

```

# CORDIC ALGORITHM FOR SIN COS TAN COT CALCULATION WITHOUT MATH DEPENDENCIES
#                               SIN COS TAN COT PLOTTER
#     MIPS Assembly CODE VERSION 1.0, 06/28/16 by Frank Fu Zheng
#                               No license... lol

.data

graph: .space 524288

doublesave: .space 48

#### THIS AREA IS FOR USER DEFINABLE VALUES ####

angle: .double
# ANGLE TO BE CALCULATED HERE. INPUT IN RAD BETWEEN -PI/2 and PI/2: (NOT USED IN GRAPHING MODE)
-1.047197551196597746154214461093167628065723133125035273658

userdefines: .word
# NUMBER OF ITERATIONS HERE:
30

# DESIRED FUNCTION TYPE HERE: (NOT USED IN GRAPHING MODE)
0

# type(0) is cosine
# type(1) is sine
# type(2) is tangent
# type(n) != (0, 1, 2) is cotangent

#### END OF AREA FOR USER DEFINABLE VALUES ####

arctan: .double # arctan(2^(-i)) lookup table
0.785398163397448309615660845819875721049292349843776455243
0.463647609000806116214256231461214402028537054286120263810
0.244978663126864154172082481211275810914144098381184067127
0.124354994546761435031354849163871025573170191769804089915
0.062418809995957348473979112985505113606273887797499194607
0.031239833430268276253711744892490977032495663725400040255
0.015623728620476830802801521256570318911114139800905417881
0.007812341060101111296463391842199281621222811725014723557
0.003906230131966971827628665311424387140357490115202856215
0.001953122516478818685121482625076713931610746777233510339
0.000976562189559319430403430199717290851634197015810087590
0.000488281211194898275469239625644848666192361133135003037
0.000244140620149361764016722943259659986212417790970617611
0.000122070311893670204239058646117956300930829409015787498
0.000061035156174208775021662569173829153785143536833346179
0.000030517578115526096861825953438536019750949675119437837
0.000015258789061315762107231935812697885137429238144575874
0.0000076293945311019702633884823401050905863507439184680771
0.0000038146972656064962829230756163729937228052573039688663
0.0000019073486328101870353653693059172441687143421654501533
0.00000095367431640596087942067068992311239001963412449879016
0.00000047683715820308885992758382144924707587049404378664196
0.00000023841857910155798249094797721893269783096898769063155
0.00000011920928955078068531136849713792211264596758766458673
0.000000059604644775390554413921062141788874250030195782366297
0.00000002980232238769530367640132767709503349043907067445107
0.000000014901161193847655147092516595963247108248930025964720
0.0000000074505805969238279871365645744953921132066925545665870
0.0000000037252902984619140452670705718119235836719483287370405
0.0000000018626451492309570290958838214764904345065282835738863
0.0000000009313225746154785153557354776845613038929264961492906

k: .double #k_(n + 1) = k_n / sqrt(1 + 2^(-2i)) lookup table
0.707106781186547524400844362104849039284835937688474036588
0.632455532033675866399778708886543706743911027865043365371
0.613571991077896349607809087758040886261467649779770866724
0.608833912517752421022113507547389913143624528617436447321
0.607648256256168200929321660309523045967703113581985291064
0.607351770141295959053512390387764281786606383482658421890
0.607277644093525999046915367337588998005456642227999175372
0.607259112298892730060294541822503590690139052011544331317
0.607254479332562329717398086325156626310997420219005882325
0.607253321089875163343435198563766418556579073503620611962
0.607253031529134335402284654661528141479976749060743640578
0.607252959138944813630351797637571156281372917543585046705
0.607252941041397163512970186424103551653601717702543799152
0.607252936517010234128971242079738890823360692670630211165
0.607252935385913500729555602745276786381154762838370714891
0.607252935103139317313863198069543375030385783764218712243
0.607252935032445771455825190959085549512096344265682760120
0.607252935014772384991058507559908749451104292756440578457
0.607252935010354038374850762858763511328595352263088877548
0.607252935009249451720797822067767708922089735150230489682
0.607252935008973305057284524081536914187686954245752110940
0.607252935008904268391406195660699100232849704577988559005

