Algorithmique IV (UE-41) - TD 5.

TD 5. Calcul multiprécision [1]

EXERCICE 1. Soit k un entier naturel non-nul. On cherche à calculer le produit de deux entiers naturels A et B codés comme des entiers non-signés en machine sur des registres de 2^k bits.

(1) Quelle est l'entier *naturel* le plus grand que l'on peut représenter avec un registre de 64 bits ?

(2) Quelle est la condition sur les tailles de deux entiers A et B pour lesquels on peut réaliser l'opération de multiplication $A \cdot B$ en machine ?

On se propose d'écrire un algorithme qui calcule le produit $R := A \cdot B$ de deux entiers non-signés A et B de $n := 2^k$ bits, en segmentant les $2n$ bits du résultat R dans deux registres de n bits chacun.

Si X est un registre codé sur un nombre pair de bits, alors X_H (resp. X_L) désigne le registre contenant les bits de la première moitié « haute » (resp. de la deuxième moitié « basse ») de X. Donc $X = X_H|X_L$ si $|$ est l'opérateur de concaténation binaire, ou exprimé autrement

$$X = 2^p X_H + X_L$$

si X est codé sur $n = 2p$ bits.

Ce sont ces deux registres que nous allons calculer $R_H|R_L := A \cdot B$ pour dépasser la limitation du produit.

On dispose des algorithmes suivants dont les paramètres sont toujours codés sur n bits (exemples pour $k = 2$ et $n = 4$) :

- `MUL(X,Y)` : renvoie $X_L \cdot Y_L$.
  Exemple : $\texttt{Mul}(7, 10) = 6$ $((0111, 1010) \mapsto 0110)$.
- `ADD(X,Y)` : renvoie $X + Y$ en opérant sur les $n - 1$ bits faibles.
  Exemple : $\texttt{Add}(7, 13) = 12$ $((0111, 1101) \mapsto 1100)$.
- `HIGH(X)` : renvoie $0^{\frac{n}{2}}|X_H$.
  Exemple : $\texttt{H}(13) = 3$ $(1101 \mapsto 0011)$.
- `LOW(X)` : renvoie $0^{\frac{n}{2}}|X_L$.
  Exemple : $\texttt{L}(13) = 1$ $(1101 \mapsto 0001)$.
- `LOWLOW(X,Y)` : renvoie $X_L|Y_L$.
  Exemple : $\texttt{LOWLOW}(7, 9) = 13$ $((0111, 1001) \mapsto 1101)$.

Le principe est le suivant : on partitionne chaque opérande X en deux moitiés X_H et X_L sur $\frac{n}{2}$ bits :

$$\begin{aligned} A \cdot B &= (A_H 2^{\frac{n}{2}} + A_L)(B_H 2^{\frac{n}{2}} + B_L) \\ &= A_H B_H \, 2^n + A_H B_L \, 2^{\frac{n}{2}} + A_L B_H \, 2^{\frac{n}{2}} + A_L B_L. \end{aligned} \tag{1}$$

Ces quatre produits sont calculables sans débordement par l'algorithme `MUL(X,Y)`, il reste à déterminer comment les exploiter pour déterminer le contenu des deux registres R_H et R_L codés sur n bits chacun.

(3) Écrivez les algorithmes `LOW(X)`, `HIGH(X)` et `LOWLOW(X,Y)` à l'aide des opérateurs logiques bit à bit.

(4) Pour $k = 2$, c'est-à-dire quand les opérandes A et B sont codés sur $n = 4$ bits, représentez les contenus des 4 produits (cf. (1)) pour $A = B = 2^n - 1$ ainsi que ceux des registres R_H et R_L en binaire.

(5) Dressez une table des valeurs binaires des 4 termes de la somme (1) pour mettre en évidence le calcul à réaliser pour obtenir le contenu des registres R_H et R_L contenant respectivement la moitié haute et la moitié basse du produit $A\,B$. Généralisez à des opérandes A et B codés sur n bits pour écrire un algorithme qui calcule R_H et R_L.

Solution. **(1)** Les entiers naturels codés sur n bits prennent leurs valeurs dans l'intervalle $[\![0, 2^n - 1]\!]$, donc le plus grand est l'entier

$$2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615. \tag{2}$$

(2) Il faut que le produit soit inférieur ou égal à la valeur (2), ou encore que le nombre de chiffres de $A\,B$ soit inférieur à 64. On sait (cf. planche précédente) que le nombre de chiffres en base b d'un entier N est donné par

$$\lfloor \log_2(N) \rfloor + 1,$$

on en déduit donc l'inéquation

$$\lfloor \log_2(A\,B) \rfloor + 1 \leqslant 64 \Leftrightarrow \lfloor \log_2(A) + \log_2(B) \rfloor \leqslant 63.$$

Plutôt que laisser la taille des nombres A et B varier suivant cette contrainte, on fixe généralement leur taille à une valeur inférieure ou égale à $2^{32} - 1$.

