

FILEID**UVXSINCOS

1

UV
2-

(2)	84	HISTORY : Detailed Current Edit History
(3)	163	DECLARATIONS - Declarative Part of Module
(5)	222	COEFFICIENT TABLES - Series Coefficients
(6)	315	MTH\$SINCOS - Radian arguments
(8)	290	MTH\$SIN
(8)	597	MTH\$COS
(9)	413	MTH\$SINCOSD - Degrees
(11)	490	MTH\$SINCOS_R5
(12)	615	MTH\$SIN_R4
(14)	733	MTH\$COS_R4
(15)	816	MTH\$SINCOSD_R5
(16)	860	MTH\$SIND_R4
(17)	913	MTH\$COSD_R4
(19)	949	REDUCE_MEDIUM
(20)	1024	REDUCE_LARGE
(21)	1407	REDUCE_DEGREES
(23)	1496	RADIAN POLYNOMIALS : Polynomials for arguments in radians
(24)	1629	CYCLE POLYNOMIALS : Polynomials for arguments in cycles
(25)	1747	DEGREE POLYNOMIALS
(27)	1864	DEGENERATE_SOLUTIONS

0000 1 .TITLE UVX\$SINCOS ; Floating Point Sine, Cosine and Sincos005
0000 2 ; Functions
0000 3 .IDENT /2-005/ ; File: MTHSINCOS.MAR EDIT: JCW2005
0000 4 :*****
0000 5 :*****
0000 6 :
0000 7 : COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 : DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 : ALL RIGHTS RESERVED.
0000 10 :
0000 11 : THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 : ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 : INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 : COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 : OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 : TRANSFERRED.
0000 17 :
0000 18 : THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 : AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 : CORPORATION.
0000 21 :
0000 22 : DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 : SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 :
0000 25 :
0000 26 :*****
0000 27 :
0000 28 :
0000 29 : FACILITY: MATH LIBRARY
0000 30 :++
0000 31 : ABSTRACT:
0000 32 :
0000 33 : MTH\$SIN and MTH\$COS are functions which return the floating point
0000 34 : sine or cosine value of their single precision floating point argu-
0000 35 : ment (radians). The call is standard call-by-reference.
0000 36 : MTH\$SIN R4 and MTH\$COS_R4 are special routines which are the same
0000 37 : as MTH\$SIN and MTH\$COS except a faster non-standard JSB call is
0000 38 : used with the argument in R0 and no registers are saved.
0000 39 :
0000 40 : MTH\$SINCOS is a routine which returns the floating point sine and
0000 41 : cosine value of its single precision floating point radian argument.
0000 42 : The call is standard call-by-reference. MTH\$SINCOS_R5 is a special
0000 43 : routine which is the same as MTH\$SINCOS, except a faster non-
0000 44 : standard JSB call is used with the argument in R0 and no registers
0000 45 : are saved.
0000 46 :
0000 47 : MTH\$SIND and MTH\$COSD are functions which return the floating point
0000 48 : sine or cosine value of their single precision floating point argu-
0000 49 : ment (degrees). The call is standard call-by-reference.
0000 50 : MTH\$SIND R4 and MTH\$COSD_R4 are special routines which are the same
0000 51 : as MTH\$SIND and MTH\$COSD except a faster non-standard JSB call is
0000 52 : used with the argument in R0 and no registers are saved.
0000 53 :
0000 54 : MTH\$SINCOSD is a routine which returns the floating point sine and
0000 55 : cosine value of its single precision floating point degree argument.
0000 56 : The call is standard call-by-reference. MTH\$SINCOSD_R5 is a special
0000 57 : routine which is the same as MTH\$SINCOSD, except a faster non-

