MATHEZ VOTRE AMSTRAD

LES FONCTIONS ARCSINUS ET ARCCOSINUS

Vous avez pu remarquer que quel que soit l'ordinateur ou sa marque, il n'existe pas de fonctions mathématiques telles que Arcinus et Arccosinus. Il est évident que ces fonctions n'ont pas beaucoup d'utilité et ne sont pas employées dans des classes inférieures à la terminale. Or justement à partir de ce niveau, ces fonctions peuvent devenir importantes (physique, maths, électronique, etc.). Voici donc deux de ces fonctions que nous avons redéfinies en RSX. Attention ce programme tel qu'il est donné, ne fonctionne que sur 464. Adaptez-le vous-même en fournissant les adresses équivalentes données à la fin selon les modèles.

Le procédé n'est pas sorcier puisque nous pouvons obtenir ces deux fonctions grâce à l'arctangeante. Il suffit pour cela de quelques transformations mathématiques résultants des calculs suivants.

$$\text{Tg (Arccos(x))} = \frac{\sqrt{(1-x^2)}}{x}$$

en composant par Arctangeante on obtient

$$\text{Arccos(x)} = \frac{\text{atn } (\sqrt{(1-x^2)})}{x}$$

(Attention au signe de ×). Pour obtenir Arcsinus (x) on a

$$\text{Tg (Arcsin (x))} = \frac{x}{\sqrt{(1-x^2)}}$$

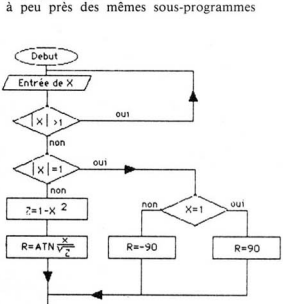
en composant par Arctangeante on obtient

$$\text{Arsin } (\times) = \text{Atn}\left(\frac{x}{\sqrt{(1-x^2)}}\right)$$

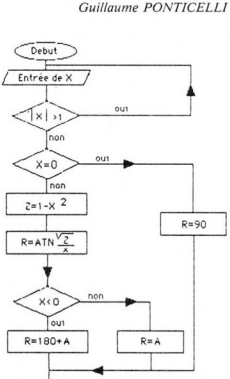
Commentaires : les deux organigrammes suivants ont été directement adaptés en langage machine. Pour les deux fonctions. × doit être compris entre − 1 inclus et 1 inclus. On a choisi d'indiquer 255 en cas d'erreur. Les calculs sont réalisés en degrés.
Organigrammes : pour l'arccosinus : comme on ne peut pas diviser par 0 et que Arccosinus (0) = 90 on l'affiche directement. Si × est différent de 0, on applique directement la formule (1). Puis on teste le signe de ×. Si × est positif, on affiche directement le résultat. Si par contre il est négatif on ajoute 180 ou on fait résultat = 180 − ABS (a). Pour l'arcsinus on ne peut pas non plus diviser par 0 (obtenu pour les deux × = − 1 et × = 1) on a choisi, d'indiquer directement le résultat ; pour Arcsin (1) = 90 deg et Arcsin (− 1) = − 90 deg. Pour le reste on applique la formule (2).
Les deux organigrammes permettent d'entrevoir que l'on peut utiliser les mêmes parties de programmes pour les deux. Le sous-programme à l'étiquette ZOU n'est que pour l'arcsinus. Avant de continuer précisons la méthode d'appel de ces routines. Du basic, après avoir assemblé le programme, on fera memory & 8fff : load "nomprog" : call & 9000. Puis pour un calcul on tapera par exemple : b = 0.2 : | arcos, àb : print b. Vous obtiendrez alors la valeur de Arccos (0.2). Cela peut paraître bizarre de prendre b puis de le réafficher. En fait b se voit attribuer par la machine une adresse arbitraire que l'on fournit (sans toutefois la connaître) par le fameux à. Une fois le calcul fini, on fournit au programme l'adresse de b (que l'on a sauvegardé) à la routine chargée de l'affichage : Print. Pour plus de clarté, expliquons le programme en assembleur.
(1) : Le registre IX possède l'adresse du contenu de la variable b. Avec le à on fournit une adresse. Donc le contenu de l'adresse IX + 0, IX + 1 sera l'adresse de la variable b. On charge alors dans HL cette adresse que l'on sauve aussi dans la pile.
(2) : attention aux manipulations de la pile. Il est fort déconseillé de l'utiliser dans un sous-programme parce que justement la machine s'en sert pour mémoriser l'adresse de départ du sous-programme. Donc, lors d'un POP vous risquez d'avoir des surprises, c'est-à-dire une tout autre donnée. C'est la raison pour laquelle on a répété les lignes ≠901E à ≠9024 aux adresses ≠9060 à ≠9066.
(3) : on charge dans DE la première adresse d'une zone de 5 octets (mem1) qui servira de mémoire de calcul. Puis on met le contenu de b dans mem1. Comme l'on veut conserver le nombre de départ pour le tester, on le sauve également dans mem0. Une fois ce travail fait, on teste le signe de mem1 (≠9099) et on revient.
(4) : on sauve le résultat contenu dans A (PUSH AF). Il a été obtenu comme suit : si le contenu de mem1 est négatif, on a l'accumulateur A égal à 255 ; si le contenu de mem1 est nul on a A = 0 ; si le contenu de mem1 est positif A = 1. On restitue ensuite le contenu de la pile dans A (POP AF).
(5) : si mem1 = 0 on va à AFF. AFF qui a la particularité de transcrire 90 (≠ 5a) en flottant et de le mettre dans mem1.
(5 bis) : (call ≠bd40) convertit un nombre entier (dans HL) en un nombre au format réel 5 octets dont le premier est pointé par le registre DE.
(6) : une fois que 90 est traduit en flottant on va à RESU (c'est la raison pour laquelle on restitue dans l'accumulateur A la réponse au test de la ligne ≠9099).

