerability allows a process to bypass the normal privilege checks that isolate the exploit process from accessing data belonging to the operating system and other running processes. The vulnerability allows an unauthorized process to read data from any address that is mapped to the current process's memory space. Since instruction pipelining is in the affected processors, the data from an unauthorized address will almost always be temporarily loaded into the CPU's cache during out-of-order execution —from which the data can be recovered. This can occur even if the original read instruction fails due to privilege checking, and/or if it never produces a readable result.

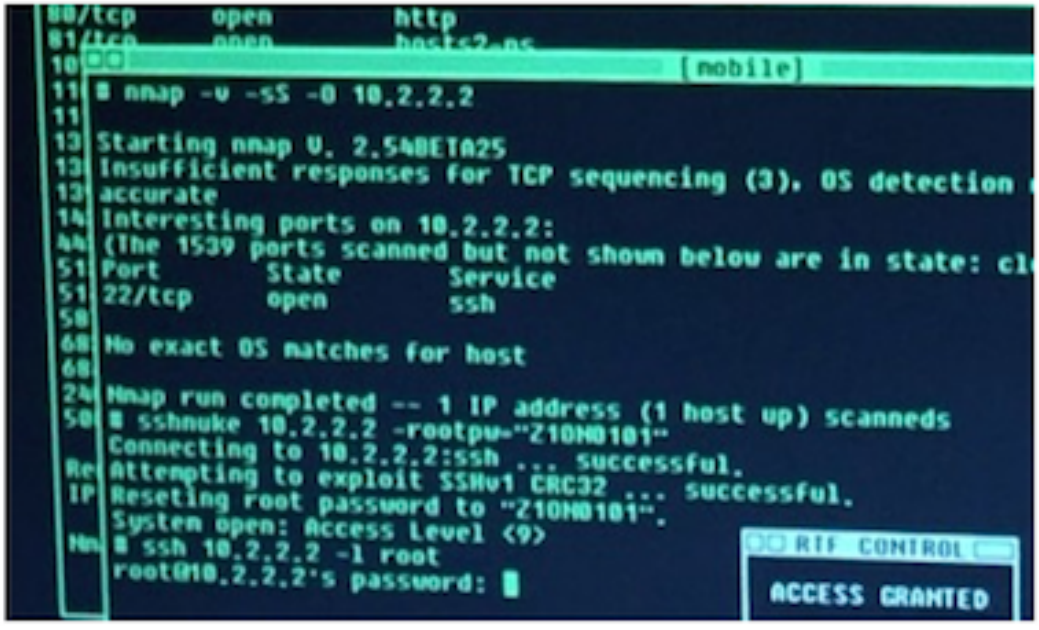
Affected hardware [edit]

The Meltdown vulnerability primarily affects Intel microprocessors,[54] but some ARM microprocessors are also affected.[55] The vulnerability does not affect AMD microprocessors.[19][56][57][58] Intel has countered that the flaws affect all processors,[59] but AMD has denied this, saying "we believe AMD processors are not susceptible due to our use of privilege level protections within paging architecture".[60]

Objectif du cours

Devenir un hacker en 6 leçons





https://www.youtube.com/watch?v=0PxTAn4g20U

Objectif du cours

~~Devenir un hacker en 6 leçons~~

Objectif du cours

~~Devenir un hacker en 6 leçons~~

- Maîtriser les éléments avancés (mais relativement classiques) de l'architecture d'un processeur moderne.
- Être conscient des conséquences que cela peut avoir sur l'exécution de vos programmes...
- ... pour apprendre à en tirer parti :
  - en évitant les écueils,
  - et en exploitant les optimisations.
- Et au final, écrire du code plus efficace !
- Et aussi acquérir un peu de culture générale, en passant...

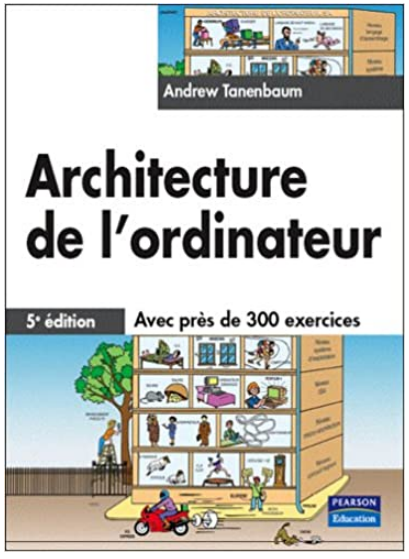
Objectif du cours

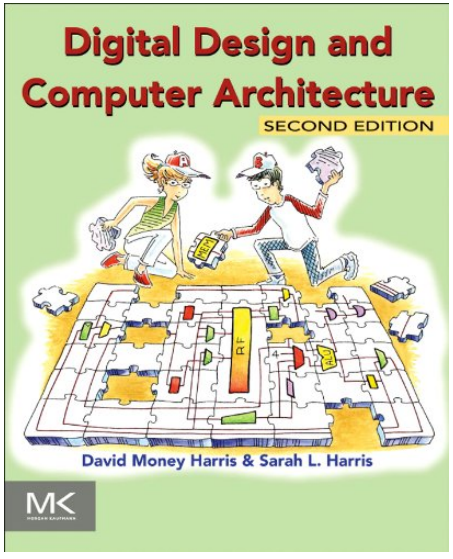
Bref, comme dirait M. Cherrier : être **aware of the hardware !**

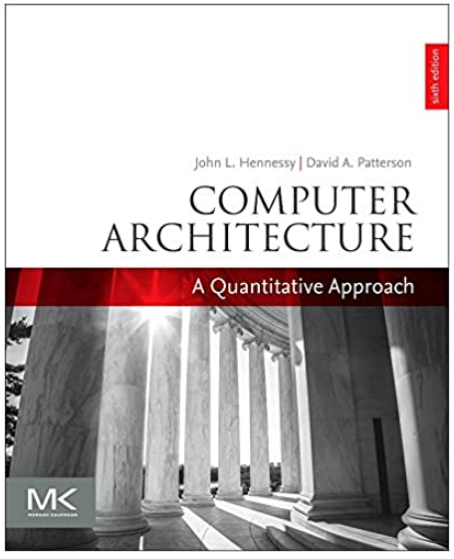
- Maîtriser les éléments avancés (mais relativement classiques) de l'architecture d'un processeur moderne.
- Être conscient des conséquences que cela peut avoir sur l'exécution de vos programmes...
- ... pour apprendre à en tirer parti :
  - en évitant les écueils,
  - et en exploitant les optimisations.
- Et au final, écrire du code plus efficace !
- Et aussi acquérir un peu de culture générale, en passant...

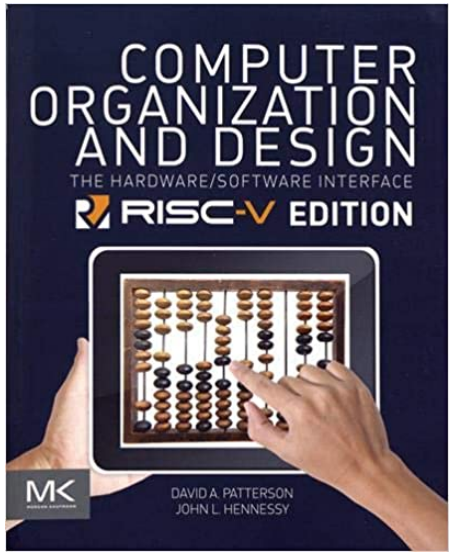
De la lecture...

- *Tanenbaum* pour les bases,
- *Harris et Harris* pour aller un peu plus loin,
- *Hennessy et Patterson* pour tout comprendre.