0000 58 : standard JSB call is used with the argument in R0 and no registers
0000 59 : are saved.
0000 60 :--
0000 61 :
0000 62 :
0000 63 : VERSION: 1
0000 64 :
0000 65 : HISTORY:
0000 66 : AUTHOR:
0000 67 : MARY PAYNE & JUD LEONARD, 25-MAY-78: Version 0
0000 68 :
0000 69 : MODIFIED BY:
0000 70 :
0000 71 : 1-1 Tryggve Fossum, 28-May-78
0000 72 :
0000 73 :
0000 74 : VERSION: 2
0000 75 :
0000 76 : HISTORY:
0000 77 : AUTHOR:
0000 78 : BOB HANEK, 25-MAY-78: Version 2
0000 79 :
0000 80 : MODIFIED BY:
0000 81 :
0000 82 : Jeffrey C. Wiener 1-APR-83

0000 84 .SBTTL HISTORY ; Detailed Current Edit History
0000 85
0000 86
0000 87 : ALGORITHMIC DIFFERENCES FROM FP-11C ROUTINE:
0000 88 1. Uses POLYF, so more accuracy.
0000 89 2. Checks for $1-2^{**}-14$ before polynomial instead of checking for
0000 90 greater than 1.0 afterwards.
0000 91 3. Signals complete loss of significance.
0000 92
0000 93 Edit History for Version 1 of MTH\$SINCOS
0000 94
0000 95
0000 96 1-2 Add JSB entry points, MTH\$COS_R4 and MTH\$SIN R4. Tryggve, JUN-12-78
0000 97 1-3 Change routines so as not to use R5 and R6. Tryggve, Jun-15-78
0000 98 1-4 Use EMODD if $|x| > 8$. Tryggve, June-26-78
0000 99 1-5 Do CVTDF after EMODD. TNH June-26-78
0000 100 1-6 Check argument range after EMODF. Tryggve June-27-78
0000 101 1-7 Do CVTDF before EMODD at CBIG. TNH 25-July-78
0000 102 1-8 Fix overflow SIN($2^{**}25*\pi$), too large answers SIN($2^{**}I*\pi$)
0000 103 I=21:24. TNH 26-July-78
0000 104 1-9 Same fix for negative arg as in 1-8. TNH 26-July-78
0000 105 1-10 Pickup arg with MOVF so reserved op check. TNH 16-Aug-78
0000 106 1-011 - Put version number in standard format: three digits in
0000 107 edit number field. JBS 16-NOV-78
0000 108 1-012 - Change MTH_SINSIGLOS to MTH\$K_SINCOSSIG. JBS 07-DEC-78
0000 109 1-013 - Add A" to the PSECT directive. JBS 22-DEC-78
0000 110 1-014 - Add RSB after call to MTH\$SIGNSIGNAL so that TAN doesn't
0000 111 go into infinite loop. SBL 02-Feb-79
0000 112 1-015 - Declare externals. SBL 17-May-1979
0000 113 1-016 - Change MTH\$K_SINCOSSIG to MTH\$K_SIGLOSMAT. JBS 19-SEP-1979
0000 114 1-017 - Change argument limit to $2^{**}31$ so as to be compatible with
0000 115 higher precision routines. SBL 31-Oct-1979
0000 116 1-018 - Reduce limit to $2^{**}30$ because this routine can't handle $2^{**}31$.
0000 117 SBL 2-Nov-1979
0000 118 1-019 - Add MTH\$FLAG_JACKET, somehow left out! SBL 2-Nov-1979
0000 119 1-019A - Changed BLSS to BLSSU after unbiassing the exponent to exercise
0000 120 small argument logic. This logic was never reached prior to this
0000 121 change.
0000 122 - Modified logic for processing reduced arguments close to pi (sin)
0000 123 and pi/2 (cos) to perform the operation 1 - RARG in double
0000 124 precision. Performing the operation in single precision results
0000 125 in losing as much as the last 6 bits. This modification was made
0000 126 only for input argument with magnitude less than $2^{**}22$.
0000 127 - Modified the first word of the LOW OF C0 entry of the CSTB2 table
0000 128 from octa 022174 to octal 32174. This eliminated a negative 1 lsb
0000 129 biasing of the values returned by COS for small arguments.
0000 130 - RNH 12-FEB-1981
0000 131 - Added degree entry points. RNH 1-MAR-1981
0000 132
0000 133
0000 134 Edit History for Version 2 of MTH\$SINCOS
0000 135
0000 136 Algorithmic differences form Version 1
0000 137 1) Introduction of SINCOS entry point
0000 138 2) Elimination of the size restriction on the argument
0000 139 3) Elimination of the MTH\$K_SIGLOSMAT error
0000 140 4) Introduction of a possible underflow error for radian arguments