1. Version du 14 mars 2025, 14 : 04

Remarque. Les processeurs modernes savent multiplier deux nombres de 64 bits, en codant, comme nous venons de le faire ici, le résultat dans deux registres. Celui de poids faible est renvoyé par le calcul mais celui de poids fort n'est accessible que via un code en assembleur. La fonction `produit128` ci-dessous renvoie le même résultat que le produit `A*B` mais passe la valeur du registre de poids fort via la variable `H`. Les registres machine utilisés ici sont ceux d'une architecture type `x86-64` :

```
uint64_t produit128(uint64_t A, uint64_t B, uint64_t *H) {
    uint64_t L;
    asm ("mulq %2" : "=d" (*H), "=A" (L) : "A" (A), "rm" (B));
    return L;
}
```

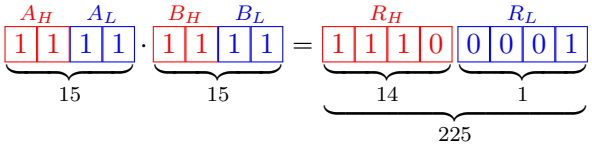
(3) On se contente de décrire les opérations bit à bit à réaliser, l'algorithme se limitant à renvoyer la valeur ainsi calculée :

(1) `HIGH(X)` : `X` $\gg (n \gg 1)$. On décale les bits de X de $\frac{n}{2}$ positions vers la droite.

(2) `LOW(X)` : (`X` $\ll (n \gg 1)) \gg (n \gg 1)$. On décale les bits de X de $\frac{n}{2}$ bits à gauche puis à droite.

(3) `LOWLOW(X,Y)` : (`LOW(X)` $\ll (n \gg 1)) \vee$ `LOW(Y)`. On décale les bits de X de $\frac{n}{2}$ positions vers la gauche et on rajoute les $\frac{n}{2}$ de poids faible de Y.

(4) On a $A = B = 15$, ce qui nous donne $A_H = A_L = B_H = B_L = 3$ et ainsi les 4 produits sont tous égaux à 9, soit 1001 en binaire. On résume les différents calculs ci-dessous :



(5) Il faut réaliser quelques additions sur des demi-registres de $\frac{n}{2}$ bits pour obtenir successivement les demi-registres de poids faibles et de poids forts de chacun des registres R_L et R_H. L'équation (1) nous indique comment construire la table 1.

	retenues	0	1	0	1				
AHBH	$A_H B_H\, 2^n$	1	0	0	1				
AHBL	$A_H B_L\, 2^{\frac{n}{2}}$			1	0	0	1		
ALBH	$A_L B_H\, 2^{\frac{n}{2}}$			1	0	0	1		
ALBL	$A_L B_L$					1	0	0	1
	+	1	1	1	0	0	0	0	1
	$A \cdot B$	R_H				R_L			

TABLE 1. Contenus des différents registres.

Les deux registres R_H et R_L sont segmentés en 4 demi-registres de $\frac{n}{2}$ bits, matérialisés par 4 couleurs dans la table. Les contenus de ces 4 demi-registres sont obtenus en additionnant certains des 8 demi-registres des 4 produits $A_X B_Y$ qui sont partitionnés en conséquence dans la table 1 suivant les couleurs correspondantes. On commence le calcul par la droite pout obtenir le contenu du demi-registre de poids faible de R_L. C'est tout simplement le demi-registre de poids faible du produit $A_L B_L$, il n'y a donc pas de retenue. On construit ensuite le demi-registre de poids fort de R_L puis on fait de même avec le demi-registre de poids faible de R_H et on termine avec son demi-registre de poids fort. L'algorithme s'en déduit directement.