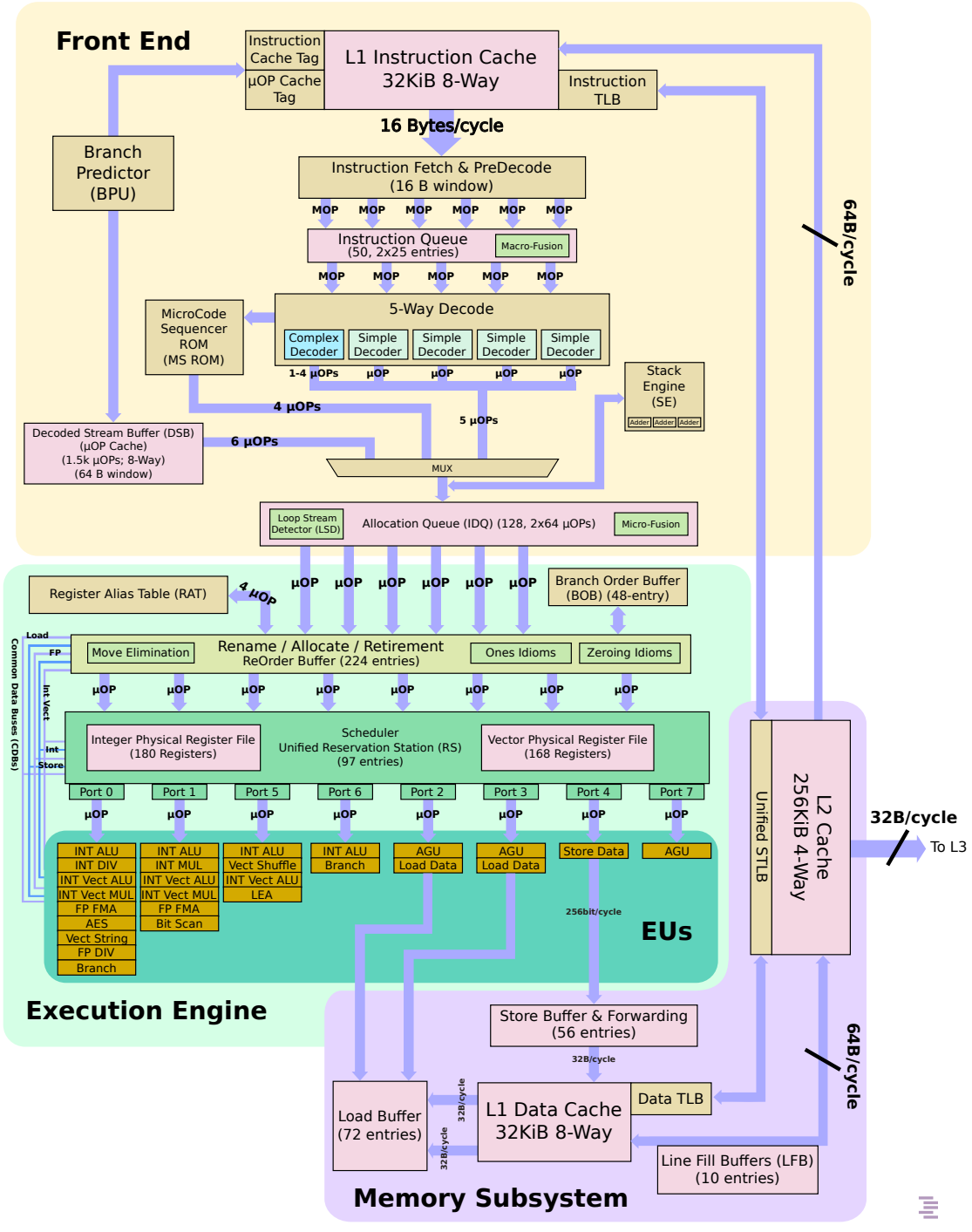
Contenu du cours

- calcul (entiers, flottants)
- méthodologie de mesure du temps d'exécution, avec application aux accès mémoire
- organisation de la mémoire, coût des accès
- parallélisme d'instructions, pipeline
- optimisations de compilation (avec `gcc`) liées à l'architecture
- prédicteurs de branchement
- vectorisation

Architecture Skylake (Intel Core i3, i5, i7)

source :

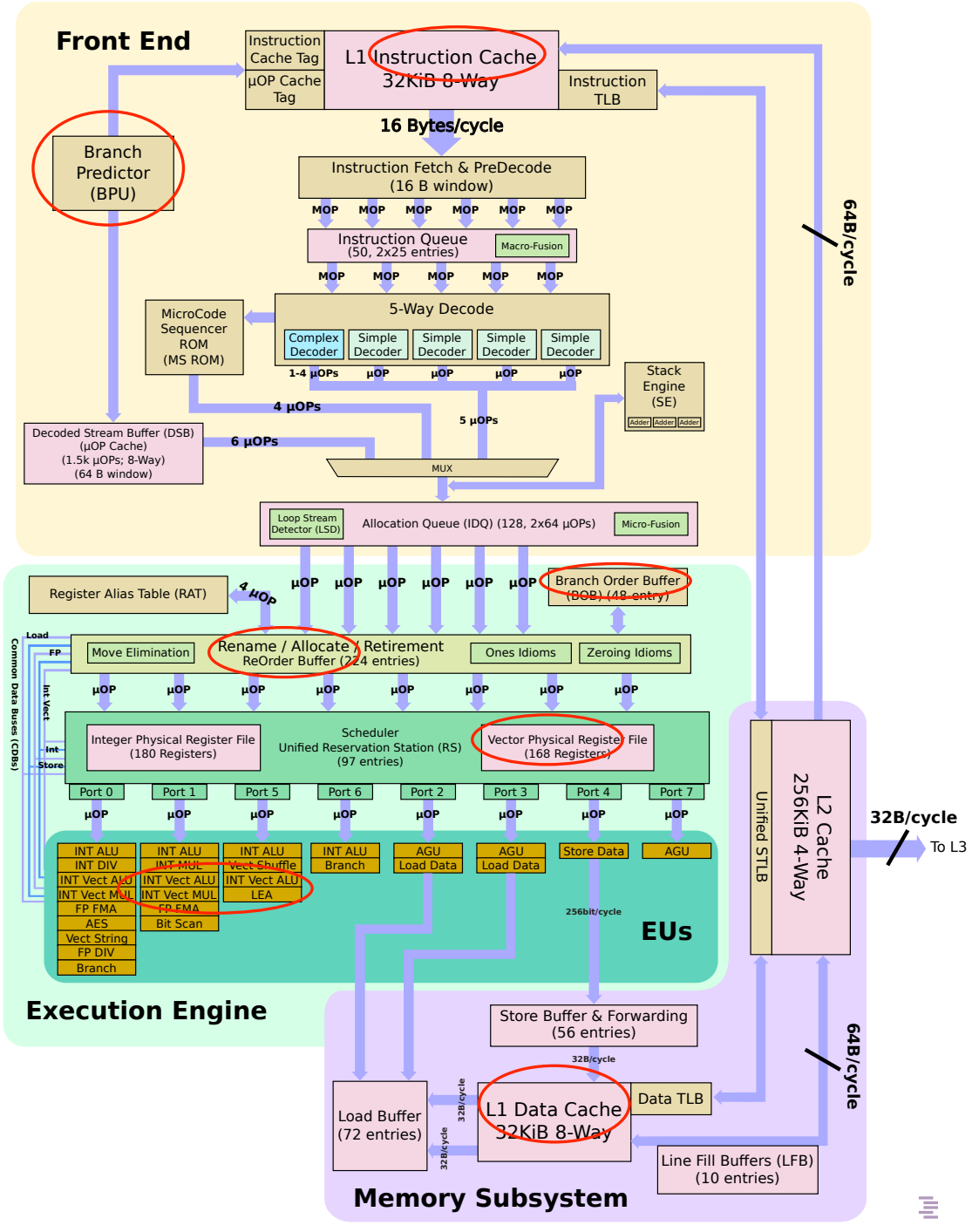
`https://en.wikichip.org/wiki/skylake`



Architecture Skylake (Intel Core i3, i5, i7)

source :

`https://en.wikichip.org/wiki/skylake`



Organisation du cours

CM sur discord

1 seul groupe de TP, 2 intervenants (c'est du luxe !) :

- Wenjie Fang
- Carine Pivoteau

Organisation :

- 6 séances de CM (avec de expériences faites par les étudiants et des exos) pour préparer aux TPs
- 6 séances de TP : manipulations (encore des expériences) + analyse
- **le plus important : l'analyse !**
- TPs à rendre (analyse + code sur e-learning)

Représentation des entiers en machine

Le cas d'Ariane 5

Vol inaugural du lanceur européen Ariane 5, le 4 juin 1996 :

https://www.youtube.com/watch?v=gp_D8r-2hwk

Le cas d'Ariane 5

Vol inaugural du lanceur européen Ariane 5, le 4 juin 1996 :

https://www.youtube.com/watch?v=gp_D8r-2hwk

La fusée a explosé à 4 000 mètres au-dessus du centre spatial de Kourou, en Guyane. Heureusement, il n'y a eu aucune victime.

Le cas d'Ariane 5

Ariane 501 Inquiry Board report
page 4

- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

L'incident, dû à un dépassement d'entier (une mauvaise conversion d'un nombre flottant 64 bits en entier 16 bits...) dans les registres mémoire des calculateurs électroniques utilisés par le pilote automatique, a provoqué la panne du système de navigation de la fusée, causant sa destruction.

Remarque : c'est une erreur qu'il est possible de détecter au niveau du processeur, mais les vérifications nécessaires ne se trouvaient pas dans le code.

Coût : 370 millions de dollars.

détails : `http://sunnyday.mit.edu/nasa-class/Ariane5-report.html`

Rappels : décimal / binaire / hexadécimal

Tout entier naturel compris entre 0 et $2^k - 1$ s'écrit de manière unique

$$a_0 + a_1 \times 2 + a_2 \times 2^2 + a_3 \times 2^3 + \cdots = \sum_{i=0}^{k-1} a_i 2^i$$

avec $a_i \in \{0, 1\}$. Son codage binaire est noté $(a_{k-1} a_{k-2} \ldots a_0)_2$.

1. Comment convertir un nombre binaire en base 10 ?
2. Et dans l'autre sens (décimal vers binaire) ?
3. Et entre les nombres décimaux et hexadécimaux (base 16) ?

Rappels : décimal / binaire / hexadécimal

Tout entier naturel compris entre 0 et $2^k - 1$ s'écrit de manière unique

$$a_0 + a_1 \times 2 + a_2 \times 2^2 + a_3 \times 2^3 + \cdots = \sum_{i=0}^{k-1} a_i 2^i$$

avec $a_i \in \{0, 1\}$. Son codage binaire est noté $(a_{k-1} a_{k-2} \ldots a_0)_2$.

1. Comment convertir un nombre binaire en base 10 ?
   ▷ Avec la formule
2. Et dans l'autre sens (décimal vers binaire) ?

3. Et entre les nombres décimaux et hexadécimaux (base 16) ?

Rappels : décimal / binaire / hexadécimal

Tout entier naturel compris entre 0 et $2^k - 1$ s'écrit de manière unique

$$a_0 + a_1 \times 2 + a_2 \times 2^2 + a_3 \times 2^3 + \cdots = \sum_{i=0}^{k-1} a_i 2^i$$

avec $a_i \in \{0, 1\}$. Son codage binaire est noté $(a_{k-1} a_{k-2} \ldots a_0)_2$.