0000 141 :
0000 142 : 2-001 - Original
0000 143 : 2-002 - Modified REDUCE_LARGE to eliminate potential bug (similar to the
0000 144 : bugs mentioned in QAR 896.) RNH 14-Jan-82
0000 145 : 2-003 - Modified MTH\$SINCOS for microvax by replacing all D_floating
0000 146 : point instructions with equivalent G_floating point instructions.
0000 147 : JCW 1-APR-83
0000 148 : 2-004 - Modified MTH\$SIN so that it returns the same values as MTH\$SINcos
0000 149 : in the interval (0.5,PI/4). The fix was to modify the special code
0000 150 : used in MTH\$SINCOS for |X| in that interval. The modification
0000 151 : of this code was done because there was no need to JSB to
0000 152 : NEEDS_DOUBLE_SINCOS to compute COS. This problem was discovered
0000 153 : by J. Clinkebeard in the Fortran group. JCW 25-Oct-82.
0000 154 : 2-005 - To reduce the LSB error to less than 1, the COSTBC1 table was changed
0000 155 : from a polynomial of degree 3 (Hart 3820) to a polynomial of degree 4
0000 156 : (Hart 3821). The C0 coefficient is Hart's C0; the earlier version used
0000 157 : C0-1 to improve accuracy. Now, it is more accurate to use C0. The
0000 158 : appropriate code modifications were also made. Various comments were
0000 159 : also corrected. JCW 21-Jun-84
0000 160 :
0000 161 : --

```

0000 163 .SBTTL DECLARATIONS - Declarative Part of Module
0000 164
0000 165
0000 166 : INCLUDE FILES: MTH$JACKET.MAR
0000 167
0000 168 : EXTERNAL SYMBOLS:
0000 169 :
0000 170 .DSABL GBL
0000 171 .EXTRN MTH$AL_4_OV_PI
0000 172 .EXTRN MTH$$SIGNAL
0000 173 .EXTRN MTH$K_FLOUNDMAT
0000 174 .EXTRN MTH$$JACKET_TST
0000 175 :
0000 176 : EQUATED SYMBOLS:
0000 177
0FDB4049 0000 178 LF_PI_OV_4 = ^XOFDB4049 : 0.78539819
31D641E2 0000 179 LF_9_PI_OV_4 = ^X31D641E2 : 7.06858349
0000 180 :
5F304004 0000 181 LF_2_OV_PI = ^XF9834022 : 636619772
CBE44116 0000 182 G_2_OV_PI = ^XC8826DC95F304004 : 6366197723675813E+00
53D1417B 0000 183 LF_3_PI_OV_4 = ^XCBE44116 : 2.35619450
EDDF41AF 0000 184 LF_5_PI_OV_4 = ^X53D1417B : 3.92699075
0000 185 LF_7_PI_OV_4 = ^XEDDF41AF : 5.49778700
00004334 0000 186 W_45 = ^X4334 : First word of 45 in F-format
00004334 0000 187 LF_45 = ^X00004334
0000C334 0000 188 LF_M45 = ^X0000C334
2EE13D65 0000 189 LF_SMALLD = ^X2EE13D65 : 180/PI*2^-12
00000086 0000 190 X_T_OV_45 = ^XB6
0B603DB6 0000 191 LF_T_OV_45 = ^X0B603DB6
A3513BEF 0000 192 LF_CONVERT = ^XA3513BEF : pi/180 - 2^-6
2EE142E5 0000 193 LF_90_OV_PI = ^X2EE142E5 : 28.64789009
2EE10365 0000 194 LF_SMALLEST_DEG = ^X2EE10365 : 2^-128*180/PI
0000 195 :
0000 196 : MACROS:
0000 197
0000 198 $$FDEF ; Define SF$ (stack frame) symbols
0000 199 $$PSLDEF ; Define PSL (Processor Status Longword)
0000 200
0000 201
0000 202 : PSECT DECLARATIONS:
0000 203
00000000 204 .PSECT _MTH$CODE PIC,SHR,LONG,EXE,NOWRT
0000 205 ; program section for math routines
0000 206 :
0000 207 : OWN STORAGE: none
0000 208 :
0000 209 : CONSTANTS:
0000 210
0000 211 G_PI_OV_2:
0000 212 .QUAD ^XD18544421FB4019 : 1.5707963267948966
0008 213 G_PI: .QUAD ^XD18544421FB4029 : 3.1415926535897932
0008 214 G_3_PI_OV_2: .QUAD ^XD18544421FB4032 : 4.7123889803846899
0010 215 G_2_PI: .QUAD ^XD18544421FB4039 : 6.2831853071795865
0018 216
0018 217
0020 218
0020 219

