RACINES CARRÉES ET DISTANCES SUR µC

La fonction racine carrée est une des fonctions fondamentales et est nécessaire dans de nombreux calculs.

On l'utilisera par exemple pour calculer ArcSin(x) (1) ou ArcCos(x) (2) à partir de la fonction ArcTan(x) déjà décrite dans ces colonnes, pour calculer des distances euclidiennes ou dans tout autre calcul.

L'algorithme le plus connu de calcul de racine carré est la méthode de NEWTON qui peut se résumer de la manière suivante :

$r = (x + r^2)/(2.r)$

```
typedef union{                          /* long_int est soit */
        unsigned long   l;              /* un long (32 bits) */
        unsigned int    i[2];           /* soit deux int de 16 bits */
        }long_int;
#define MSB 1                           /* dépend de la machine */

/* calcul de racine carrée d'une valeur entière */
unsigned char square(unsigned int value)
{
        long_int        valeur;         /* valeur entrée */
        unsigned int    résultat;       /* résultat intermédiaire */
        unsigned int    temporaire;
        unsigned int    loop;           /* compteur de boucle */

        valeur.l = (unsigned long)value;/* initialise la valeur */
        résultat = 0;                   /* et le résultat */
        for(loop = 0; loop < 8; loop++){
          résultat <<= 1;               /* pousse le résultat précédant */
          valeur.l <<= 2;               /* prend les deux bits MSB */
          temporaire = (résultat*2) + 1;
          if(valeur.i[MSB] >=temporaire){ /* teste la racine carrée */
            valeur.i[MSB] -=temporaire; /* recadre le MSB */
            résultat++;                 /* racine carrée = 1 */
            }
          }
        return résultat;
}
```

■ *Listing 2 : la version C*

L'idée est d'essayer une racine (r) de la valeur dont on désire extraire la racine (x), puis de réintroduire la nouvelle valeur de r trouvée. En réitérant l'opération plusieurs fois, la valeur r convergera rapidement vers la valeur désirée. Par exemple, calculons la racine carrée de 100 :
x = 100, on prend comme valeur approchée de la racine la valeur 100 (pas une très bonne approximation)
r = (100 + (100 X 100)) / (2 X 100) = 50,50
r = (100 + (50,50 X 50,50)) / (2 X 50,50) = 26,24
r = (100 + (26,24 X 26,24)) / (2 X 26,24) = 15,03
r = ... = 10,84
r = ... = 10,03
r = ... = 10,00001
r = ... = 10,00000
Chaque nouveau calcul nous rapproche de la solution on pourra donc théoriquement arrêter les itérations quand on obtiendra la précision souhaitée. En fait si on effectue les calculs en flottant, les erreurs de troncature sur les calculs flottants limiteront rapidement la précision du résultat. Il sera prudent de limiter le nombre d'itérations pour éviter d'osciller indéfiniment autour de la solution si la précision souhaitée est importante.
De même un bon choix de la racine de départ diminuera sensiblement le nombre d'itérations à effectuer. On pourra aussi réécrire l'équation :
r = (r + x / r) / 2 pour limiter les débordements dans les calculs intermédiaires.
Cet algorithme universellement employé pour les calculs en flottant n'est pas optimum pour des calculs entiers. Il oblige à fixer le nombre d'itérations pour éviter l'oscillation du résultat. De plus sur les processeurs qui ne disposent pas de division câblée suffisante, (ce qui est le cas des microprocesseurs 80C5X, 68HC11 et 68HC05), cet algorithme est inutilement lent.
Il existe un autre algorithme très simple pour extraire les racines carrées qui ne nécessite pas de division (ni de multiplication).
Considérons le **tableau 1**.
Dans la dernière colonne, la différence de deux carrés successifs est la suite des nombres impairs. On peut donc écrire l'algorithme suivant :

n	n^2	différence
0	0	
1	1	1
2	4	3
3	9	5
4	16	7
5	25	9

■ *Tableau 1*

```
r = 1;
do{
        x = x - r;
        r = r + 2;
        }
while( x >= 0 );
r = (r / 2) - 1;
```

C'est vraisemblablement l'algorithme le plus simple de calcul de racine carrée.

Petit problème : le nombre d'itérations est égal à la valeur de la racine carrée. Fort heureusement, on peut limiter les itérations si on se limite à des calculs de petites racines carrées. Quand on réalise une racine carrée à la main, on groupe les chiffres deux par deux, et on calcule la racine carrée deux chiffres par deux chiffres. De même, en base deux, on pourra grouper les bits deux par deux et chercher la racine carrée de ces deux bits (0 ou 1), exactement comme dans la méthode manuelle. Le nombre d'itérations n'est donc plus que de un, en revanche, il faudra effectuer n calculs de racines carrées, n étant le nombre de bits du résultat. Le **listing 1** réalise le calcul de la racine carrée d'une valeur 32 bits contenue dans ACCU32 et restitue le résultat dans R6:R7 en assembleur 80C31. La durée d'exécution est au maximum de 1,5 millisecondes pour un microprocesseur cadencé à 12 MHz. Le **listing 2** est une implémentation du même algorithme en langage C.