1. Comment convertir un nombre binaire en base 10 ?
   ▷ Avec la formule
2. Et dans l'autre sens (décimal vers binaire) ?
   ▷ Divisions par 2 successives. Pourquoi ça marche ?
3. Et entre les nombres décimaux et hexadécimaux (base 16) ?

Rappels : décimal / binaire / hexadécimal

Tout entier naturel compris entre 0 et $2^k - 1$ s'écrit de manière unique

$$a_0 + a_1 \times 2 + a_2 \times 2^2 + a_3 \times 2^3 + \cdots = \sum_{i=0}^{k-1} a_i 2^i$$

avec $a_i \in \{0, 1\}$. Son codage binaire est noté $(a_{k-1} a_{k-2} \ldots a_0)_2$.

1. Comment convertir un nombre binaire en base 10 ?
   ▷ Avec la formule
2. Et dans l'autre sens (décimal vers binaire) ?
   ▷ Divisions par 2 successives. Pourquoi ça marche ?
3. Et entre les nombres décimaux et hexadécimaux (base 16) ?
   ▷ En passant par le binaire

Exo : conversions

- Convertir 37 en binaire.
- Convertir $(1001101)_2$ en décimal
- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$

Exo : conversions

- Convertir 37 en binaire.
  $\triangleright 37 = 32 + 4 + 1$, donc $(100101)_2$

- Convertir $(1001101)_2$ en décimal

- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$

Exo : conversions

- Convertir 37 en binaire.
  $\triangleright 37 = 32 + 4 + 1$, donc $(100101)_2$
- Convertir $(1001101)_2$ en décimal
  $\triangleright$ 77
- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$

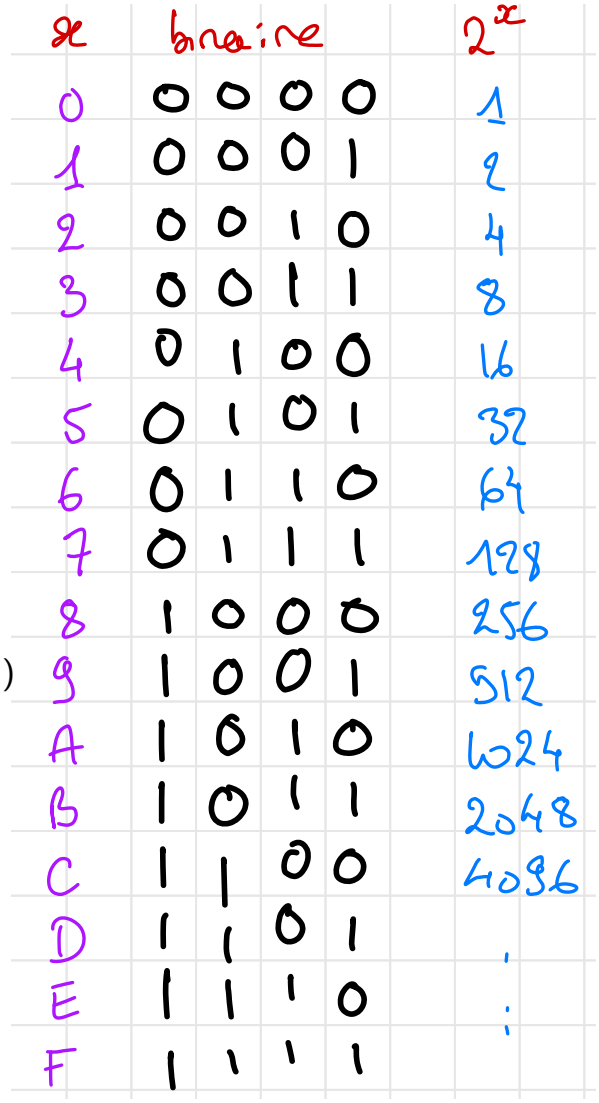
Exo : conversions

- Convertir 37 en binaire.
  $\triangleright 37 = 32 + 4 + 1$, donc $(100101)_2$
- Convertir $(1001101)_2$ en décimal
  $\triangleright$ $77 = ((((((1 \times 2 + 0) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1)$
- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$

remarque : $a_0 + 2a_1 + 2^2 a_2 + \cdots + 2^n a_n = a_0 + 2(a_1 + 2(a_2 + \cdots + (2a_n)))$

Exo : conversions

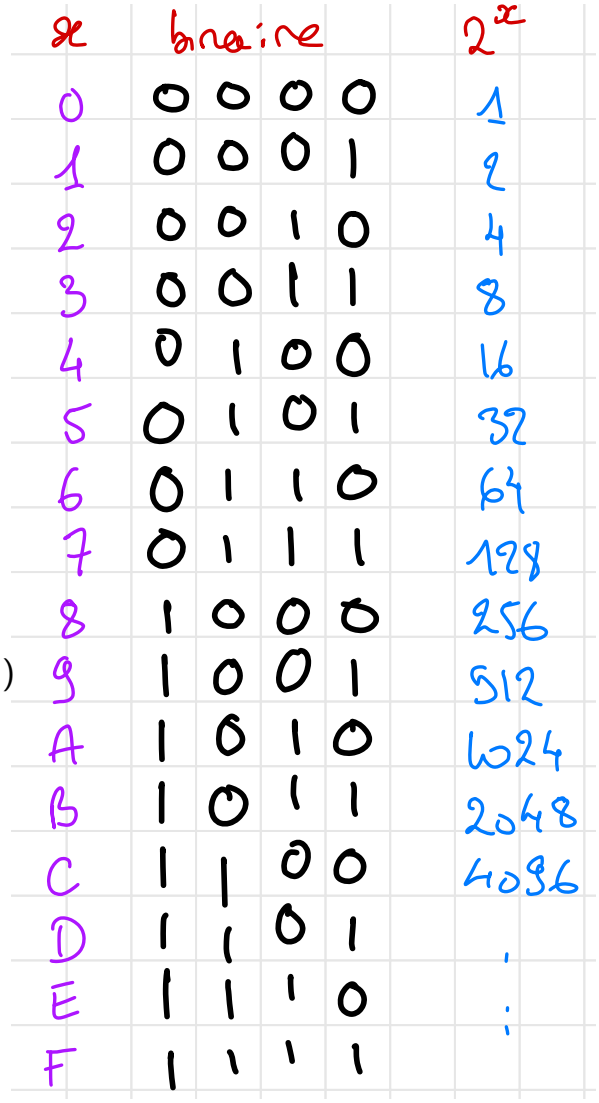
- Convertir 37 en binaire.
  $\triangleright 37 = 32 + 4 + 1$, donc $(100101)_2$
- Convertir $(1001101)_2$ en décimal
  $\triangleright\ 77 = ((((((1 \times 2 + 0) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1)$
- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$
  $\triangleright\ 0x4c809743$



remarque : $a_0 + 2a_1 + 2^2 a_2 + \cdots + 2^n a_n = a_0 + 2(a_1 + 2(a_2 + \cdots + (2a_n)))$

Exo : conversions

- Convertir 37 en binaire.
  $\triangleright 37 = 32 + 4 + 1$, donc $(100101)_2$
- Convertir $(1001101)_2$ en décimal
  $\triangleright\ 77 = ((((((1 \times 2 + 0) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1)$
- Convertir ce nombre binaire en hexadécimal :
  $(0100\,1100\,1000\,0000\,1001\,0111\,0100\,0011)_2$
  $\triangleright\ 0x4c809743$



Et les calculs ?

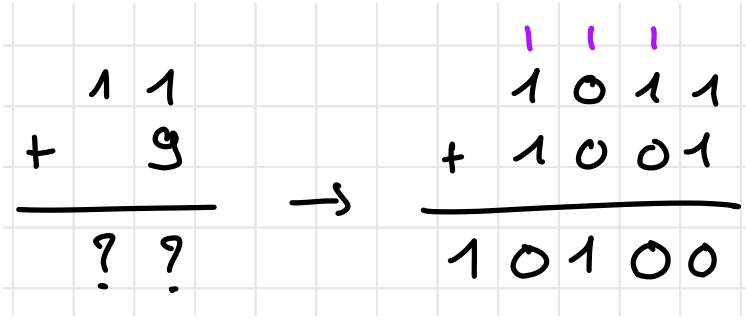
remarque : $a_0 + 2a_1 + 2^2 a_2 + \cdots + 2^n a_n = a_0 + 2(a_1 + 2(a_2 + \cdots + (2a_n)))$

Exo : addition binaire

- Faire (poser) l'addition suivante en binaire : $11 + 9$.
- Combien de bits son nécessaires pour représenter les opérandes ?
- Et le résultat ?
- Et la multiplication (11×9) ?

Exo : addition binaire

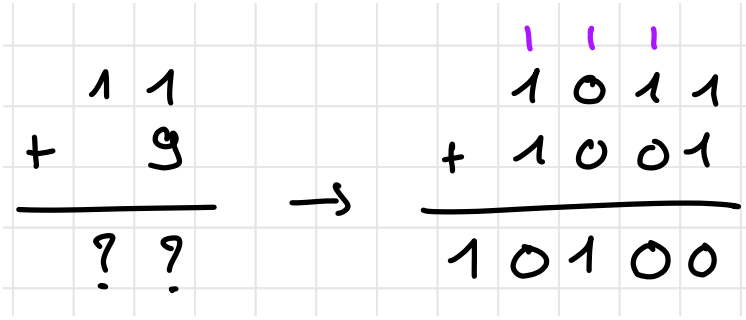
- Faire (poser) l'addition suivante en binaire : $11 + 9$.
  $\triangleright$ $(1011)_2 + (1001)_2 = (10100)_2$
- Combien de bits son nécessaires pour représenter les opérandes ?
- Et le résultat ?
- Et la multiplication (11×9) ?