Les variables auxiliaires `H` et `L` servent dans un premier temps à construire respectivement le demi-registre de poids fort et le demi-registre de poids faible du registre R_L puis ceux du registre R_H. Notons qu'après la première addition la partie gauche de cette variable contient la retenue éventuelle.

```
ALGORITHME MMUL (A,B):couple d'entiers
DONNEES
 · A,B: entiers
VARIABLES
 · AHBH,AHBL,ALBH,ALBL,RH,RL,H,L: entiers
DEBUT
 · // INITIALISATIONS:
 · AHBH ← MUL(H(A), H(B))
 · AHBL ← MUL(H(A), L(B))
 · ALBH ← MUL(L(A), H(B))
 · ALBL ← MUL(L(A), L(B))
 · // CONSTRUCTION RL:
```

```
  · L ← LOW(ALBL)
  · H ← ADD(LOW(AHBL), ADD(LOW(ALBH), HIGH(ALBL)))
  · RL ← LOWLOW(H, L)
  · // CONSTRUCTION RH:
  . L ← HIGH(H) // RETENUE DE LA L'ADDITION
  · L ← ADD(ADD(ADD(L, LOW(AHBH)), HIGH(AHBL)), HIGH(ALBH))
  . H ← HIGH(L) // RETENUE DE L'ADDITION
  · H ← ADD(H, HIGH(AHBH))
  · RH ← LOWLOW(H, L)
  · RENVOYER (RH, RL)
FIN
```

EXERCICE 2. Soit n un entier naturel. On veut estimer le nombre de chiffres nécessaires pour représenter n en base 2 (asymptotiquement).

(1) À l'aide de la formule de Stirling ci-dessous :

$$\lim_{n\to+\infty} \frac{n!}{\sqrt{2\pi n}(n/e)^n} = 1, \tag{3}$$

donnez une estimation asymptotique du nombre de chiffres nécessaires pour représenter la factorielle $n!$ d'un entier naturel n en base 2.

(2) Même question, mais en comparant avec une somme intégrale.

Solution. (1) On rappelle que le nombre de chiffres nécessaires pour représenter un nombre n en base b est égal à $\lfloor \log_b n \rfloor + 1$. La fonction logarithme est continue sur $\mathbb{R}^+$ on peut donc commuter avec la limite :

$$\lim_{n\to+\infty} \log_2 \left(\frac{n!}{\sqrt{2\pi n}(n/e)^n} \right) = 0$$

$$\text{donc} \quad \lim_{n\to+\infty} \log_2(n!) = \lim_{n\to+\infty} \underbrace{\log_2 \left(\sqrt{2\pi n}(n/e)^n \right)}_{L}$$

Calculons l'expression L à droite de la dernière égalité :

$$\begin{aligned} L &= \log_2(\sqrt{2\pi n}) + n \log_2 \left(\frac{n}{e}\right) \\ &= \frac{1}{2} \left(\log_2(2) + \log_2(\pi) + \log_2(n)\right) + n \left(\log_2(n) - \log_2(e)\right) \\ &= \underbrace{n \log_2(n)}_{f(n)} - \underbrace{\left(\log_2(e) n - \frac{\log_2(n)}{2} - \frac{1 + \log_2(\pi))}{2} \right)}_{g(n)} \end{aligned}$$

On vérifie aisément que

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

On en conclut que L a le même comportement asymptotique que la fonction définie par $n \mapsto n \log_2(n)$. Le nombre de chiffres de $n!$ exprimé en base 2 est donc asymptotiquement égal à $n \log_2(n)$.

(2) Cette fois on écrit directement :

$$\log_2(n!) = \sum_{k=2}^{n} \log_2(k) = \frac{1}{\ln(2)} \sum_{i=2}^{n} \ln(i).$$

Or on peut encadrer la somme par une intégrale :

$$\int_1^n \ln(x)\mathrm{d}x \leqslant \sum_{i=2}^{n} \ln(i) \leqslant \int_2^{n+1} \ln(x)\mathrm{d}x.$$

On rappelle qu'une primitive de $\ln(x)$ est la fonction $x \ln(x) - x$ (sinon faire une intégration par partie avec $u'(x) = 1$ et $v(x) = \ln(x)$) :

$$[x \ln(x) - x]_1^n \leqslant \sum_{i=2}^{n} \ln(i) \leqslant [x \ln(x) - x]_2^{n+1}$$

$$n \ln(n) + \underbrace{(1-n)}_{a(n)} \leqslant \sum_{i=2}^{n} \ln(i) \leqslant n \ln(n+1) + \underbrace{\left(\ln(n+1) - (n+1) - 2\ln(2)\right)}_{b(n)}$$

Là encore, les fonctions $a(n)$ et $b(n)$ sont négligeables asymptotiquement par rapport aux fonctions $n \ln(n)$ et $n \ln(n+1)$ et bien sûr $n \ln(n) \sim n \ln(n+1)$. On en conclut que le nombre de chiffres de $n!$ exprimé en base 2 est asymptotiquement égal à $n \log_2(n)$.

EXERCICE 3. Écrivez un algorithme qui réalise la soustraction $A - B$ de deux entiers naturels en multiprécision et en supposant que $A \geqslant B$. Calculez sa complexité dans le meilleur des cas et dans le pire des cas.