```

```

0020 221
0020 222 .SBTTL COEFFICIENT TABLES - Series Coefficients
0020 223
0020 224
0020 225
0020 226
0020 227 ; Polynomial Coefficient tables for arguments in radians
0020 228 ;
0020 229 ;
0020 230
0020 231 COSTBR1: ; COS coefficients for arguments less than 1/2
6CE6BBB4 0020 232 .LONG ^X6CE6BBB4 : C3 = -.13765364E-02
ABA53E2A 0024 233 .LONG ^XA8A53E2A : C2 = .41664738E-01
FFFDBFFF 0028 234 .LONG ^XFFFDBFFF : C1 = -.49999991E+00
00004080 002C 235 .LONG ^X00004080 : C0 = .10000000E+01
00000004 0030 236 COSLENR1 = .-COSTBR1/4
0030 237
0030 238 COSTBR2: ; COS coefficients for arguments greater than 1/2
836538CC 0030 239 .LONG ^X836538CC : C4 = .24379880E-04
03C3BBB6 0034 240 .LONG ^X03C3BBB6 : C3 = -.13886619E-02
AA9D3E2A 0038 241 .LONG ^XAA9D3E2A : C2 = .41666616E-01
3C263284 003C 242 .LONG ^X3C263284 : C1 = .38485437E-08
00000000 0040 243 .LONG ^X00000000 : C0 = .00000000E+00
00000005 0044 244 COSLENR2 = .-COSTBR2/4
0044 245
0044 246 SINTBR: ; SIN coefficients
8AE4BA4C 0044 247 .LONG ^X8AE4BA4C : C3 = -.195066968E-03
83023D08 0048 248 .LONG ^X83023D08 : C2 = .833201595E-02
AAA0BF2A 004C 249 .LONG ^XAAA0BF2A : C1 = -.166666508E+00
8ADEB25E 0050 250 .LONG ^X8ADEB25E : C0 = -.323841887E-08
00000004 0054 251 SINLENR = .-SINTBR/4
0054 252
0054 253
0054 254
0054 255
0054 256
0054 257 ; Polynomial coefficients for arguments in cycles
0054 258 ;
0054 259 ;
0054 260
0054 261 COSTBC1: ; COS coefficients for arguments less than 2/pi
CFAA376C 0054 262 .LONG ^XCFAA376C : C4 = 0.35287617E-05
E275BAAA 0058 263 .LONG ^XE275BAAA : C3 = -.32593650E-03
EOED3D81 005C 264 .LONG ^XEOED3D81 : C2 = 0.15854323E-01
E9E6BF9D 0060 265 .LONG ^XE9E6BF9D : C1 = -.30842513E+00
00004080 0064 266 .LONG ^X00004080 : C0 = 0.10000000E+01
00000005 0068 267 COSLEN1 = .-COSTBC1/4
0068 268
0068 269 COSTBC2: ; COS coefficients for arguments greater than 2/pi
CFAA376C 0068 270 .LONG ^XCFAA376C : C4 = 0.35287617E-05
E275BAAA 006C 271 .LONG ^XE275BAAA : C3 = -.32593650E-03
EOED3D81 0070 272 .LONG ^XEOED3D81 : C2 = 0.15854323E-01
4F32BE6F 0074 273 .LONG ^X4F32BE6F : C1 = -.58425136E-01
00000000 0078 274 .LONG ^X00000000 : C0 = 0.00000000E+00
00000005 007C 275 COSLEN2 = .-COSTBC2/4
007C 276
007C 277 SINTBC: ; SIN coef for arg in cycles