Exo : addition binaire

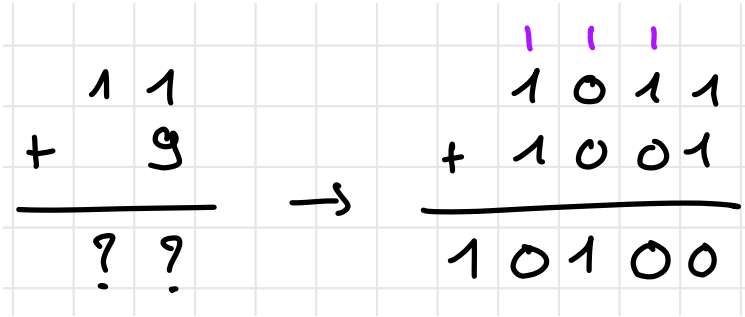
- Faire (poser) l'addition suivante en binaire : $11 + 9$.
  $\triangleright$ $(1011)_2 + (1001)_2 = (10100)_2$
- Combien de bits son nécessaires pour représenter les opérandes ?
  $\triangleright$ 4
- Et le résultat ?

- Et la multiplication (11×9) ?



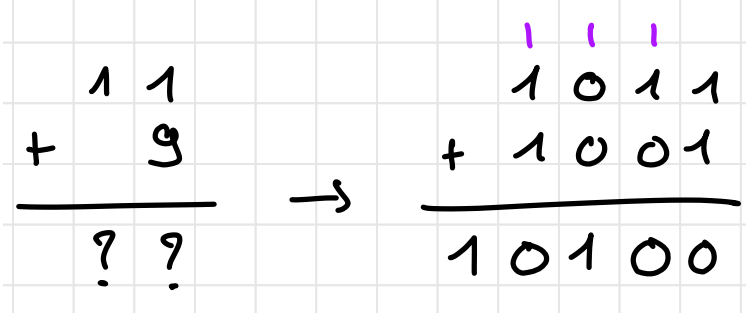
Exo : addition binaire

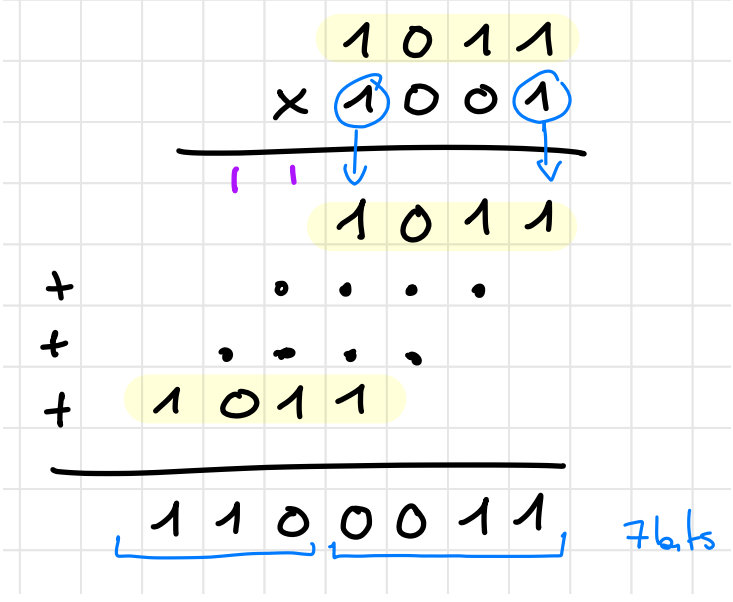
- Faire (poser) l'addition suivante en binaire : $11 + 9$.
  $\triangleright$ $(1011)_2 + (1001)_2 = (10100)_2$
- Combien de bits son nécessaires pour représenter les opérandes ?
  $\triangleright$ 4
- Et le résultat ?
  $\triangleright$ 5
- Et la multiplication (11×9) ?



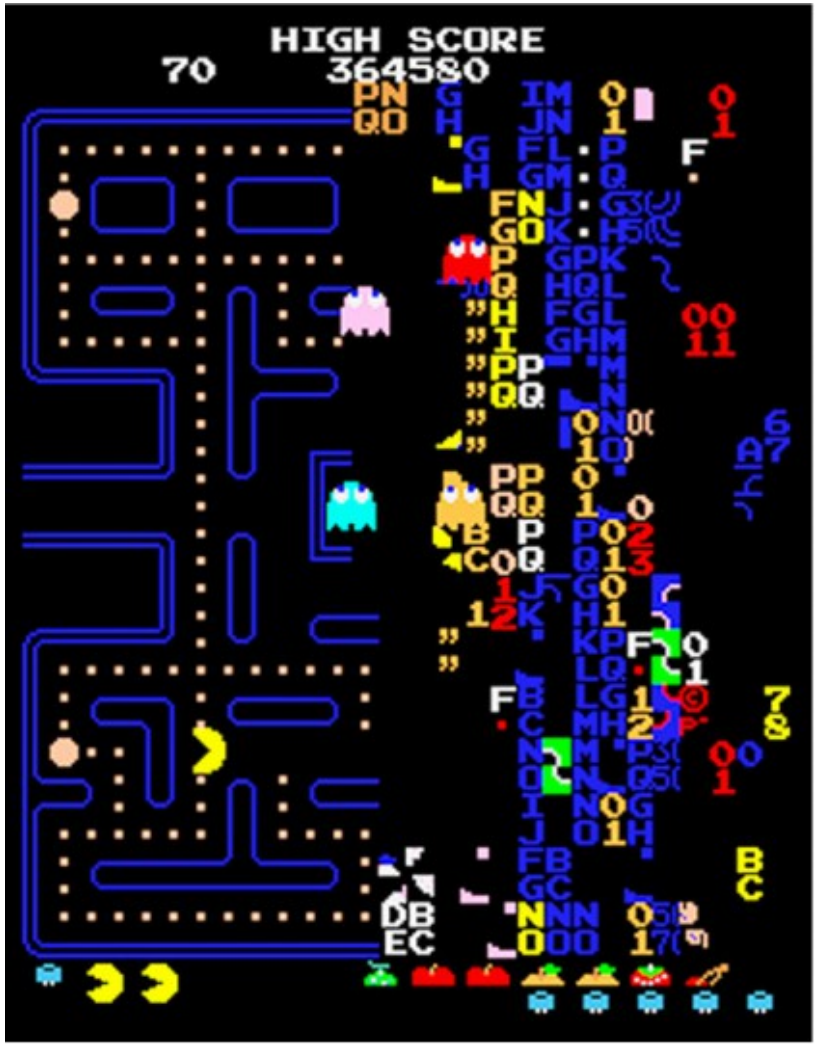
Exo : addition binaire

- Faire (poser) l'addition suivante en binaire : $11 + 9$.
  ▷ $(1011)_2 + (1001)_2 = (10100)_2$
- Combien de bits son nécessaires pour représenter les opérandes ?
  ▷ 4
- Et le résultat ?
  ▷ 5
- Et la multiplication (11×9) ?
  ▷ $(1100011)_2$: 7 bits





Le 256^{e} niveau de PAC-MAN (1980)

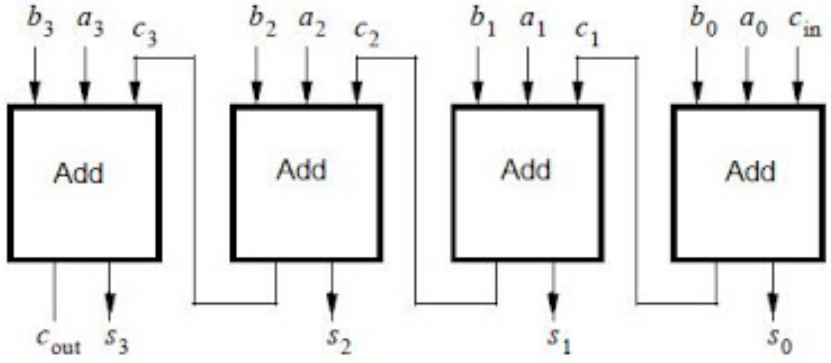


https://www.youtube.com/watch?v=NKKfW8X9uYk

Arithmétique finie

L'arithmétique des ordinateurs est une arithmétique finie !

