```
# 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:
        31

        # 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
        0.78539816339744830961566084581987572104929234984377645524 3
        0.46364760900080611621425623146121440202853705428612026381 0
        0.24497866312686415417208248121127581091414409838118406712 7
        0.12435499454676143503135484916387102557317019176980408991 5
        0.06241880999595734847397911298550511360627388779499194607
        0.03123983343026827625371174489249097703249566372540004025 5
        0.01562372862047683080280152125657031891111413980090541788 1
        0.00781234106010111129646339184219928162122281172501472355 7
        0.00390623013196697182762866531142438714035749011520285621 5
        0.00195312251647881868512148262507671393161074677723351033 9
        0.00097656218955931943040343019971729085163419701581008759 0
        0.00048828121119489827546923962564484866619236113313500303 7
        0.00024414062014936176401672294325965998621241779097061761 1
        0.00012207031189367020423905864611795630093082940901578749 8
        0.00006103515617420877502166256917382915378514353683334617 9
        0.00003051757811552609686182595343853601975094967511943783 7
        0.00001525878906131576210723193581269788513742923814457587 4
        0.00000762939453110197026338848234010509058635074391846807 71
        0.00000381469726560649628292307561637299372280525739039688663
        0.00000190734863281018703536536930591724416871434216545015 33
        0.00000095367431640596087942067068992311239001963412449879016
        0.00000047683715820308885992758382144924707587049404378664196
        0.00000023841857910155798249094797721893269783096898769063155
        0.00000011920928955078068531136849713792211264596758766458673
        0.00000005960464477539055441392106214178887425003019578236629 7
        0.00000002980232238769530367674013276770950334904390706744510 7
        0.00000001490116119384765514709251659596324710824893002596472 0
        0.00000000745058059692382798713656457449539211320669255456658 70
        0.00000000372529029846191404526707057181192358367194832873704 05
        0.00000000186264514923095702909588382147649043450652828357388 63
        0.00000000093132257461547851535573547768456130389292649614929 06

k: .double #k_(n + 1) = k_n / sqrt(1 + 2^(-2i)) lookup table
        0.70710678118654752440084436210484903928483593768847403658 8
        0.63245553203367586639977870888654370674391102786504336537 1
        0.61357199107789634960780908775804086261467649779770866724
        0.60883391251775242102211350754738991314362452861743644732 1
        0.60764825625616820092932166030952304596770311358198529106 4
        0.60735177014129595905351239038776428178660638348265842189 0
        0.60727764409352599904691536733758899800545664222799917537 2
        0.60725911229889273006029454182250359069013905201154433131 7
        0.60725447933256232971739808632515662631099742021900588232 5
        0.60725332108987516334343519856376641855657907350362061196 2
        0.60725303152913433540228465466152814147997674906074364057 8
        0.60725295913894481363035179763757115628137291754358504670 5
        0.60725294104139716351297018642410355165360171770254379915 2
        0.60725293651701023412897124207973889082336069267063021116 5
        0.60725293538591350072955560274527678638115476283837071489 1
        0.60725293510313931731386319806954337503038578376421871224 3
        0.60725293503244577145582519095908554951209634426568276012 0
        0.60725293501477238499105850755990874945110429275644057845 7
        0.60725293501035403837485076285876351132859535226308887754 8
        0.60725293500924945172079782206776770892208973515023048968 2
        0.60725293500897330505728452408153691418768695424575211094 0
        0.60725293500890426839140619566069910023284970457798855900 5
```

```
        0.607252935008887009224936613310222139539485950775592819693
        0.607252935008882694433319217707273680165850951039452140567
        0.607252935008881615735414868805578489122423772920399505258
        0.607252935008881346060938781580094811599065825857906018796
        0.607252935008881278642319759773720149733070017046937521452
        0.607252935008881261787665004322126250361248794716173053629
        0.607252935008881257574001315459227760899210847250477598437
        0.607252935008881256520585393243503137620008695266365917534
        0.607252935008881256257231412689571981743102365700482508021

const: .double
        3.14159265358979323846264338327950288419716939937510582097     #  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
        1.57079632679489661923132169163975144209858469968755291048 7    # 40 constant PI/2
        -1.57079632679489661923132169163975144209858469968755291048 7   # 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 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

        # 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

        j       continue

plot:
        # $a0: Iterations count
        # $a1: Function type

        # BACKUP ROUTINE
        subi    $sp, $sp, 28
        sw      $ra, 0($sp)
        sw      $s0, 4($sp)
        sw      $s1, 8($sp)
        sw      $s2, 12($sp)
        sw      $s3, 16($sp)
        sw      $s4, 20($sp)
        sw      $s5, 24($sp)

        sgt     $s5, $a1, 1             # Set $s5 if tangent or cotangent

        la      $s0, const              # Load address of constants to $s0