```

```

0.607252935008887009224936613310222139539485950775592819693
0.607252935008882694433319217707273680165850951039452140567
0.607252935008881615735414868805578489122423772920399505258
0.607252935008881346060938781580094811599065825857906018796
0.607252935008881278642319759773720149733070017046937521452
0.607252935008881261787665004322126250361248794716173053629
0.607252935008881257574001315459227760899210847250477598437
0.607252935008881256520585393243503137620008695266365917534
0.607252935008881256257231412689571981743102365700482508021

const: .double
3.141592653589793238462643383279502884197169399375105820975 # 0 constant PI
1.0 # 8 constant 1, xrp Initial
0.0 # 16 constant 0, yrp Initial
2.0 # 24 constant 2, Div const
-1.0 # 32 constant -1.0, sigma negative
1.570796326794896619231321691639751442098584699687552910487 # 40 constant PI/2
-1.570796326794896619231321691639751442098584699687552910487 # 48 constant -PI/2
64.0 # 56 constant 64
255.0 # 64 constant 255
0.006135923151542564918872350357967779070697596483154503556 # 72 constant pi/512

.globl main
.text

main:
# Draw Axes
jal plotX
jal plotY

# PLOT COSINE -pi/2 to pi/2
la $a0, userdefines # Load address of userdefines
lw $a0, 0($a0) # Load iterations to $a0
li $a1, 0
jal plot

# PLOT SINE -pi/2 to pi/2
la $a0, userdefines # Load address of userdefines
lw $a0, 0($a0) # Load iterations to $a0
li $a1, 1
jal plot

# PLOT TANGENT -pi/2 to pi/2
#la $a0, userdefines # Load address of userdefines
#lw $a0, 0($a0) # Load iterations to $a0
#li $a1, 2
#jal plot

# PLOT COTANGENT -pi/2 to pi/2
#la $a0, userdefines # Load address of userdefines
#lw $a0, 0($a0) # Load iterations to $a0
#li $a1, 3
#jal plot

j continue

plot: # Übergabevektor: plot(iterationen, funktion)
# $a0: Anzahl der Iterationen
# $a1: Gewünschte Funktion, die berechnet werden soll

# Hier könnte Dein Code stehen

jr $ra

draw: # Übergabevektor: (x, y, farbe)
# $a0: 0 ≤ x ≤ 512
# $a1: 0 ≤ y ≤ 256
# $a2: Farbe von 0 bis 0xFFFFFFFF

# Hier könnte Dein Code stehen

jr $ra

graph_update:
addi $sp, $sp, -4
sw $s0, 0($sp)
la $s0, const
ld $f4, 72($s0) # Load constant pi/512
add.d $f12, $f12, $f4 # x += pi/512
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra

```

```

graph_initialize:
    addi    $sp, $sp, -4
    sw      $s0, 0($sp)
    la      $s0, const                # Load address of constants to $s0
    ld      $f12, 48($s0)             # Load value initial (-PI/2)
    lw      $s0, 0($sp)
    addi    $sp, $sp, 4
    jr      $ra

plotX: # EXPRESS
    li      $s0, 0                    # Counter = 0
    li      $s1, 512                  # Counter end
    li      $s2, 128                  # Load 128 for middle line
    li      $s3, 0xFFAAAAA          # Color
    la      $s4, graph                # Load graph address
    sll     $s2, $s2, 11

    pXl:
    add     $t0, $s4, $s2
    sll     $t1, $s0, 2
    add     $t0, $t0, $t1
    sw      $s3, 0($t0)
    addi    $s0, $s0, 1
    bne     $s0, $s1, pXl

    jr      $ra

plotY: # EXPRESS
    li      $s0, 0                    # Counter = 0
    li      $s1, 256                  # Counter end
    li      $s2, 256                  # Load 128 for middle line
    li      $s3, 0xFFAAAAA          # Color
    la      $s4, graph                # Load graph address
    sll     $s2, $s2, 2

    pYl:
    add     $t0, $s4, $s2
    sll     $t1, $s0, 11
    add     $t0, $t0, $t1
    sw      $s3, 0($t0)

    addi    $s0, $s0, 1
    bne     $s0, $s1, pYl

    jr      $ra

cordic:
    # $a0: Iterations count
    # $a1: Function type
    # $f12: Angle
    # $f0: Result

    ### BACKUP ROUTINE
    subi    $sp, $sp, 16              # Backup $s0, $s1 to stack
    sw      $s0, 0($sp)
    sw      $s1, 4($sp)
    sw      $s2, 8($sp)
    sw      $s3, 12($sp)
    s.d     $f20, doublesave + 0
    s.d     $f22, doublesave + 8
    s.d     $f24, doublesave + 16
    s.d     $f26, doublesave + 24
    s.d     $f28, doublesave + 32
    s.d     $f30, doublesave + 40
    s.d     $f12, doublesave + 48

    li      $s3, 0                    # Clear $s3

    ### Load constants
    la      $s2, const                # Load constants address to $s2

    ### ANGLE INPUT CHECK

    ld      $f4, 48($s2)              # Set $f4 = -pi/2
    c.lt.d  $f12, $f4                 # If angle < -pi/2 set (condition flag 0) to 1 in coproc 1
    bc1t   outOfRange                # If angle < -pi/2, GOTO outOfRange

    ld      $f4, 40($s2)              # Set $f4 = 0
    c.le.d  $f12, $f4                 # If angle <= pi/2 set (condition flag 0) to 1 in coproc 1
    bc1f   outOfRange                # Otherwise it will be 0 and we are out of range

    ld      $f4, 16($s2)              # Set $f4 = 0
    c.lt.d  $f12, $f4                 # If angle < 0 set (condition flag 0) to 1 in coproc 1
    bc1t   invA                       # Invert angle & set negative flag

    ### CORDIC FUNCTION VARIABLE DEFINITIONS