(7) : on met dans DE l'adresse de la variable b (que l'on ne connaît toujours pas) qui est dans la pile. On charge dans HL la première adresse de mem1 et on transfert le contenu de mem1 à l'adresse contenu dans DE qui sert de référence à la fonction basic Print. En effet Print va chercher la première adresse d'une zone de 5 octets (pour des nombres réels) dans le registre DE.
(8) : toujours par rapport au test effectué auparavant : si le contenu de mem1 est négatif on forme sa valeur absolue. Puis on met #ff dans la variable système à l'adresse #b8ff qui indique que le calcul est en degré. Ensuite on charge 1 dans l'accumulateur que l'on traduit en réel pour le mettre dans mem3. Ensuite on compare (#bd6a) le contenu de mem1 (×) avec 1 en mem3. L'accumulateur A contient le résultat.
(9) : si la valeur absolue de × est supérieure à 1 le calcul est impossible ; on affiche 255 (sous prog RETO).
(10) : on transcrit deux en flottant puis on élève × au carré (#bd7C) ; on met un signe moins devant × au carré (#BD6D) et on l'ajoute à mem3 − 1 en flottant − (#BD58). On fait la racine carré du tout (#BD79).
(11) : on divise le tout par × (#BD64) puis on en prend l'arctangeante (#BD91).
(12) : on prend la valeur de départ × ; on teste son signe (#BD7D). S'il est négatif (A = #ff), on prend 180 que l'on convertit en flottant et que l'on ajoute au résultat obtenu en (11). ATTENTION : Vous voyez qu'une fois traduit en flottant, on inverse le signe de 180 (#BD6D en ligne #9052). En effet l'opération s'effectue en binaire signé. Il faut donc inverser le signe obtenu après conversion de tout nombre positif dépassant 127. A partir de 128 on a le bit 7 de l'octet à 1 qui signale à l'ordinateur (en opérations signées) que le nombre est négatif. Donc prudence avec la routine #BD40. Une fois l'addition faite on saute à RESU qui affiche le résultat.
(13) : si × est positif, on affiche directement le résultat obtenu en (11).
(14) : pour ce qui est de l'arcsinus, on se sert à peu près des mêmes sous-programmes selon les besoins de l'organigramme. Le sous-programme ZOU teste la valeur absolue de × puis si × est égal à 1 ou −1 on affichera 90 ou −90 degrés.