- Les types des nombres que l'on manipule définissent le nombre de bits de de leur codage binaire, et donc l'intervalle des valeurs représentables.
- Les calculs peuvent donc entraîner une perte d'information.
- Pour l'addition, le processeur garde l'information de la retenue finale (carry) :

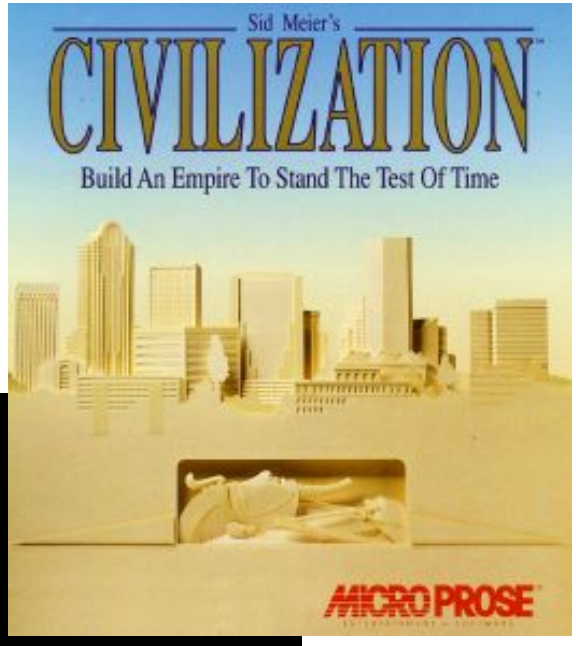


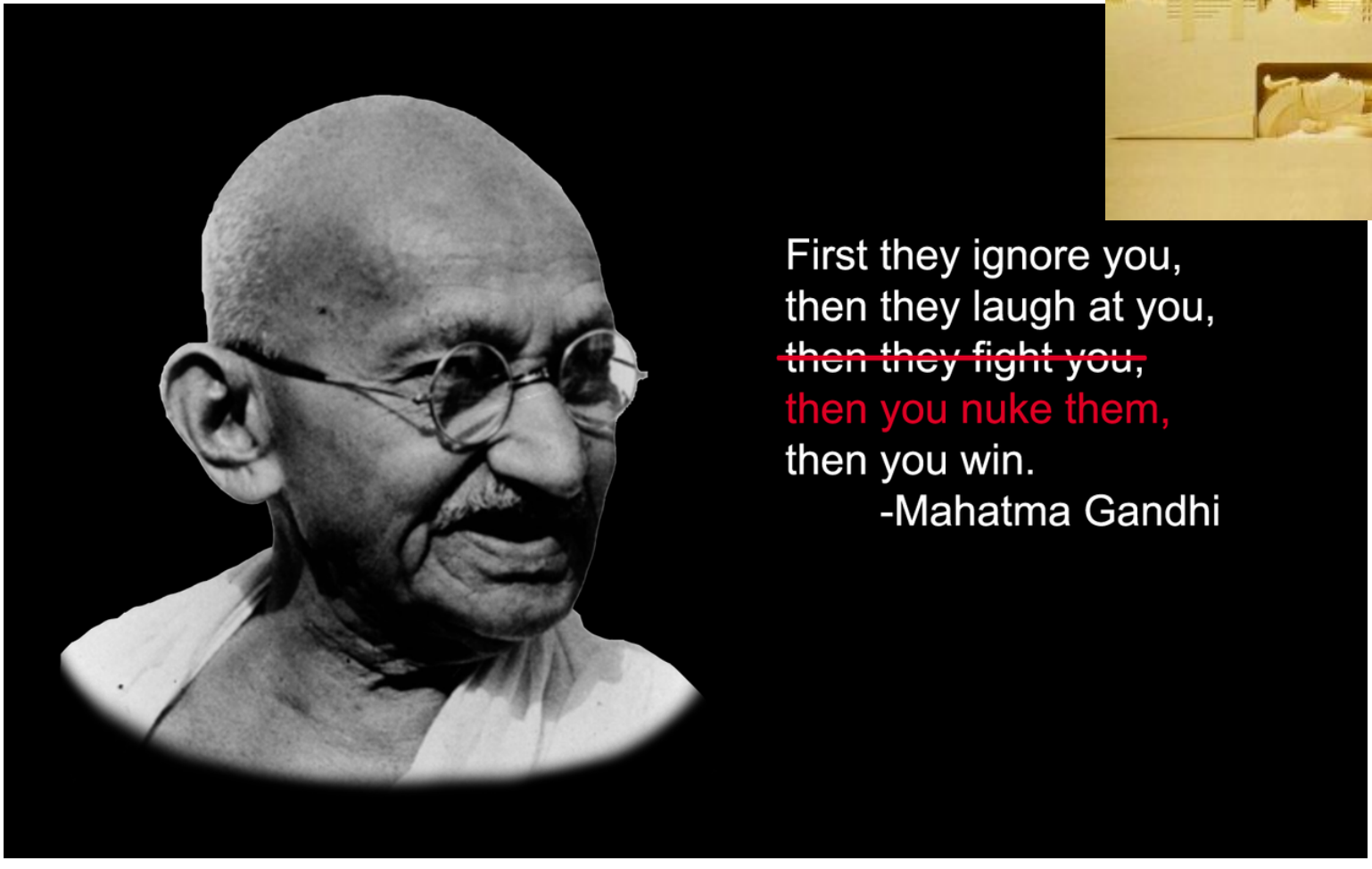
- Pour la multiplication, il calcule le résultat sur 2 registres.

Et les négatifs ?

Civilization I : Nuclear Gandhi ...

... vrai bug ou légende urbaine ?



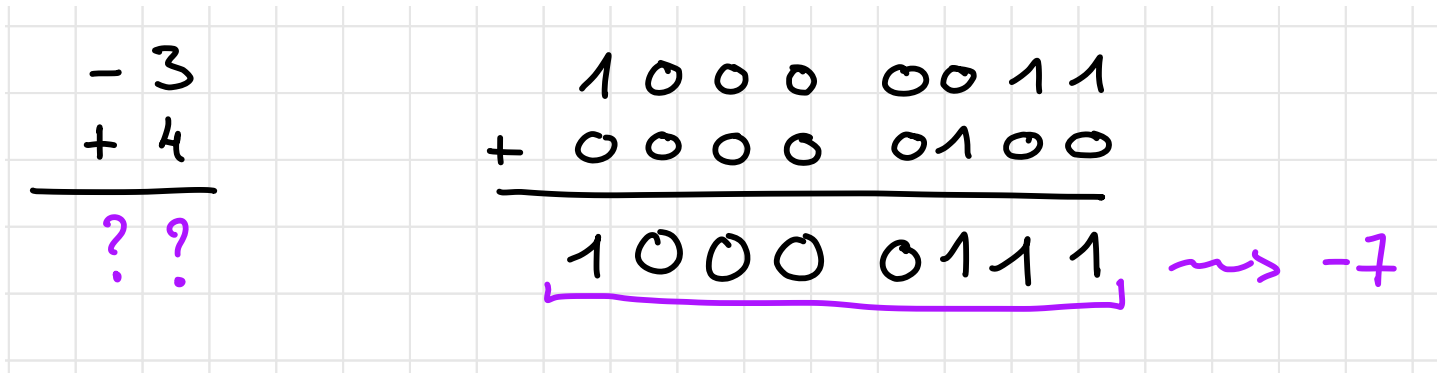


Représenter les entiers relatifs

- Idée naturelle : sacrifier un bit de signe et utiliser la valeur absolue
  - Est-ce satisfaisant ?
  - Faire le test en posant l'addition : $-3 + 4$ sur 8 bits
  - Comment construire un circuit additionneur avec cette convention ?
  - Autre problème : il y a 2 zéros, et donc seulement 255 valeurs.

Représenter les entiers relatifs

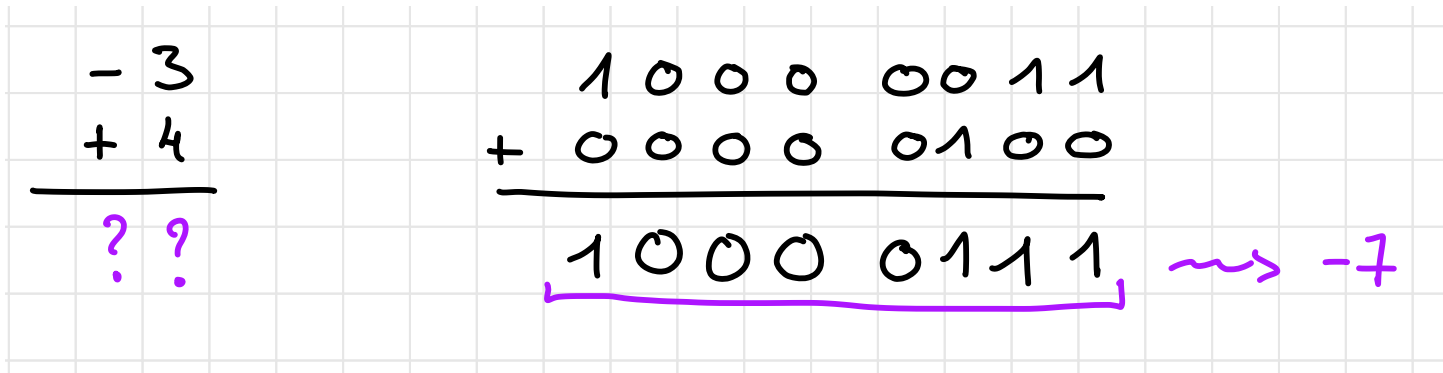
- Idée naturelle : sacrifier un bit de signe et utiliser la valeur absolue
  - Est-ce satisfaisant ?
  - Faire le test en posant l'addition : $-3+4$ sur 8 bits



  - Comment construire un circuit additionneur avec cette convention ?
  - Autre problème : il y a 2 zéros, et donc seulement 255 valeurs.

Représenter les entiers relatifs

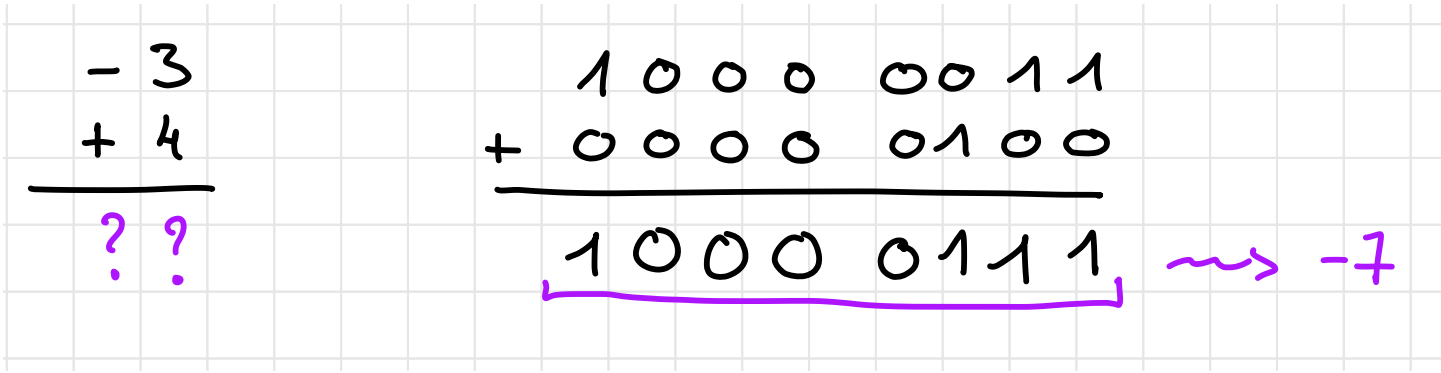
- Idée naturelle : sacrifier un bit de signe et utiliser la valeur absolue
  - Est-ce satisfaisant ?
  - Faire le test en posant l'addition : $-3+4$ sur 8 bits



  - Comment construire un circuit additionneur avec cette convention ?
  - Autre problème : il y a 2 zéros, et donc seulement 255 valeurs.