```

```

C97DB916 007C 278 .LONG ^XC97DB916 ; C3 = -.35950437E-04
2F493C23 0080 279 .LONG ^X2F493C23 ; C2 = 0.24900010E-02
5DDCBEA5 0084 280 .LONG ^X5DDCBEA5 ; C1 = -.80745429E-01
FDA93E10 0088 281 .LONG ^XFDA93E10 ; C0 = 0.35398159E-01
00000004 008C 282 SINLEN = .-SINTBC/4

008C 283
008C 284
008C 285
008C 286
008C 287
008C 288 : Polynomial coefficients for arguments in degrees
008C 289 :
008C 290 :
008C 291
008C 292 COSDTB1: : COS coefficients for arguments less than 90/pi
E231AA2D 008C 293 .LONG ^XE231AA2D ; C3 = -0.386099064E-13
D3133284 0090 294 .LONG ^XD3133284 ; C2 = 0.386570198E-08
B502BA1F 0094 295 .LONG ^XB502BA1F ; C1 = -0.152308523E-03
00004080 0098 296 .LONG ^X00004080 ; C0 = 0.100000000E+01
00000003 009C 297 COSDLN1 = .-COSDTB1/4 - 1

009C 298
009C 299 COSDTB2: : COS coefficients for arguments greater than 90/pi
C1342177 009C 300 .LONG ^XC1342177 ; C4 = 0.209856393E-18
C612AA30 00A0 301 .LONG ^XC612AA30 ; C3 = -0.392516491E-13
D8803284 00A4 302 .LONG ^XD8803284 ; C2 = 0.386631882E-08
A876B8FD 00A8 303 .LONG ^XA876B8FD ; C1 = -0.302383960E-04
9935AF76 00AC 304 .LONG ^X9935AF76 ; C0 = -0.560699993E-10
00000004 00B0 305 COSDLN2 = .-COSDTB2/4 - 1

00B0 306
00B0 307 SINDTB: : SIN coefficients
D7E3A5DD 00B0 308 .LONG ^XD7E3A5DD ; C3 = -0.962091984E-16
630C2E6D 00B4 309 .LONG ^X630C2E6D ; C2 = 0.134938831E-10
DC01B66D 00B8 310 .LONG ^XDC01B66D ; C1 = -0.886095279E-06
A3513BEF 00BC 311 .LONG ^XA3513BEF ; C0 = 0.182829250E-02
00000003 00C0 312 SINDLN = .-SINDTB/4 - 1
00C0 313

