Arithmétique

23-30 janvier

À travers ce td, nous allons nous placer au niveau de l'ordinateur afin de voir comment il peut faire des calculs sur des entiers positifs et essayer de reproduire les calculs que peut faire une calculette. Le but du td est d'écrire d'implémenter des opérations élémentaires sur les entiers en ne se servant que de `for`, `if` et des comparaisons entre 0 et 1.

Sur un ordinateur, les nombres sont représentés en base 2. Dans la suite du TD, on représentera un entier par une liste de 32 nombres compris entre 0 et 1.

Exercice 1. Pour commencer

Avant de faire un calcul, on commence par préparer notre ordinateur.

a. Écrire une fonction `nouveau_calcul :=proc()` qui définit les variables globales suivantes :
- `overflow` à `false` : quand un processeur essaie de faire des opérations qui rendent des résultat trop grands, il s'en rend compte en modifiant une variable globale. Dans la suite du tp, on définira une variable globale `overflow` et à chaque fois qu'un calcul est impossible ou renvoie une erreur, il mettre cette valeur à `true`.
- `TAILLE` à 32, dans la suite du td, on utilisera cette variable plutôt que le nombre 32 afin de pouvoir facilement le modifier.
- `zero` au tableau correspondant à 0.
- `max_int` au plus grand entier que l'on peut écrire avec 32 chiffres.

b. Écrire une fonction `entier_to_binaire := proc(n)` qui prend en entrée un nombre entier et qui rend la liste de 0-1 correspondante.

c. Écrire une fonction `binaire_to_entier := proc(l)` qui prend en entrée une liste de 32 0-1 et qui rend le nombre correspondant (pensez à utiliser la commande `sum`). Si la variable `overflow` est à `true`, il affichera *le résultat est faux*

Exercice 2. L'addition

L'addition s'effectue comme l'addition que vous avez vu en primaire : on additionne les chiffres des unités puis des "deux-aines",... En n'oubliant pas les retenues.

a. Programmer une fonction `add :=proc(a,b)` qui effectue l'addition de deux nombres en n'utilisant pas l'addition de maple. On fera attention aux overflow.

b. Vérifier que votre fonction marche pour $1023+1, 16+18, 16844*468755$. Quand la variable `overflow` est-elle modifiée ?

Exercice 3. Tests

a. Écrire une fonction `egal := proc(a,b)` qui renvoie vrai si $a = b$

b. Écrire une fonction `plus_grand := proc(a,b)` qui renvoie vrai si $a > b$

Exercice 4. Multiplication, division par 2

La multiplication par une puissance de 2 correspond à un décalage vers la gauche du nombre.

a. Écrire une fonction `mult_puiss_2 := proc(n,k)` qui multiplie n par 2^k. Attention aux overflows.

b. Écrire une fonction `div_puiss_2 := proc(n,k)` qui rend le reste de la division euclidienne de n par 2^k

Exercice 5. Multiplication à la russe

Dans cet exercice, nous allons multiplier deux nombres entiers en utilisant l'addition mise en place à la partie précédente.

Si $b = \sum_i b_i 2^i$ où $b_i \in \{0; 1\}$, on a :

$$a * b = \sum_i b_i (2^i * a)$$

a. En se souvenant qu'on a défini une fonction qui multiplie un nombre par 2^k

b. Programmer la multiplication en se servant de l'addition précédente.

c. Tester votre fonction sur $37 * 19, 10 * 10, (101 + 5000 * 100) * 100000$ et comparer le résultat avec celui que calcule maple. Que vaut la valeur overflow à chaque fois ?

Exercice 6. Puissance d'un nombre

La puissance $n^{\text{ième}}$ d'un nombre se défini comme :

$$x^n = x * x^{n-1}$$

a. En déduire un algorithme récursif calculant la puissance d'un nombre en n'utilisant la multiplication et addition précédente.

b. Quelle est le nombre d'opérations effectuées par votre algorithme.

En utilisant les formules :

$$x^{2n} = (x^n)^2$$
$$x^{2n+1} = x(x^n)^2$$

c. En déduire un algorithme qui calcul x^n de façon plus efficace.

d. Programmer cet algorithme en maple. Combien d'opérations effectue-t-il ?

e. Calculer 3^6, 4^8 et 13^{16}. Comparer les résultats avec maple.

Exercice 7. Soustraction, division

a. Écrire une fonction `sous := proc(a,b)` qui rend `a-b`

Pour écrire une division (euclidienne) de a par b, on cherche le plus grand c tel que $bc \leq a$. Une méthode pour trouver c est donc de parcourir tous les $c \geq 0$ tels que $bc \leq a$.

b. Programmer cet algorithme.

Afin d'améliorer sensiblement cet algorithme, on peut chercher plus rapidement le résultat. Par exemple trouver le plus grand n_1 tel que $2^{n_1} b \leq a$ puis le plus grand n_2 tel que $(2^{n_2} + 2^{n_1})b \leq a$, jusqu'à trouver le résultat.

c. Programmer cet algorithme.

Exercice 8. Précision arbitraire ?

Normalement votre programme doit être capable de traiter des nombres avec une précision plus grande en remplaçant le 32 par un nombre plus grand.

a. Essayer en remplaçant 32 par 64 de calculer $(3^{15} + 2) * 66^3$

b. Quelle est l'impact sur la performance de remplacer 32 par 64 ?

c. Comment pourrait-on faire en maple pour gérer des nombres de taille arbitraire ? Programmer l'addition en précision arbitraire.