Représenter les entiers relatifs

- Idée naturelle : sacrifier un bit de signe et utiliser la valeur absolue
  - Est-ce satisfaisant ?
  - Faire le test en posant l'addition : $-3 + 4$ sur 8 bits



  - Comment construire un circuit additionneur avec cette convention ?
  - Autre problème : il y a 2 zéros, et donc seulement 255 valeurs.

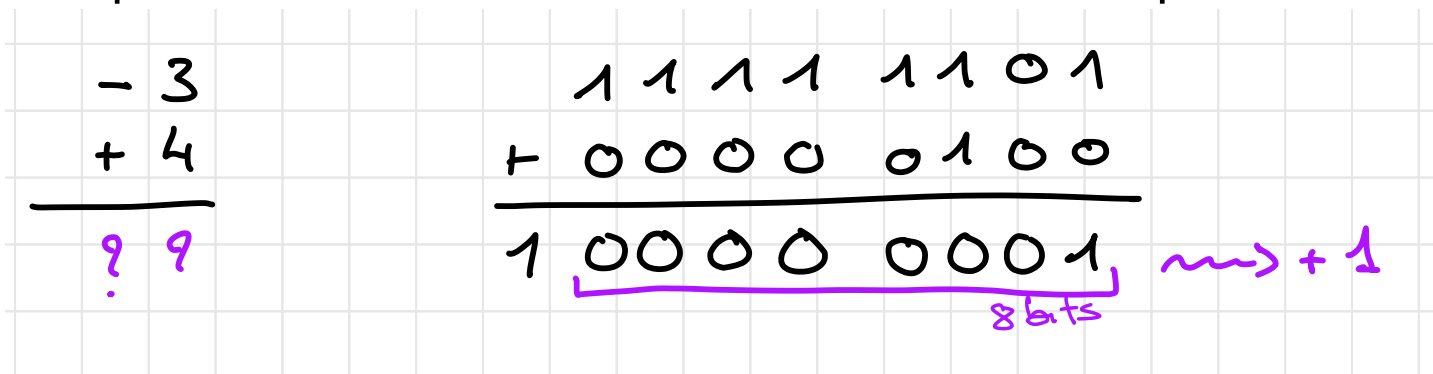
Représenter les entiers relatifs

- Autre idée : on aimerait que les opérations suivantes se fassent naturellement en binaire : $-1+1=0$, $-2+1=-1$, ...
  - Règle du complément à 2 sur 8 bits : on écrit $-x$ comme $256-x$ c'est à dire $255+1-x = \text{not}(x)+1$.
  - Autrement dit, pour convertir les entiers négatifs (dans les deux sens), on inverse les bits et on ajoute 1.
- Test : poser l'addition $-3+4$ sur 8 bits en complément à 2.
- Et le problème des 2 zéros ?

Représenter les entiers relatifs

- Autre idée : on aimerait que les opérations suivantes se fassent naturellement en binaire : $-1+1=0$, $-2+1=-1$, ...
  - Règle du complément à 2 sur 8 bits : on écrit $-x$ comme $256-x$ c'est à dire $255+1-x = \text{not}(x)+1$.
  - Autrement dit, pour convertir les entiers négatifs (dans les deux sens), on inverse les bits et on ajoute 1.

- Test : poser l'addition $-3+4$ sur 8 bits en complément à 2.

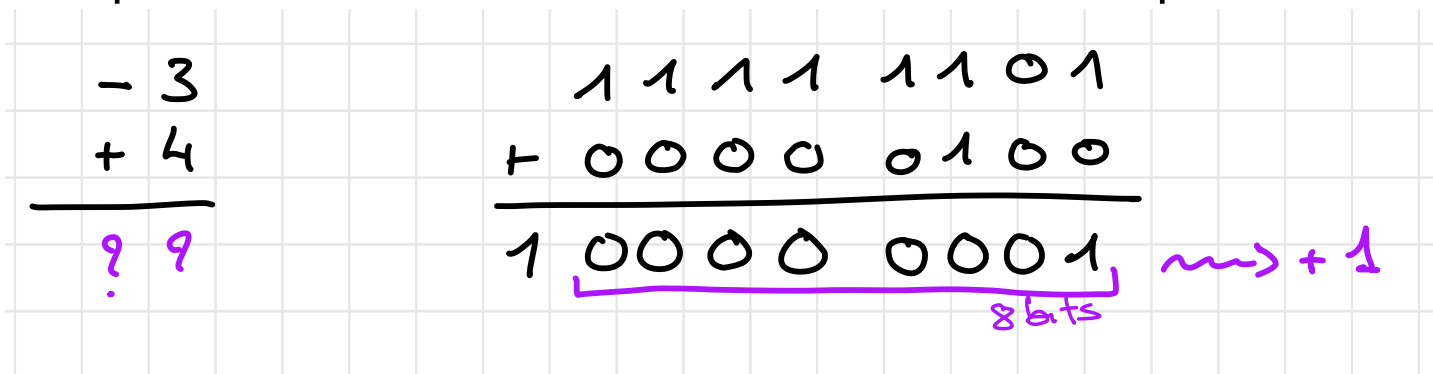


- Et le problème des 2 zéros ?

Représenter les entiers relatifs

- Autre idée : on aimerait que les opérations suivantes se fassent naturellement en binaire : $-1+1=0$, $-2+1=-1$, ...
  - Règle du complément à 2 sur 8 bits : on écrit $-x$ comme $256-x$ c'est à dire $255+1-x = \text{not}(x)+1$.
  - Autrement dit, pour convertir les entiers négatifs (dans les deux sens), on inverse les bits et on ajoute 1.

- Test : poser l'addition $-3+4$ sur 8 bits en complément à 2.



- Et le problème des 2 zéros ?

Précautions avec les négatifs

- Tout entier binaire a (au moins) deux **interprétations** : une valeur non signée et une valeur signée ! C'est la différence que l'on voit si l'on affiche avec `printf` et `%d` ou `%u` en C.
- En représentation signée, ajouter des 0 à gauche change la valeur du nombre !
- Un dépassement de capacité (*overflow*) **n'engendre pas forcément de carry**. Exemple avec l'addition $65 + 65$ sur 8 bits.
- Pour résumer : il y a dépassement de capacité si le résultat d'un calcul est en dehors de l'intervalle de représentation. Il y a donc **deux dépassements de capacité distincts** sur b bits :
  - signé : si le résultat est hors de $[0, 2^b[$,
  - non signé : s'il est hors de $[-2^{b-1}, 2^{b-1}[$

  Le processeur peut détecter les deux.

Les données en mémoire ne sont pas typées

L'information binaire est dépendante de l'interprétation qu'on en fait !
Observez, par exemple (fichier **demo-C1-0.c**) :

```
int a = 5; // essayer aussi avec -5
int *p = &a; // adresse de a
printf("%d, %u, %p \n", a, a, a);
printf("%d, %u, %p", p, p, p);
```

Par contre, les opérations $+$, $-$, $\times$ sont indépendantes de l'interprétation : on utilise le même circuit en signé / non signé. Par exemple, la valeur de $(*p)$ après les 2 codes suivants est identique (**demo-C1-1.c** et **demo-C1-2.c**) :

```
int a = -100, b = -200;
int c = a + b;

int* p = &c;
```

```
int a = -100, b = -200;
unsigned int c = (unsigned int)a
                 + (unsigned int)b;
int* p = (int*) &c;
```

Représentation des réels en machine

Un petit calcul facile

- Déclarer les flottants suivants (**demo-C1-3.c**)...

```
float a = 0.1, b = 0.2, c = 0.3;
```

  et tester `a + b == c`.

- Faire la même chose en `double`.
- Puis, pour les deux précisions, avec

```
a = 1.1, b = 1.2, c = 2.3;
```

- Cela devrait mettre en évidence que le calcul en nombres à virgule flottante (on dira simplement *flottants*) est imprécis et qu'il faut se méfier des tests d'égalité sur les flottants.

Un petit calcul facile

- Déclarer les flottants suivants (**demo-C1-3.c**)...

```
float a = 0.1, b = 0.2, c = 0.3;
```

  et tester `a + b == c`.

- Faire la même chose en `double`.
- Puis, pour les deux précisions, avec

```
a = 1.1, b = 1.2, c = 2.3;
```

- Cela devrait mettre en évidence que le calcul en nombres à virgule flottante (on dira simplement *flottants*) est imprécis et qu'il faut se méfier des tests d'égalité sur les flottants.

Avec, parfois, des conséquences... conséquentes (1)

- En 1991, à Dharan en Arabie Saoudite, un missile Patriot (anti-missile) américain rate un missile Scud irakien. Bilan : 28 morts et plus d'une centaine de blessés...