Le calcul de racine carrée est souvent utilisé pour évaluer des distances. La distance de deux points de coordonnées (x_1, y_1) et (x_2, y_2) est :

$d = racine(X^2 + Y^2)$ avec $X = x_1 - x_2$ et $Y = y_1 - y_2$

Dans les cas où il n'est pas nécessaire de faire un calcul avec une grande précision, on peut utiliser les approximations (4) et (5) pour évaluer la distance. Max(x, y) et Min(x, y) sont respectivement des fonctions qui renvoient les valeurs maximum et minimum de x et de y. |x| est la valeur absolue de x.

L'approximation (4) donne un résultat avec une erreur minimum de 0 et une erreur maximum de 11,76 % soit une erreur inférieure à 1 dB.

Si cette approximation est insuffisante, on utilisera la formule (5). L'erreur minimum est ici de -2,77 % et l'erreur maximum de 0,78 %. Ces deux approximations pourront être très facilement programmées en n'utilisant que des additions, soustractions et décalages.

$$(1) : ArcSin(x) = ArcTan\left(\frac{x}{\sqrt{1 - x^2}}\right)$$

$$(2) : ArcCos(x) = ArcTan\left(\frac{\sqrt{1 - x^2}}{x}\right)$$

$$(3) : Distance(X, Y) = \sqrt{X^2 + Y^2}$$

$$(4) : Distance(X, Y) \approx Max(|X|, |Y|) + \frac{1}{2} Min(|X|, |Y|)$$

$$(5) : Distance(X, Y) \approx Max\left(|X|, \frac{7}{8}|X| + \frac{1}{2}|Y|, \frac{1}{2}|X| + \frac{7}{8}|Y|, |Y|\right)$$

```
        RSEG    ?DT?_square?SQUARE
ACCU32: DS      4               ; registre 32 bits
        RSEG    ?PR?_square?SQUARE

mulacc: MOV     A,ACCU32+3      ; multiplie R1:R2:R3:ACCU32 par 2
        ADD     A,ACCU32+3      ; équivalent à un décalage à gauche
        MOV     ACCU32+3,A
        MOV     A,ACCU32+2
        ADDC    A,ACCU32+2
        MOV     ACCU32+2,A
        MOV     A,ACCU32+1
        ADDC    A,ACCU32+1
        MOV     ACCU32+1,A
        MOV     A,ACCU32+0
        ADDC    A,ACCU32+0
        MOV     ACCU32+0,A
        MOV     A,R3            ; décale les MSB de ACCU32
        ADDC    A,R3            ; dans R1:R2:R3
        MOV     R3,A
        MOV     A,R2
        ADDC    A,R2
        MOV     R2,A
        MOV     A,R1            ; en fait on n'utilise que
        ADDC    A,R1            ; les deux bits LSB de R1
        MOV     R1,A
        RET

; _square calcule la racine carrée de ACCU32 et place le résultat dans R6:R7
; R4:R5 est un intermediaire de calcul qui contient ((R6:R7) * 2) + 1
; R1:R2:R3 accumule les bits décales de ACCU32
; R0 est le compteur de boucle
; à la fin de l'exécution, ACCU32 = 0
; L'algorithme est très proche de l'algorithme de division mis a part que
; dans la recherche de la racine carrée, le «numerateur» est décalé par blocs
; de deux bits.
; dans le pire cas, l'exécution dure 1494 cycles
_square:
        CLR     A
        MOV     R6,A            ; résultat R6:R7
        MOV     R7,A            ; = 0
        MOV     A,ACCU32+3      ; teste si ACCU32 = 0
        ORL     A,ACCU32+2
        ORL     A,ACCU32+1
        ORL     A,ACCU32+0
        JZ      squar3          ; ACCU32 = 0, sort avec 0
        CLR     A
        MOV     R1,A            ; accumulateur de bits a 0
        MOV     R2,A
        MOV     R3,A
        MOV     R0,#16          ; résultat sur 16 bits
; boucle de calcul effectuée 16 fois
squar0: MOV     A,R7            ; résultat = résultat * 2
        ADD     A,R7
        MOV     R7,A
        MOV     A,R6
        ADDC    A,R6
        MOV     R6,A
        CALL    mulacc          ; décale les deux prochains bits
        CALL    mulacc          ; dont on veut extraire la racine carrée
        MOV     A,R7            ; temp = (résultat * 2) + 1
        ADD     A,R7
        MOV     R4,A
        MOV     A,R6
        ADDC    A,R6
        MOV     R5,A
        MOV     A,R4
        ADD     A,#1
        XCH     A,R5
        ADDC    A,#0
        MOV     R4,A            ; ici C = 0
; teste si R1:R2:R3 >= R4:R5 (partie haute >= temp)
        MOV     A,R1            ; si msb non nul
        JNZ     squar1          ; R1:R2:R3 > R4:R5
        MOV     A,R3
        SUBB    A,R5
        MOV     A,R2
        SUBB    A,R4
        JC      squar2          ; R4:R5 > R1:R2:R3
; R1:R2:R3 >= R4:R5, calcule R1:R2:R3 = R1:R2:R3 - R4:R5
squar1: MOV     A,R3            ; partie haute = partie haute - temp
        SUBB    A,R5
        MOV     R3,A
        MOV     A,R2
        SUBB    A,R4
        MOV     R2,A
        MOV     A,R1
        SUBB    A,#0
        MOV     R1,A
        MOV     A,R7            ; résultat = résultat + 1
        ADD     A,#1
        MOV     R7,A
        MOV     A,R6
        ADDC    A,#0
        MOV     R6,A
squar2: DJNZ    R0,squar0
squar3: RET
```

■ ***Listing 1 : calcul de racine en assembleur 51.***