        move    $s3, $a0                # Load iteration count to $s3
        move    $s4, $a1                # Load function type to $s4

        # COLOR CONFIG
        beq     $a1, $zero, cosineColor # If cosine, do cosine color
        li      $t0, 1
        beq     $a1, $t0, sineColor     # If sine, do sine color
        li      $t0, 2
        beq     $a1, $t0, tangentColor  # If tangent, do tangent color
        li      $t0, 3
        beq     $a1, $t0, cotangentColor # If cotangent, do cotangent color

        cpl1:
```

```
        # FOR LOOP INITIALIZATION for (int i = 0; i < 512; i++)
        l.d     $f20, 48($s0)           # Load value initial (-PI/2)
        li      $s1, 0                  # Load counter
        li      $s2, 512                # Max Value

        # FOR LOOP:
        loop_nextx:

        # Call Cordic
        move    $a0, $s3                # Load iterations count to $a0
        move    $a1, $s4                # Load function type to $a1
        mov.d   $f12, $f20              # Load current angle to CORDIC argument

        jal     cordic                  # returns result to $f0
        l.d     $f4, 24($s0)            # Load constant 2.0
        add.d   $f0, $f0, $f4           # cordic(x) +=2
        l.d     $f4, 56($s0)            # 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    $a1, $f4                # set $a1 to $f4 -> y = int($f4)
        li      $t0, 255                # Load 255 to $t0
        bne     $s5, $zero, tanCheck    # Check tangent graph overflow
        tGood:
        sub     $a1, $t0, $a1           # Get negative y Offset y = 256-y, now y is good

        move    $a0, $s1                # Set $a0 to x coordinate

        jal     draw                    # AND DRAW!!

        # FOR LOOP UPDATE
        tOvldct:                        # Directly jump to update if tangent overflow detected
        l.d     $f4, 72($s0)            # Load constant pi/512
        add.d   $f20, $f20, $f4         # x += pi/512
        addi    $s1, $s1, 1             # i++

        # FOR LOOP EXIT CHECK
        bne     $s1, $s2, loop_nextx    # loop if i < 512

        # RESTORE ROUTINE
        lw      $s5, 24($sp)
        lw      $s4, 20($sp)
        lw      $s3, 16($sp)
        lw      $s2, 12($sp)
        lw      $s1, 8($sp)
        lw      $s0, 4($sp)
        lw      $ra, 0($sp)
        addi    $sp, $sp, 28
        jr      $ra

        cosineColor:
        li      $a2, 0xFFFF0000         # Cosine is red!
        j       cpl1

        sineColor:
        li      $a2, 0xFF00FF00         # Sine is green!
        j       cpl1

        tangentColor:
        li      $a2, 0xFF0000FF         # Tangent is blue!
        j       cpl1

        cotangentColor:
        li      $a2, 0xFFFFFF00         # Cotangent is Yellow!
        j       cpl1

        tanCheck:
        blt     $a1, $zero, tOvldct     # j: Tangent Overflow Continue, do not plot since it's out of range
        bgt     $a1, $t0, tOvldct
        j       tGood                   # Tangent is in range and good

plotX:  # EXPRESS
        li      $s0, 0                  # Counter = 0
        li      $s1, 512                # Counter end
        li      $s2, 128                # Load 128 for middle line
        li      $s3, 0xFFAAAAAA         # 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, 0xFFAAAAAA         # 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

        li      $s3, 0                  # Clear $s3

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

        ### ANGLE INPUT CHECK

        l.d     $f4, 48($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

        l.d     $f4, 40($s2)
        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

        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    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              # xrpb = 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    invS                    # If angle < 0, GOTO invS (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 = -yrp

### 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 / yrp)

### RESTORE ROUTINE
cont2:

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

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:                           # 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


#################################################################################################################
############################
################################################## ASM GRAPHER
#########################################################################

draw:                           # receives draw vector (x, y, color)
                                # $a0: 0 ≤ x ≤ 512
                                # $a1: 0 ≤ y ≤ 256
                                # $a2: color from 0 to 0xFFFFFFFF
        la      $t0, graph              # Load graph address
        sll     $t1, $a0, 2             # $a0 * x is x position of matrix (max 512)
        add     $t0, $t0, $t1           # Add $t1 to $t0
        sll     $t1, $a1, 11            # $a1 * y is y position of matrix (max 256)
        add     $t0, $t0, $t1           # Add $t1 to $t0
        sw      $a2, 0($t0)             # Store $a2 in draw(x, y)
        jr      $ra                     # Return

#################################################################################################################
############################

continue:
        nop                             # Do absolutely nothin...
```