Extrait du rapport du *Government Accountability Office* :

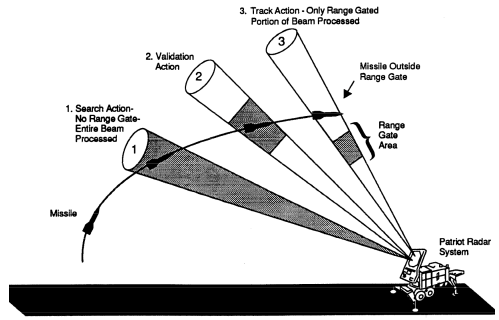
On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans. This report responds to your request that we review the facts associated with this incident and determine if a computer software problem was involved. If so, you asked that we provide information on what the specific software problem was, and what has been done to correct it. Appendix I details our objectives, scope, and methodology.

Results in Brief

The Patriot battery at Dhahran failed to track and intercept the Scud missile because of a software problem in the system's weapons control computer. This problem led to an inaccurate tracking calculation that became worse the longer the system operated. At the time of the incident, the battery had been operating continuously for over 100 hours. By then, the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud.

The Patriot had never before been used to defend against Scud missiles nor was it expected to operate continuously for long periods of time. Two weeks before the incident, Army officials received Israeli data indicating some loss in accuracy after the system had been running for 8 consecutive hours. Consequently, Army officials modified the software to improve the system's accuracy. However, the modified software did not reach Dhahran until February 26, 1991—the day after the Scud incident.

Incorrectly Calculated Range Gate



https://fr.wikipedia.org/wiki/MIM-104_Patriot

http://www-users.math.umn.edu/~arnold/disasters/patriot.html

Avec, parfois, des conséquences... conséquentes (2)

- En 1982 la bourse de Vancouver introduit un nouvel indice boursier (comme le CAC 40, le Dow Jones ou le Nasdaq, ...). Sa valeur initiale 1000.0, et il est actualisé environ 3000 fois par jour. À chaque fois, une partie du calcul est tronqué (et non pas arrondi)... Après 22 mois, la valeur de l'indice calculé est de 524,811... au lieu de 1098,892.

`https://en.wikipedia.org/wiki/Vancouver_Stock_Exchange#History`

Conversion des flottants en binaire

On traite la partie fractionnaire indépendamment de la partie entière :

binaire : $0, d_1 d_2 d_3 \dots d_n \leftrightarrow$ décimal : $\frac{d_1}{2} + \frac{d_2}{4} + \frac{d_3}{8} + \cdots + \frac{d_n}{2^n}$.

Pour passer du décimal au binaire, on peut utiliser l'algorithme suivant :

```
tant que la partie fractionnaire n'est pas nulle :
    la multiplier par 2
    garder la partie entière obtenue
    recommencer avec la nouvelle partie fractionnaire
```

Exemple pour $0,125$:

$$\begin{aligned} 0,125 \times 2 = 0,250 &\quad \rightarrow \quad \text{on garde 0 et on continue avec 0,250} \\ 0,25 \times 2 = 0,50 &\quad \rightarrow \quad \text{on garde 0 et on continue avec 0,50} \\ 0,5 \times 2 = 1,0 &\quad \rightarrow \quad \text{on garde 1 et on s'arrête parce qu'il reste 0} \end{aligned}$$

Résultat : $0,125 = 0,001_2$.

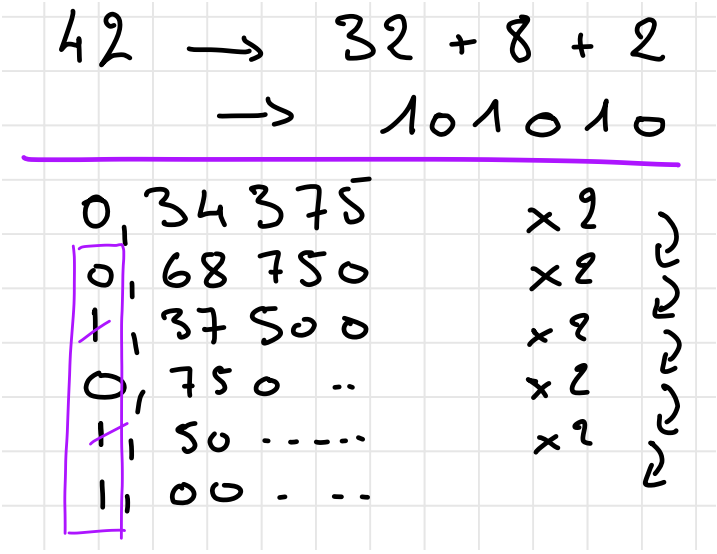
Exo : conversion

1. Convertir $42,34375_{10}$ en binaire.
2. Expliquer pourquoi cet algorithme fonctionne.

3. Convertir $0,1_{10}$ en binaire. Que remarque-t-on ?

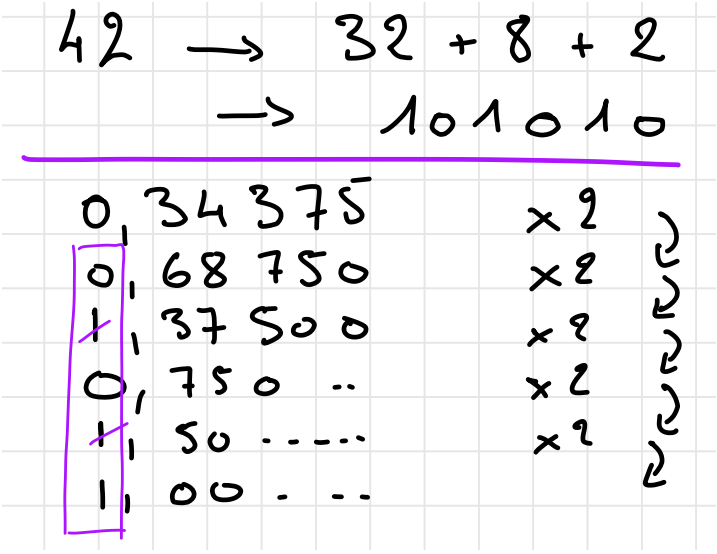
Exo : conversion

1. Convertir $42,34375_{10}$ en binaire. $\triangleright$ 101010,01011
2. Expliquer pourquoi cet algorithme fonctionne.
3. Convertir $0,1_{10}$ en binaire. Que remarque-t-on ?



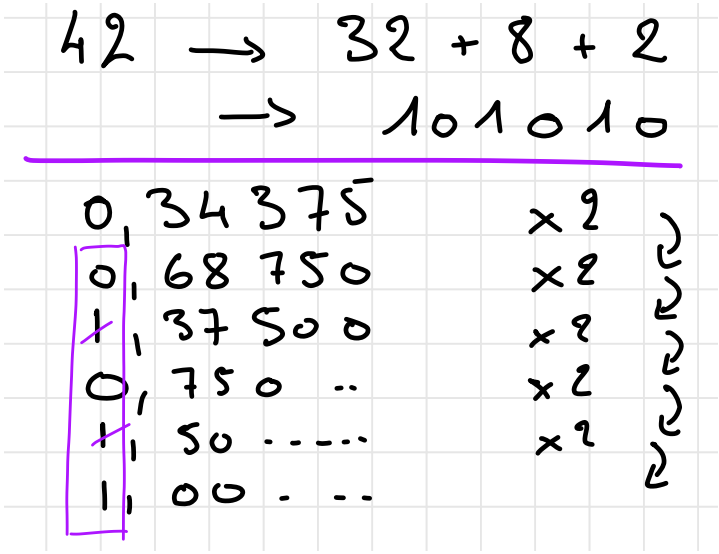
Exo : conversion

1. Convertir $42,34375_{10}$ en binaire. ▷ 101010,01011
2. Expliquer pourquoi cet algorithme fonctionne.
   ▷ la multiplication par 2 décale d'un bit vers la gauche
3. Convertir $0,1_{10}$ en binaire. Que remarque-t-on ?



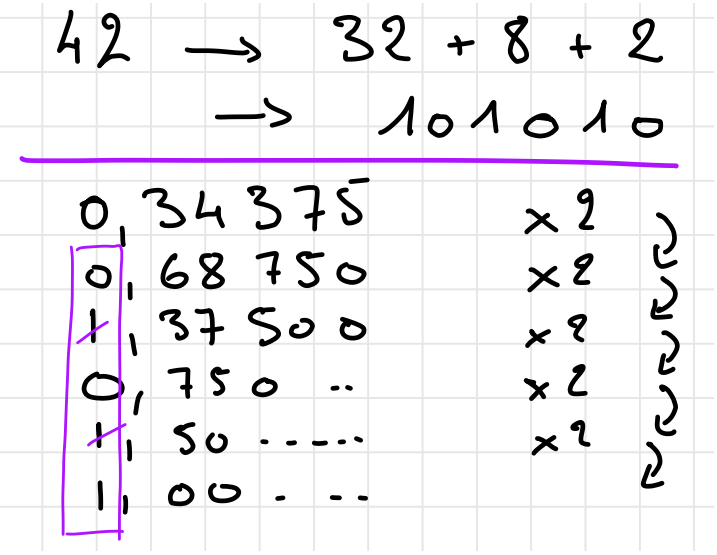
Exo : conversion

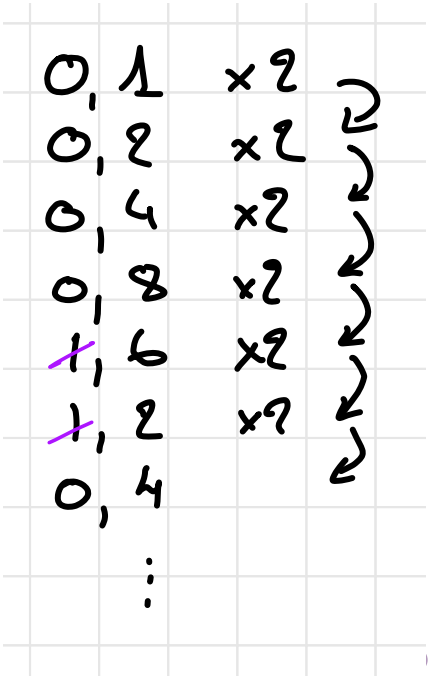
1. Convertir $42,34375_{10}$ en binaire. ▷ 101010,01011
2. Expliquer pourquoi cet algorithme fonctionne.
   ▷ la multiplication par 2 décale d'un bit vers la gauche
3. Convertir $0,1_{10}$ en binaire. Que remarque-t-on ?
   ▷ 0.0001100110011001100... boucle infinie !