Solution. On considère que A et B sont deux listes contenant les chiffres de l'écriture dans une base donnée des entiers correspondant, l'indexation commençant à 0 et correspondant au chiffre le *moins* significatif. Il faut initialiser une liste résultat R à la taille de la liste A.

```
ALGORITHME SOUSTRACTION(A,B,base):liste
DONNÉES
 · A,B: listes
 · base: entier
VARIABLES
 · i,retenue: entiers
 · R: liste
DEBUT
 · R ← Allouer(#A,0) // cellules initialisees a 0
 · retenue ← 0
 · i ← 0
 · TQ (i < #B) FAIRE
 ·  · SI ((B[i] + retenue) <= A[i]) ALORS
 ·  ·  · R[i] ← A[i] - (B[i] + retenue)
 ·  ·  · retenue ← 0
 ·  · SINON
 ·  ·  · R[i] ← (A[i] + base) - (B[i] + retenue)
 ·  ·  · retenue ← 1
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · TQ (i < #A) FAIRE
 ·  · SI (retenue <= A[i]) ALORS
 ·  ·  · R[i] ← A[i] - retenue
 ·  ·  · retenue ← 0
 ·  · SINON
 ·  ·  · R[i] ← (A[i] + base) - retenue
 ·  ·  · retenue ← 1
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · RENVOYER R
FIN
```

Si l'on note $n = |A|$ et $m = |B|$, la complexité dans le meilleur des cas est en $\Theta(m)$ puisque le nombre d'opérations est proportionnel à la longueur de B et dans le pire des cas en $\Theta(n)$ puisque le nombre d'opérations est alors proportionnel à la longueur de A.

EXERCICE 4. On considère deux entiers A et B de n chiffres en base b et on note $\ell := \frac{n}{2}$. On scinde A (resp. B) à l'aide de deux nombres de ℓ chiffres A_H et A_L (resp. B_H et B_L) :

$$A = A_H\, b^\ell + A_L \quad \text{et} \quad B = B_H\, b^\ell + B_L.$$

(1) Calculez les produits $A\,B$ et $(A_H + A_L)(B_H + B_L)$.

(2) À partir des résultats de la question (1), vérifiez que

$$A\,B = A_H B_H b^n + \big((A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L)\big) b^\ell + A_L B_L \quad (4)$$

(3) En supposant que le nombre de multiplications pour calculer le produit de deux nombres à n chiffres est $P(n)$ et que le produit de deux nombres à un chiffre a un coût constant de $\Theta(1)$ (c'est une recherche en table), exprimez la fonction P avec une relation de récurrence. Pour simplifier les calculs, on suppose que n est une puissance de 2. Indication : utilisez l'égalité (4).

(4) Montrez que $P(n) = \Theta(n^{\log_2(3)})$.

Solution. **(1)** On calcule les produits :

$$\begin{aligned} A\,B &= (A_H\, b^\ell + A_L)(B_H\, b^\ell + B_L) \\ &= A_H B_H\, b^n + (A_H B_L + A_L B_H) b^l + A_L B_L. \end{aligned}$$

$$\text{Et} \quad (A_H + A_L)(B_H + B_L) = A_H B_H + (A_H B_L + A_L B_H) + A_L B_L$$

$$\text{donc} \quad (A_H B_L + A_L B_H) = (A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L).$$

(2) Par conséquent

$$A\,B = A_H B_H b^n + \big((A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L)\big) b^\ell + A_L B_L$$

ce qui permet de calculer le produit de deux nombres à n chiffres en effectuant uniquement 3 produits de nombres à $\frac{n}{2}$ chiffres (et 5 additions que l'on néglige).

(3) On déduit directement de l'égalité (4) :

$$P(n) = \begin{cases} 3P(\frac{n}{2}) & \text{si } n > 0 \\ \Theta(1) & \text{sinon} \end{cases}$$

(4) On obtient facilement $P(n) = 3^k\Theta(1) = \Theta(3^k)$ si $n = 2^k$. Mais $k = \log_2(n)$, donc

$$\begin{aligned} P(n) &= \Theta(3^{\log_2(n)}) \\ &= \Theta\left(\left(2^{\log_2(3)}\right)^{\log_2(n)}\right) \\ &= \Theta\left(\left(2^{\log_2(n)}\right)^{\log_2(3)}\right) \\ &= \Theta\left(n^{\log_2(3)}\right). \end{aligned}$$

Comme $\log_2(3) \simeq 1{,}585$, cet algorithme a une complexité proche de $n\sqrt{n}$ ce qui améliore notablement le temps de calcul d'une multiplication par rapport à l'algorithme naïf. Il a été conçu par Anatolli Karatsuba en 1960 et porte son nom.