

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 Arcsinus 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'arctangente. Il suffit pour cela de quelques transformations mathématiques résultant des calculs suivants.

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

en composant par Arctangente on obtient

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

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

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

en composant par Arctangente on obtient

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

Commentaires : les deux organigrammes suivants ont été directement adaptés en langage machine. Pour les deux fonctions, \times 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 \times est différent de 0, on applique directement la formule (1). Puis on teste le signe de \times . Si \times 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 $\times = -1$ et $\times = 1$) 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 : | arcs, ab : 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 transfère 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 l dans l'accumulateur que l'on traduit en réel pour le mettre dans mem3. Ensuite on compare (#bd6a) le contenu de mem1 (x) avec 1 en mem3. L'accumulateur A contient le résultat.

(9) : si la valeur absolue de x est supérieure à 1 le calcul est impossible ; on affiche 255 (sous prog RETO).

(10) : on transcrit deux en flottant puis on élève x au carré (#bd7c) ; on met un signe moins devant x au carré (#BD6D) et on l'ajoute à mem3 - 1 en flottant - (#BD58). On fait la racine carrée du tout (#BD79).

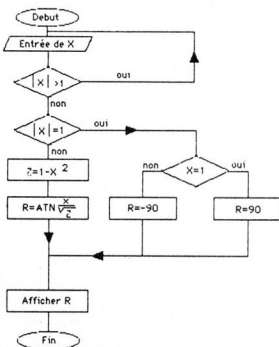
(11) : on divise le tout par x (#BD64) puis on prend l'arctangente (#BD91).

(12) : on prend la valeur de départ x ; on teste son signe (#BD70). 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 x est positif, on affiche directement le résultat obtenu en (11).

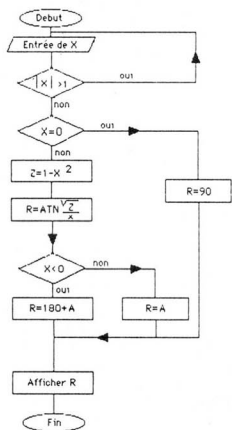
(14) : pour ce qui est de l'arcsinus, on se sert à peu près des mêmes sous-programmes



ORGANIGRAMME ARCSINUS

selon les besoins de l'organigramme. Le sous-programme ZOU teste la valeur absolue de x puis si x est égal à 1 ou -1 on affichera 90 ou -90 degrés.

