Projet d'architecture 2 : implémentation d'un diviseur entier performant

L'objectif de ce mini-projet est d'implémenter un diviseur entier naturel performant. La division présentée en cours, ci-dessous, est simple mais très peu performante. Dans le pire des cas (avec des mots de 32-bits), 2^{32}-1 divisé par 1, il faudra effectuer 2^{32}-1 itérations.

```
@ X = Q × Y + R
R ← X
Q ← 0
Tantque R ≥ Y Faire
      R ← R – Y
      Q ← Q + 1
FinTantQue
```

Pour obtenir un algorithme alternatif (et plus performant), on peut s'inspirer de la méthode utilisée pour réaliser des divisions en colonnes (tel que appris au primaire). Un exemple en base 10 et en binaire est donné ci-dessous :

	2	5	3	1	3
-	1	3		1	9=q
	1	2			
	1	2	3		
-	1	1	7		
r=	0	0	6		

	1	1	1	1	1	1	0	1	1 1 0 1
-	1	1	0	1					
	0	0	1	0					1 0 0 0 0
			1	0	1	1	0		
-				1	1	0	1		0 0 0 1 0
			0	1	0	0	1		
				1	0	0	1	1	
-					1	1	0	1	0 0 0 0 1
r=				0	0	1	1	0	
									1 0 0 1 1

L'algorithme est itératif, c'est-à-dire que le calcul se répète pour obtenir à chaque étape un chiffre C_i supplémentaire du quotient Q, du poids fort au poids faible. En base 10, le calcul débute par la détermination de n choisi tel que $Y \times 10^{n-1} \leq X < Y \times 10^{n}$. On pose alors $R_n = X$. L'itération *i* (i variant de n-1 à 0) permet de calculer le chiffre C_i et le reste R_i (qui sera utilisé à l'itération suivante) ; elle est réalisée en deux étapes :

- Le chiffre C_i de rang courant est obtenu grâce à la formule $C_i \times Y \times 10^{i} \leq R_{i+1} \leq (C_i + 1) \times Y \times 10^{i}$.
- On produit le nouveau R_i grâce à $R_i = R_{i+1} - C_i \times Y \times 10^{i}$.

On peut remarquer que, dans le cas binaire, il y aura de nombres simplifications :

- C_i peut seulement être 0 ou 1,

- par conséquent, $C_i \times Y \times 2^i$ s'évalue simplement en 0 ou $Y \times 2^i$,
- enfin, la différence à calculer devient en base 2, $R_{i+1} - C_i \times Y \times 2^i$, c'est-à-dire qu'il sera possible d'utiliser des décalages plutôt qu'une multiplication (beaucoup moins performante).

Enfin, il est facile de voir que les performances ont été améliorées. Le pire des cas est toujours 2^{32}-1 divisé par 1 mais sera réalisé seulement en 32 itérations !

Travail à réaliser

L'objectif du projet est de réaliser une fonction de division performante en assembleur ARM. Cette fonction suivra le protocole d'appel suivant :

```
@ implémentation performante de la division X = Q Y + R
@ si Y = 0, branchement sur le sous-programme div_by_0
@ à fournir par l'utilisateur de cette fonction
@     r0 (en entrée) = X (dividende)
@     r1 (en entrée) = Y (diviseur)
@     r0 (en sortie) = R (reste)
@     r1 (en sortie) = Q (quotient)
fast_div :
```

1. Réaliser la fonction `fast_div` en utilisant l'algorithme de la division en colonne.
2. Améliorer cette fonction en remarquant que, quand on divise par une puissance de 2, $Y = 2^n$, il suffit alors de faire un décalage à droite de *n* positions. Pour savoir, de manière performante, si un nombre est une puissance de 2, on pourra utiliser une table à 256 entrées donnant, pour chaque valeur entre 0 et 255, *n* si la valeur est une puissance de 2 (2^n), 32 sinon. On pourra tester chaque octet du mot Y du poids fort au poids faible.
3. Écrire un programme principal qui permet d'exécuter les jeux de tests suivants :

X	***Y***	***Q***	***R***
13	3	4	1
250	50	5	0
5	24	0	5
1000	1	1000	0
2048	8	256	0
65536	512	128	0
1000	0	appel de div_by_0	