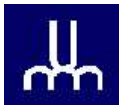

DIRO
IFT 1215

INTRODUCTION AUX SYSTÈMES INFORMATIQUES
DEVOIR Nº 3

Max Mignotte

DIRO, Département d'Informatique et de Recherche Opérationnelle
Http : //www.iro.umontreal.ca/∼mignotte/ift1214/
E-mail : mignotte@iro.umontreal.ca

Programmation en Langage Machine

Dans ce devoir, on se propose d'implémenter deux routines en langage machine (avec un jeux d'instruction réduit) pour extraire la racine carrée d'un nombre tel que le ferait un calculateur moderne. La première méthode nous est donnée par Newton et la seconde à été proposé par Wilco Dijkstra.

I. Méthode de Newton

Le but de cette procédure est de rechercher la racine du nombre x, c'est à dire de trouver $y = \sqrt{x}$. La méthode proposée par Newton propose une solution numérique à cette recherche et plus précisément à la résolution de l'équation $y^2 = x$. Le raisonnement proposé par Newton fut le suivant ;

Supposons que l'on ait une première approximation pour cette racine, et appelons y_n, cette première approximation. Cherchons maintenant un incrément δ qui ajouter à cette première approximation (i.e., $y_n + \delta$) permettrait d'obtenir la solution, c'est à dire $(y_n + \delta)^2 = x$. On a,

$$(y_n + \delta)^2 = y_n^2 + 2y_n\delta + \delta^2 = x$$

Si on néglige le terme δ^2 (ce qui est raisonnable quant on compare ce terme à y_n^2 et $2y_n\delta$), on peut écrire $y_n^2 + 2y_n\delta \approx x$ et finalement $\delta \approx \frac{x-y_n^2}{2y_n}$. On se retrouve donc avec l'approche itérative suivante,

$$\begin{aligned} y_{n+1} &= y_n + \delta, \\ &= y_n + \frac{x - y_n^2}{2y_n}, \end{aligned}$$

dans laquelle y_{n+1} est l'estimée de la racine à la $n+1$ ième itération et y_n est l'estimée de la racine à l'itération précédente (c'est à dire à l'itération n). La solution de cette relation itérative est appelée un point fixe.

En exemple, supposons que l'on recherche la racine de $x = 12$ et que l'on commence la relation itérative par $y_0 = 1$, on obtiendrait la suite des estimations suivantes,

$$\begin{aligned} y_0 &= 1, \\ y_1 &= 6.5, \\ y_2 &= 4.173076923, \\ y_3 &= 3.52432648, \\ y_4 &= 3.464616186, \\ y_5 &= 3.464101653, \\ y_6 &= 3.464101615, \\ y_7 &= 3.464101615, \\ &\vdots \end{aligned}$$

qui converge très rapidement vers la valeur $\sqrt{12} = 3.464101615$.

Implémenter en langage machine avec le jeux d'instruction donnée plus bas cette méthode permettant d'extraire la racine d'un nombre. On supposera que la valeur dont on désire extraire la racine est entrée au début du programme par l'instruction IN, que l'on commence toujours par $y_0 = 1$, que la procédure itérative s'arrête à l'itération n lorsque $y_{n+1} - y_n = 0$ (dans la réalité c'est un plus complexe que cela) et que l'accumulateur de notre modèle ainsi que nos code arithmétique peuvent traiter des nombres non entiers.

II. Méthode de Dijkstra

Le but de cette méthode est de rechercher la partie entière de la racine du nombre entier x (ex : la partie entière de la racine du nombre 325 est 18). L'approche proposée par Dijkstra est résumé dans le pseudo code suivant,

Algorithme de Dijkstra

x Calcul de la partie entière de $\sqrt{x}$

1. Initialisation
$y \leftarrow 0$ $\quad \delta \leftarrow 2^{15}$

2. Itération

while $(\delta \neq 0)$ **do**
  if $(y+\delta)^2 \leq x$ ▶ $y \leftarrow y + \delta$
  else $\delta \leftarrow \delta/2$

A la fin de la boucle WHILE, y contient la partie entière de la racine carrée de x. La division par 2 de l'entier δ qui doit être fait à chaque itération de la boucle est fait très rapidement par la machine par un décalage à droite.

Implémenter en langage machine avec le jeux d'instruction donnée plus bas cette méthode.

Jeux d'Instructions Machine

On utilisera le jeux d'instruction présenté dans le chapitre LITTLE MAN COMPUTER (Cf. Fig. 1) et auquel on ajoutera les deux instructions suivantes :

1. **MUL** xx Multiplie le contenu de la case mémoire xx au nombre stocké dans l'accumulateur (registre du calculateur). Le résultat de la multiplication est stocké dans l'accumulateur.
2. **DIV** xx Divise le contenu de la case mémoire xx au nombre stocké dans l'accumulateur (registre du calculateur). Le résultat de la division est stocké dans l'accumulateur.

LDA	5xx	Load
STO	3xx	Store
ADD	1xx	Add
SUB	2xx	Subtract
IN	901	Input
OUT	902	Output
COB or HLT	000	Coffee break (or Halt)
BRZ	7xx	Branch if zero
BRP	8xx	Branch if positive or zero
BR	6xx	Branch unconditional
DAT		Data storage location

FIG. 1 – Jeux d'instructions utilisé pour élaborer le programme auquel on doit rajouter l'instruction **MUL** et **DIV** définie dans l'énoncé.

Remise & Rapport

Ce rapport devra être fait par groupe de deux étudiants. Vous devez rendre ce travail au démonstrateur le jour de la démonstration, i.e., le vendredi à $10h30$, à la date de remise spécifiée dans le fichier *planning* situé sur la page web du cours dans la rubrique *Introduction & Programme*. Vous pouvez écrire votre rapport à la main ou de préférence par traitement de texte. Dans les deux cas, le rapport devra être clair, présentable et concis (il devra toutefois contenir les résultats intermédiaires nécessaires permettant de montrer sans ambiguïté que vous êtes arrivés au résultat demandé). Il y aura une pénalité de 20% par jour de retard.