```

00C0 315 .SBTTL MTH\$SINCOS - Radian arguments
00C0 316
00C0 317
00C0 318
00C0 319 : FUNCTIONAL DESCRIPTION:
00C0 320
00C0 321 : The SIN, COS and SINCOS routines are based on octant reduction. Given an
00C0 322 argument, x, it is written in the form
00C0 323
00C0 324 : $x = I1*(2*\pi) + I*(\pi/4) + Y1,$
00C0 325
00C0 326 where $I1$ and I are integers, $0 \leq I < 8$ and $0 \leq Y1 < \pi/4$. Since SIN and
00C0 327 COS have a period of $2*\pi$ it follows that
00C0 328
00C0 329 : $SIN(x) = SIN(I*(\pi/4) + Y1)$ and
00C0 330 : $COS(x) = COS(I*(\pi/4) + Y1).$
00C0 331
00C0 332 : Using the trigonometric identities for the sum and difference of two angles,
00C0 333 : the following table can be generated:
00C0 334
00C0 335 : If $I =$ then $SIN(x) =$ and $COS(x) =$
00C0 336 : -----
00C0 337 : 0 $SIN(Y1)$ $COS(Y1)$
00C0 338 : 1 $COS(\pi/4-Y1)$ $SIN(\pi/4-Y1)$
00C0 339 : 2 $COS(Y1)$ $-SIN(Y1)$
00C0 340 : 3 $SIN(\pi/4-Y1)$ $-COS(\pi/4-Y1)$
00C0 341 : 4 $-SIN(Y1)$ $-COS(Y1)$
00C0 342 : 5 $-COS(\pi/4-Y1)$ $-SIN(\pi/4-Y1)$
00C0 343 : 6 $-COS(Y1)$ $SIN(Y1)$
00C0 344 : 7 $-SIN(\pi/4-Y1)$ $COS(\pi/4-Y1)$
00C0 345 :
00C0 346 : Let Y be defined as $Y = Y1$ if I is even and $Y = \pi/4 - Y1$, if I is odd, then
00C0 347 : each entry of the above table is of the form $\pm SIN(Y)$ or $\pm COS(Y)$. Based
00C0 348 : on the above remarks, the SIN, COS and SINCOS routines process the input
00C0 349 : argument x , to obtain I and Y , and based on I selects a suitable polynomial
00C0 350 : approximation, $p(Y)$, to evaluate the desired function.
00C0 351
00C0 352 :
00C0 353 : INPUT PARAMETERS:
00C0 354 :
00000004 00C0 355 : LONG = 4
00000004 00C0 356 : x = 1*LONG : x is input angle in radians
00000008 00C0 357 : sine = 2*LONG : sine is SIN(x)
0000000C 00C0 358 : cosine = 3*LONG : cosine is COS(x)
00C0 359

```

      00C0 361
      00C0 362
      00C0 363
      00C0 364 ; Return sine and cosine of argument
      00C0 365 ;
      00C0 366
      00C0 367
      003C 00C0 368 .ENTRY MTH$SINCOS, ^M<R2, R3, R4, R5>
      00C2 369
      00C2 370 MTH$FLAG_JACKET
      6D 00000000'GF 9E 00C2
      00C9
      00C9
      00C9
      00C9 371
      50 04 BC 50 00C9 372 MOVF @x(AP), R0
      00000148'EF 16 00CD 373 JSB MTH$SINCOS_R5
      08 BC 50 D0 00D3 374 MOVL R0, @sine(AP)
      OC BC 51 D0 00D7 375 MOVL R1, @cosine(AP)
      04 00DB 376 RET
      00DC 377
      00DC 378
      00DC 379
      00DC 380 .SBTTL MTH$SIN
      00DC 381
      00DC 382
      00DC 383 ; Return sine of argument
      00DC 384 ;
      00DC 385
      00DC 386
      001C 00DC 387 .ENTRY MTH$SIN, ^M<R2, R3, R4>
      00DE 388
      00DE 389 MTH$FLAG_JACKET
      6D 00000000'GF 9E 00DE
      00E5
      00E5
      00E5 390
      50 04 BC 50 00E5 391 MOVF @x(AP), R0
      00000221'EF 16 00E9 392 JSB MTH$SIN_R4
      04 00EF 393 RET
      00F0 394
      00F0 395
      00F0 396
      00F0 397 .SBTTL MTH$COS
      00F0 398
      00F0 399
      00F0 400 ; Return cosine of argument
      00F0 401 ;
      00F0 402
      00F0 403
      001C 00F0 404 .ENTRY MTH$COS, ^M<R2, R3, R4>
      00F2 405
      00F2 406 MTH$FLAG_JACKET
      00F2

```

UVX\$SINCOS
2-005

: Floating Point Sine, Cosine and Sincos ^{G 2}
MTH\$COS 16-SEP-1984 02:06:12 VAX/VMS Macro V04-00
6-SEP-1984 11:29:16 [MTHRTL.SRC]UVXSINCOS.MAR;1 Page 10
(8)