Exo : conversion

1. Convertir $42,34375_{10}$ en binaire. ▷ 101010,01011
2. Expliquer pourquoi cet algorithme fonctionne.
   ▷ la multiplication par 2 décale d'un bit vers la gauche
3. Convertir $0,1_{10}$ en binaire. Que remarque-t-on ?
   ▷ 0.0001100110011001100... boucle infinie !





Virgule fixe vs. virgule flottante

On peut décider de représenter des nombres à virgule en consacrant par exemple k bits à la partie entière et ℓ bits à la partie fractionnaire ; il s'agit d'un codage à virgule fixe.

La notation scientifique décimale consiste à écrire les réels sous la forme : $\pm m \times 10^e$, de telle sorte que la partie entière de m soit comprise entre 1 et 9. De la même façon, on peut écrire les nombres binaires à virgule sous la forme $\pm 1, \ldots \times 2^e$.
On appelle e l'exposant et les chiffres après la virgule de m la mantisse.

Ici, on a un codage à virgule flottante car e n'est pas constant.

Voici deux exemples d'écriture scientifique en base 2 de nombres 16 bits :

$$\begin{aligned} (1000\ 0000)_2 &= 1 \times 2^{(111)_2} \\ (1000\ 0100\ 0010\ 0001)_2 &= 1.000\ 0100\ 0001 \times 2^{(1111)_2} \end{aligned}$$

Norme IEEE 754-2008

Quelle que soit la convention de codage, comme il y a 2^b mots binaires distincts sur b bits, on ne peut coder qu'au plus 2^b nombres à virgule distincts. Ces nombres sont dits *représentables* (par le codage choisi).

▷ développement de nombreux formats propriétaires

▷ problèmes de non-portabilité associés

L'*Institute of Electrical and Electronics Engineers* met au point une norme en 1985, qui est suivie d'une révision en 2008 pour l'étendre et la simplifier/préciser. En simple précision, on a :

- les nombres sont codés sur 32 bits,
- le premier bit correspond au signe,
- les 8 bits suivants codent l'*exposant* $E = e + 127$,
- les 23 bits restants codent la *mantisse* M, c'est-à-dire les chiffres après la virgule.

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

3. Et le complément à 2 ?

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

   ▷ 1 10000010 01000100000000000000000

2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

3. Et le complément à 2 ?

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

   ▷ 1 10000010 01000100000000000000000

2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

   ▷ 1 10000101 11011010100000000000000 ($e = 6$)

3. Et le complément à 2 ?

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

   ▷ 1 10000010 01000100000000000000000

2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

   ▷ 1 10000101 11011010100000000000000 ($e = 6$)
   ▷ $1,110110101$, avec la virgule décalée de 6 $\rightarrow 1110110,101$

3. Et le complément à 2 ?

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

   ▷ 1 10000010 01000100000000000000000

2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

   ▷ 1 10000101 11011010100000000000000 ($e = 6$)
   ▷ $1,110110101$, avec la virgule décalée de 6 $\rightarrow 1110110,101$
   ▷ -118,625

3. Et le complément à 2 ?

Exo : conversion avec la norme IEEE 754-2008

1. Écrire $-10,125$ en utilisant la norme IEEE 754-2008.

   ▷ 1 10000010 01000100000000000000000

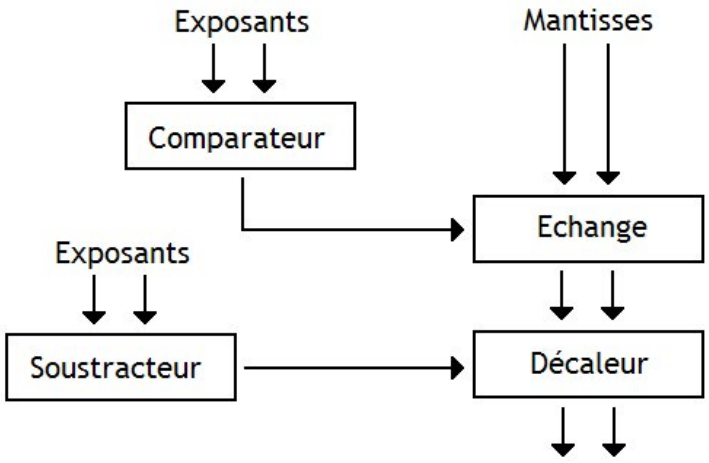
2. Donner la valeur décimale de nombre suivant codé avec la norme IEEE 754-2008 : 1100 0010 1110 1101 0100 0000 0000 0000.

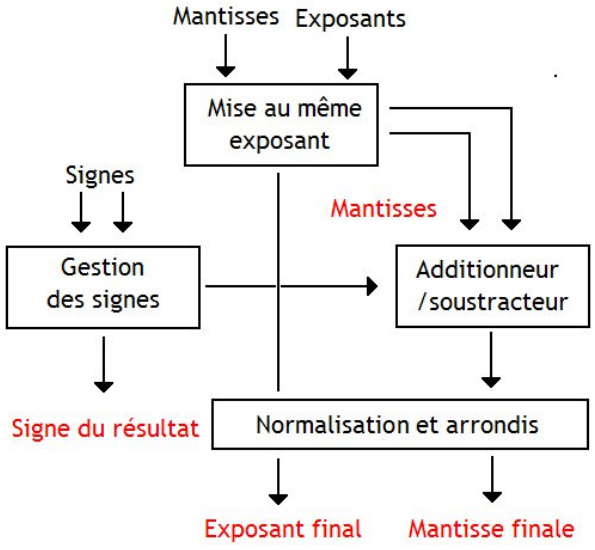
   ▷ 1 10000101 11011010100000000000000 ($e = 6$)
   ▷ $1,110110101$, avec la virgule décalée de 6 $\rightarrow 1110110,101$
   ▷ -118,625

3. Et le complément à 2 ?

On change de méthode de calcul

Pour faire une addition/soustraction, on doit commencer par se ramener au même exposant :





source : `https://fr.wikibooks.org/wiki/Fonctionnement_d%27un_ordinateur/Les_circuits_de_calcul_flottant`

▷ Unité de calcul spéciale : floating-point unit *FPU* (au lieu de ALU).

Que doit-on retenir de tout cela ?

- Le binaire stocké en mémoire n'est pas typé.
- On ne peut représenter qu'un nombre fini de valeurs (quel que soit leur type) en machine.
- Avec les entiers, on choisit en général un intervalle (et ce n'est pas possible avec les réels).
- Un réel dont l'écriture décimale est finie n'a pas forcément d'écriture binaire finie.
- Cela engendre des approximations...
- ... qui peuvent conduire à de grosses erreurs de calcul.

- En TP, nous étudierons la question de l'égalité des flottants.

(Et en bonus, nous parlerons de Quake III...)

Pour les curieux...

Software, Environments and Tools

Bits and Bugs: A Scientific and Historical Review on Software Failures in Computational Science

Author(s): **Thomas Huckle** and **Tobias Neckel**

Thomas Huckle
Technical University of Munich, Munich, Germany
Tobias Neckel
Technical University of Munich, Munich, Germany

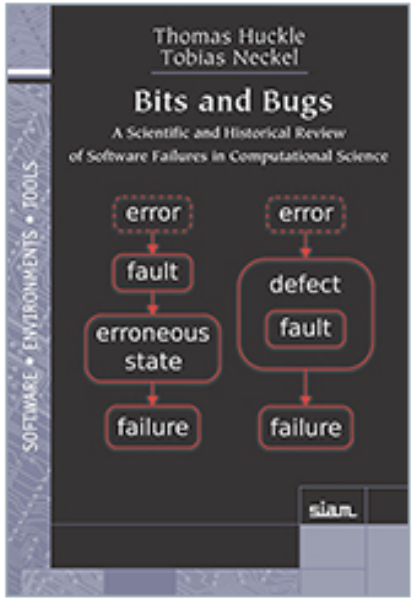
https://doi.org/10.1137/1.9781611975567

Permalink: https://doi.org/10.1137/1.9781611975567

Keywords: Software Errors, Software Failures, Computational Science, Scientific Computing, Numerical Methods

- Hide Description

In scientific computing (also known as computational science), advanced computing capabilities are used to solve complex problems. This self-contained book describes and analyzes reported software failures related to the major topics within scientific computing: mathematical modeling of phenomena; numerical analysis (number representation, rounding, conditioning); mathematical aspects and complexity of algorithms, systems, or software; concurrent computing (parallelization, scheduling, synchronization); and numerical data (such as input of data and design of control logic).



Title Information

Published: 2019

ISBN: 978-1-61197-555-0

eISBN: 978-1-61197-556-7