Guillaume PONTICELLI



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 DEFB ARCD
DEFB "S"+#80
DEFB ARCSI
DEFB "N"+#80
DEFB #00
acos LD hl,(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
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
  
```

TRUCS ET BICOUILLES

bu	LD de,mem1 CALL trans LD hl,mem1 LD de,mem0 CALL trans LD hl,mem1 CALL #bd70 RET	(3)	CALL cef LD hl,mem2 CALL #bd6d LD de,mem1 CALL trans JR resu	
cont	CP #ff 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	(8)	Text:26869 Hmem:36153	End:27891 1022 Bytes
didi	LD a,2 CALL cef LD hl,mem1 LD de,mem2 CALL #bd7c CALL #bd6d LD de,mem3 CALL #bd58 CALL #bd79 LD de,mem0 RET	(10)		
resu	POP de LD hl,mem1 CALL trans RET	(7)		
abs	LD hl,mem1 CALL #bd6d RET	(8)		
aff	LD a,#5a CALL cef LD hl,mem2 LD de,mem1 CALL trans RET	(5 bis)		
trans	LD bc,5 LDIR RET			
cef	LD l,a LD h,0 LD de,mem2 CALL #bd40 RET	(5 bis)		
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)		

TABLEAU D'EQUIVALENCE ASSEMBLEUR

464 664 6128

#BD64	#BD64	#BD88
#BD91	#BD82	#BD85
#BD70	#BD91	#BD94
#BD58	#BD79	#BD7C
#BD6A	#BD88	#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#:=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]



100 DATA dd,66,01,dd,6e,00	[806]	40 CALL &9000:END	[420]
110 DATA e5,cd,87,90,f5,cc	[1259]	50 DATA 21,37,91,01,0a,90	[853]
120 DATA ee,90,f1,ca,df,90	[857]	60 DATA cd,d1,bc,c9,12,90	[1093]
130 DATA cd,9d,90,ca,3b,91	[1453]	70 DATA c3,1e,90,c3,60,90	[1142]
140 DATA cd,c1,90,cd,64,bd	[1012]	80 DATA 41,52,43,4f,d3,41	[877]
150 DATA cd,91,bd,21,23,91	[933]	90 DATA 52,43,53,49,ce,00	[1321]
160 DATA cd,70,bd,fe,ff,c2	[1158]	100 DATA dd,66,01,dd,6e,00	[806]
170 DATA df,90,3e,b4,cd,03	[811]	110 DATA e5,cd,87,90,f5,cc	[1259]
180 DATA 91,21,2d,91,cd,6d	[604]	120 DATA ee,90,f1,ca,df,90	[857]
190 DATA bd,21,28,91,11,2d	[1025]	130 DATA cd,9d,90,ca,3b,91	[1453]
200 DATA 91,cd,58,bd,18,7f	[741]	140 DATA cd,c1,90,cd,88,bd	[692]
210 DATA dd,66,01,dd,6e,00	[806]	150 DATA cd,b5,bd,21,23,91	[1822]
220 DATA e5,cd,87,90,cd,9d	[1198]	160 DATA cd,94,bd,fe,ff,c2	[591]
230 DATA 90,ca,3b,91,fe,00	[253]	170 DATA df,90,3e,b4,cd,03	[811]
240 DATA ca,0d,91,cd,c1,90	[757]	180 DATA 91,21,2d,91,cd,91	[793]
250 DATA eb,cd,64,bd,cd,91	[1030]	190 DATA bd,21,28,91,11,2d	[1025]
260 DATA bd,11,28,91,cd,fd	[717]	200 DATA 91,cd,7c,bd,18,7f	[723]
270 DATA 90,18,58,11,28,91	[328]	210 DATA dd,66,01,dd,6e,00	[806]
280 DATA cd,fd,90,21,28,91	[1043]	220 DATA e5,cd,87,90,cd,9d	[1198]
290 DATA 11,23,91,cd,fd,90	[1025]	230 DATA 90,ca,3b,91,fe,00	[253]
300 DATA 21,28,91,cd,70,bd	[596]	240 DATA ca,0d,91,cd,c1,90	[757]
310 DATA c9,fe,ff,cc,e7,90	[1263]	250 DATA eb,cd,88,bd,cd,b5	[970]
320 DATA 21,f7,b8,36,ff,3e	[2062]	260 DATA bd,11,28,91,cd,fd	[717]
330 DATA 01,cd,03,91,11,32	[659]	270 DATA 90,18,58,11,28,91	[328]
340 DATA 91,21,2d,91,cd,fd	[1084]	280 DATA cd,fd,90,21,28,91	[1043]
350 DATA 90,21,28,91,11,32	[179]	290 DATA 11,23,91,cd,fd,90	[1025]
360 DATA 91,cd,6a,bd,fe,01	[749]	300 DATA 21,28,91,cd,94,bd	[663]
370 DATA c9,3e,02,cd,03,91	[1408]	310 DATA c9,fe,ff,cc,e7,90	[1263]
380 DATA 21,28,91,11,2d,91	[1133]	320 DATA 3e,ff,cd,97,bd,3e	[810]
390 DATA cd,7c,bd,cd,6d,bd	[1195]	330 DATA 01,cd,03,91,11,32	[659]
400 DATA 11,32,91,cd,58,bd	[1542]	340 DATA 91,21,2d,91,cd,fd	[1084]
410 DATA cd,79,bd,11,23,91	[826]	350 DATA 90,21,28,91,11,32	[179]
420 DATA c9,d1,21,28,91,cd	[890]	360 DATA 91,cd,8e,bd,fe,01	[1405]
430 DATA fd,90,c9,21,28,91	[1322]	370 DATA c9,3e,02,cd,03,91	[1408]
440 DATA cd,6d,bd,c9,3e,5a	[1243]	380 DATA 21,28,91,11,2d,91	[1133]
450 DATA cd,03,91,21,2d,91	[1127]	390 DATA cd,a0,bd,cd,91,bd	[1315]
460 DATA 11,28,91,cd,fd,90	[1084]	400 DATA 11,32,91,cd,7c,bd	[1325]
470 DATA c9,01,05,00,ed,b0	[1208]	410 DATA cd,9d,bd,11,23,91	[938]
480 DATA c9,6f,26,00,11,2d	[1072]	420 DATA c9,d1,21,28,91,cd	[890]
490 DATA 91,cd,40,bd,c9,cd	[847]	430 DATA fd,90,c9,21,28,91	[1322]
500 DATA ee,90,21,23,91,11	[623]	440 DATA cd,91,bd,c9,3e,5a	[322]
510 DATA 32,91,cd,6a,bd,28	[1463]	450 DATA cd,03,91,21,2d,91	[1127]
520 DATA c4,21,28,91,cd,6d	[973]	460 DATA 11,28,91,cd,fd,90	[1084]
530 DATA bd,18,bc,00,00,00	[437]	470 DATA c9,01,05,00,ed,b0	[1208]
540 DATA 00,00,00,00,00,00	[705]	480 DATA c9,6f,26,00,11,2d	[1072]
550 DATA 00,00,00,00,00,00	[705]	490 DATA 91,cd,64,bd,c9,cd	[939]
560 DATA 00,00,00,00,00,00	[705]	500 DATA ee,90,21,23,91,11	[623]
570 DATA 00,00,00,3e,ff,cd	[645]	510 DATA 32,91,cd,8e,bd,28	[1724]
580 DATA 03,91,21,2d,91,cd	[1304]	520 DATA c4,21,28,91,cd,91	[938]
590 DATA 6d,bd,11,28,91,cd	[1667]	530 DATA bd,18,bc,00,00,00	[437]
600 DATA fd,90,18,91	[742]	540 DATA 00,00,00,00,00,00	[705]
		550 DATA 00,00,00,00,00,00	[705]
		560 DATA 00,00,00,00,00,00	[705]
		570 DATA 00,00,00,3e,ff,cd	[645]
		580 DATA 03,91,21,2d,91,cd	[1304]
		590 DATA 91,bd,11,28,91,cd	[907]
		600 DATA fd,90,18,91	[742]

CHARGEUR CPC 6128

```

10 MEMORY &BFFF [207]
20 FOR t=&9000 TO &914D [850]
30 READ $:s=VAL("&"+$):POKE t,s [1935]
NEXT t

```