6D 00000000'GF 9E 00F2
00F9
00F9
00F9
00F9
50 04 BC 50 00F9 407
000002D6'EF 16 00FD 408
04 0103 410
0104 411
MOVAB G^MTH\$\$JACKET_HND, (FP)
; set handler address to jacket
; handler
MOVF @x(AP), R0
JSB MTH\$COS_R4
RET

```

0104 413 .SBTTL MTH$SINCOSD - Degrees
0104 414
0104 415
0104 416
0104 417 : FUNCTIONAL DESCRIPTION:
0104 418
0104 419 : The SIND, COSD and SINCOSD routines are based on octant reduction. Given an
0104 420 : argument, x, it is written in the form
0104 421
0104 422 :  $x = I1 \cdot 360 + I \cdot 45 + Y1,$ 
0104 423
0104 424 : where I1 and I are integers,  $0 \leq I < 8$  and  $0 \leq Y1 < 45$ . Since SIND and
0104 425 : COSD have a period of 360 it follows that
0104 426
0104 427 :  $SIND(x) = SIND(I \cdot 45 + Y1)$  and
0104 428 :  $COSD(x) = COSD(I \cdot 45 + Y1).$ 
0104 429
0104 430 : Using the trigonometric identities for the sum and difference of two angles,
0104 431 : the following table can be generated:
0104 432
0104 433 : If I = ----- then SIND(x) = ----- and COSD(x) = -----
0104 434 : ----- -----
0104 435 : 0 SIND(Y1) COSD(Y1)
0104 436 : 1 COSD(45-Y1) SIND(45-Y1)
0104 437 : 2 COSD(Y1) -SIND(Y1)
0104 438 : 3 SIND(45-Y1) -COSD(45-Y1)
0104 439 : 4 -SIND(Y1) -COSD(Y1)
0104 440 : 5 -COSD(45-Y1) -SIND(45-Y1)
0104 441 : 6 -COSD(Y1) SIND(Y1)
0104 442 : 7 -SIND(45-Y1) COS(45-Y1)
0104 443
0104 444 : Let Y be defined as Y = Y1 if I is even and Y = 45 - Y1, if I is odd, then
0104 445 : each entry of the above table is of the form +/-SIN(Y) or +/-COS(Y). Based
0104 446 : on the above remarks, the SIND, COSD and SINCOSD routines process the input
0104 447 : argument x, to obtain I and Y, and based on I selects a suitable polynomial
0104 448 : approximation, p(Y), to evaluate the desired function.
0104 449
0104 450
00000004 0104 451 LONG = 4
00000008 0104 452 sind = 2*LONG
0000000C 0104 453 cosd = 3*LONG
0104 454