Guillaume PONTICELLI



ORGANIGRAMME ARCSINUS



ORGANIGRAMME ARCCOSINUS

```
.COPYRIGHT 1985 MICRO-APPLICATION.
.DAMS.

        ORG   #9000
        LD    hl,tampon
        LD    bc,comext
        CALL  #bcd1
        RET
comext  DEFW  table
        JP    acos
        JP    asin
table   DEFM  ARCO
        DEFB  "S"+#80
        DEFM  ARCSI
        DEFB  "N"+#80
        DEFB  #00
acos    LD    h,(ix+1)          ;(1)
        LD    l,(ix+0)
        PUSH  hl                ;(2)
        CALL  bu
        PUSH  af                ;(4)
        CALL  z,aff             ;(5)
        POP   af                ;(4)
        JP    z,resu            ;(6)
        CALL  cont              ;(8)
        JP    z,reto            ;(9)
        CALL  didi              ;(10)
        CALL  #bd64             ;(11)
        CALL  #bd91
        LD    hl,mem0           ;(12)
        CALL  #bd70
        CP    #ff
        JP    nz,resu           ;(13)
        LD    a,#b4             ;(12)
        CALL  cef
        LD    hl,mem2
        CALL  #bd6d
        LD    hl,mem1
        LD    de,mem2
        CALL  #bd58
        JR    resu
asin    LD    h,(ix+1)          ;(14)
        LD    l,(ix+0)
        PUSH  hl
        CALL  bu
        CALL  cont
        JP    z,reto
        CP    0
        JP    z,zou
        CALL  didi
        EX    de,hl
        CALL  #bd64
        CALL  #bd91
        LD    de,mem1
        CALL  trans
        JR    resu
```

```
bu      LD   de,mem1          ;(3)
        CALL trans
        LD   hl,mem1
        LD   de,mem0
        CALL trans
        LD   hl,mem1
        CALL #bd70
        RET
cont    CP   #ff              ;(8)
        CALL z,abs
        LD   hl,#b8f7
        LD   (hl),#ff
        LD   a,1
        CALL cef
        LD   de,mem3
        LD   hl,mem2
        CALL trans
        LD   hl,mem1
        LD   de,mem3
        CALL #bd6a
        CP   1
        RET
didi    LD   a,2              ;(10)
        CALL cef
        LD   hl,mem1
        LD   de,mem2
        CALL #bd7c
        CALL #bd6d
        LD   de,mem3
        CALL #bd58
        CALL #bd79
        LD   de,mem0
        RET
resu    POP  de               ;(7)
        LD   hl,mem1
        CALL trans
        RET
abs     LD   hl,mem1          ;(8)
        CALL #bd6d
        RET
aff     LD   a,#5a
        CALL cef              ;(5 bis)
        LD   hl,mem2
        LD   de,mem1
        CALL trans
        RET
trans   LD   bc,5
        LDIR
        RET
cef     LD   l,a              ;(5 bis)
        LD   h,0
        LD   de,mem2
        CALL #bd40
        RET
zou     CALL aff
        LD   hl,mem0
        LD   de,mem3
        CALL #bd6a
        JR   z,resu
        LD   hl,mem1
        CALL #bd6d
        JR   resu
mem0    DEFS 5
mem1    DEFS 5
mem2    DEFS 5
mem3    DEFS 5
tampon  DEFS 4
reto    LD   a,#ff            ;(9)
        CALL cef
        LD   hl,mem2
        CALL #bd6d
        LD   de,mem1
        CALL trans
        JR   resu
```

Text:26869 End:27891 1022 Bytes
Hmem:36153

TABLEAU D'EQUIVALENCE ASSEMBLEUR

464	664	6128
#BD64	#BD64	#BD88
#BD91	#BDB2	#BDB5
#BD70	#BD91	#BD94
#BD58	#BD79	#BD7C
#BD6A	#BD8B	#BD8E
#BD7C	#BD9D	#BDA0
#BD6D	#BD8E	#BD91
#BD79	#BD9A	#BD9D
#BD40	#BD61	#BD64

Au renvoi (*), remplacer par :
Pour 664 : LD a,#ff
Call #BD94
Pour 6128: LD a,#ff
Call #BD97

Corrections du listing 464 pour 664

Lignes	464	664
150	91 (1er)	B2
160	70	91
180	6D	8E
200	58	79
250	91	B2
300	70	94
320	--	3E FF CD 94 BD
360	6A	8B
390	7C	9D
400	58	79
410	79	9A
440	6D	8E
490	40	61
510	6A	8B
520	6D	8E
590	6D	8E

CHARGEUR CPC 464

```
1  REM version 464                    [917]
2  REM------------                    [998]
10 MEMORY &8FFF                       [207]
20 FOR t=&9000 TO &914D               [850]
30 READ s$:s=VAL("&"+s$):POKE t,s     [1935]
NEXT t
40 CALL &9000:END                     [420]
50 DATA 21,37,91,01,0a,90             [853]
60 DATA cd,d1,bc,c9,12,90             [1093]
70 DATA c3,1e,90,c3,60,90             [1142]
80 DATA 41,52,43,4f,d3,41             [877]
90 DATA 52,43,53,49,ce,00             [1321]
```