```

```

cont:

la    $s1, arctan          # Load arctan base address to $s1
l.d   $f20, 8($s2)         # $f20: xrp    initial = 1
l.d   $f22, 16($s2)        # $f22: yrp    initial = 0
l.d   $f24, 8($s2)         # $f24: powVal initial = 1
l.d   $f26, 8($s2)         # $f26: sigma  initial = 1
l.d   $f30, 24($s2)        # $f30: 2      initially and always 2

### FOR LOOP INITIALIZATION
li    $s0, 0                # $s0: i

### FOR LOOP BEGIN
loop:

mov.d  $f28, $f20           # xrbp = xrp (Backup previous xrp)

mul.d  $f4, $f26, $f24      # $f4 = sigma * powVal
                                # Do this once, we need this twice

mul.d  $f6, $f4, $f22       # $f6 = sigma * powVal * yrp
sub.d  $f20, $f20, $f6      # xrp = xrp - $f6
mul.d  $f6, $f4, $f28       # $f6 = sigma * powVal * xrbp
add.d  $f22, $f22, $f6      # yrp = yrp + $f6

div.d  $f24, $f24, $f30     # powVal = powVal / 2

## ang -= sigma * arctan[i]
l.d   $f4, 0($s1)          # $f4= arctan[i] (where i is double array relative, stored in $s1)
mul.d  $f4, $f26, $f4       # $f4 = sigma * arctan[i]
sub.d  $f12, $f12, $f4      # ang = ang - (sigma * arctan[i])

l.d   $f4, 16($s2)         # Set $f4 = 0
c.lt.d $f12, $f4           # If angle < 0, set (condition flag 0) to 1 in coproc 1
bc1t  inv5                 # If angle < 0, GOTO inv5 (invert sigma)
l.d   $f26, 8($s2)         # Otherwise, sigma = 1

### FOR LOOP UPDATE
cont0:                        # Continue Label

addi   $s0, $s0, 1         # i++
addi   $s1, $s1, 8         # arctan[i + 1], because of double

### FOR LOOP EXIT CHECK
bne    $s0, $a0, loop

## get iterations[i]
la     $t1, k                # Load base address of k array
sll   $t0, $a0, 3           # $t0 = iterations * 8
add   $t1, $t1, $t0         # Add $t0 offset to base k address
subi  $t1, $t1, 8           # And subtract 8 from the offset address

l.d   $f4, 0($t1)          # Load k[iterations - 1] to $f4

mul.d  $f20, $f20, $f4      # xrp *= k[iterations - 1]
mul.d  $f22, $f22, $f4      # yrp *= k[iterations - 1]

bne    $s3, $zero, invY     # if (isNegative) yrp = -yrbp

### CHECK OUTPUT SELECTION
cont1:

beq    $a1, $zero, cosine
li     $t0, 1
beq    $a1, $t0, sine
li     $t0, 2
beq    $a1, $t0, tangent

div.d  $f0, $f20, $f22      # return (xrp / yrbp)

### RESTORE ROUTINE
cont2:

l.d   $f12, doublesave + 48
l.d   $f30, doublesave + 40
l.d   $f28, doublesave + 32
l.d   $f26, doublesave + 24
l.d   $f24, doublesave + 16
l.d   $f22, doublesave + 8
l.d   $f20, doublesave + 0
lw    $s3, 12($sp)
lw    $s2, 8($sp)
lw    $s1, 4($sp)
lw    $s0, 0($sp)          # Restore $s0, $s1 from stack
addi  $sp, $sp, 16

```

```

addi    $sp, $sp, -4
sw      $s2, 0($sp)
la      $s2, const
l.d     $f4, 24($s2)      # Load constant 2.0
add.d   $f0, $f0, $f4     # cordic(x) +=2
l.d     $f4, 56($s2)     # Load constant 64
mul.d   $f0, $f0, $f4     # cordic(x) -> (cordic(x) + 2) * 64

cvt.w.d $f4, $f0         # Set $f4 = dint((cordic(x) + 2) * 64)
mfc1    $v0, $f4         # set $a1 to $f4 -> y = int($f4)

lw      $s2, 0($sp)
addi    $sp, $sp, 4

jr      $ra              # Return to main function

outOfRange:
l.d     $f0, 16($s2)     # return 0
j       cont2            # And exit

invA:
l.d     $f4, 32($s2)     # Load const -1
mul.d   $f12, $f12, $f4  # angle = -angle
li      $s3, 1           # isNegative = 1
j       cont

invS:
l.d     $f26, 32($s2)    # Invert sigma
l.d     $f26, 32($s2)    # Load const -1
j       cont0            # Return to function

invY:
l.d     $f4, 32($s2)     # Load const -1
mul.d   $f22, $f22, $f4  # yrp = -yrp
j       cont1            # Return to function

cosine:
mov.d   $f0, $f20        # return xrp
j       cont2

sine:
mov.d   $f0, $f22        # return yrp
j       cont2

tangent:
div.d   $f0, $f22, $f20  # return(xrp / yrp)
j       cont2

```

```

#####
#####

```

```

continue:
nop      # Do absolutely nothin...

```