```

	003C	0104	456	.ENTRY MTH\$SINCOSD ^M<R2, R3, R4, R5>	
		0106	457	MTH\$FLAG_JACKET	
		0106	458	MOVAB G^MTH\$\$JACKET_HND, (FP)	
6D	00000000'GF	9E	0106	; set handler address to jacket	
			010D	; handler	
			010D		
	50 04 BC	50	010D	459	
	0000035E'EF	16	0111	460	MOVF @X(AP), R0
	08 BC 50	DO	0117	461	JSB MTH\$SINCOSD_R5
	OC BC 51	DO	011B	462	MOVL R0, @sin(AP)
			011F	463	MOVL R1, @cosd(AP)
			04	464	RET
			0120	465	
			0120	466	
			0120	467	
			001C	468	.ENTRY MTH\$SIND ^M<R2, R3, R4>
			0120	469	MTH\$FLAG_JACKET
6D	00000000'GF	9E	0122	470	MOVAB G^MTH\$\$JACKET_HND, (FP)
			0129	471	; set handler address to jacket
			0129		; handler
	50 04 BC	50	0129	472	
	000003AE'EF	16	012D	473	MOVF @X(AP), R0
			0133	474	JSB MTH\$SIND_R4
			04	475	RET
			0133	476	
			0134	477	
			0134	478	.ENTRY MTH\$COSD ^M<R2, R3, R4>
			0134	479	MTH\$FLAG_JACKET
			001C	480	MOVAB G^MTH\$\$JACKET_HND, (FP)
			0136	481	; set handler address to jacket
6D	00000000'GF	9E	0136	482	; handler
			0136		
			013D	483	
	50 04 BC	50	013D	484	MOVF @X(AP), R0
	00000412'EF	16	0141	485	JSB MTH\$COSD_R4
			0147	486	RET
			04	487	
			0148	488	

```

0148 490 .SBTTL MTH$SINCOS_R5
0148 491
0148 492 ; This routine computes the SIN and COS of the F-format value of R0. The
0148 493 ; computation is performed one of three ways depending on the size of the
0148 494 ; input argument, X:
0148 495
0148 496 1) If |X| < pi/4, then X is used directly in polynomial approximation
0148 497 of SIN and COS.
0148 498 2) If pi/4 <= |X| < 9*pi/4, then the subroutine REDUCE_MEDIUM is called
0148 499 to reduce the argument to an equivalent argument in radians, Y, and
0148 500 the octant, I, containing the argument. Y is then evaluated in two
0148 501 polynomials chosen as a function of I, to compute SIN(X) and COS(X).
0148 502 3) If 9*pi/4 <= |X|, then the subroutine REDUCE_LARGE is called to
0148 503 reduce the argument to an equivalent argument in cycles, Y, and the
0148 504 octant, I, containing the argument. Y is then evaluated in two
0148 505 polynomials chosen as a function of I, to compute SIN(X) and COS(X).
0148 506

0148 507 MTH$SINCOS_R5:::
      54 50 50 0148 508 MOVF   R0, R4 ; R4 = X
      OF 18 014B 509 BGEQ  POS_SINCOS
  00000157'EF 16 014D 510 JSB    SINCOS
      50 50 52 0153 511 MNEGF  R0, R0 ; R0 = SIN(|X|), R1 = COS(X)
      05 0156 512 RSB
      0157 513
      50 8000 8F AA 0157 514 SINCOS:
      015C 515 BICW  #^X8000, R0 ; R0 = |X|
  POS_SINCOS: 516 CMPF  #LF_PI_OV_4, R0 ; Compare pi/4 with |X|
      51 015C 517 BGTR  SMAEL_SINCOS ; No argument reduction is necessary
      2F 14 0163 518 CMPF  #LF_9_PI_OV_4, R0 ; Compare 9*pi/4 with |X|
  50 31D641E2 8F 51 0165 519 BGEQ  1$ ; Use special logic for |X| > 9*pi/4
      03 18 016C 520 BRW   LARGE_SINCOS
      008C 31 016E 521
      0171 522
      0171 523 ; pi/4 <= |X| < 9*pi/4
      0171 524
  0000044D'EF 16 0171 525 IS: JSB    REDUCE_MEDIUM ; Medium argument reduction routine
      0177 526 ; R3/R4 = Y = reduced argument
      0177 527 ; R2 = octant
      7E 53 7D 0177 528 MOVQ   R3, -(SP) ; Save reduced argument on stack
  000002F5'EF 52 DD 017A 529 PUSHL  R2 ; Save octant bits on stack
      16 017C 530 JSB    M_COS ; R0 = COS(X)
      52 8E 0182 531 MOVL   (SP)+, R2 ; R2 = Octant bits
      53 8E 0185 532 MOVQ   (SP)+, R3 ; R3/R4 = reduced argument
      50 DD 0188 533 PUSHL  R0 ; Save COS(X) on stack
  0000024C'EF 16 018A 534 JSB    M_SIN ; R0 = SIN(X)
      51 8E 0190 535 MOVL   (SP)+, R1 ; R1 = COS(X)
      05 0193 536 RSB
      0194 537
      0194 538 ; Logic for small arguments. |X| < pi/4.
      0194 539
      0194 540 ;
      0194 541
      0194 542 SMALL_SINCOS:
  50 4000 8F B1 0194 543 CMPW  #^X4000, R0 ; Compare 1/2 with |X|
      2A 19 0199 544 BLSS   2$ ; Sufficient overhang not available
  50 3A80 8F B1 019B 545 CMPW  #^X3A80, R0 ; Compare with 2^-12
      1F 18 01A0 546 BGEQ  1$ ; No polynomial evaluation is needed

```

FE71	55	54	50	00	01A2	547	MOVL R0, R4	; R4 = IX!	
	CF	03	54	45	01A5	548	MULF3 R4, R5	; R5 = X*X	
FEBD	CF	03	55	55	01A9	549	POLYF R5, #COSLENR1-1, COSTBR1	; R0 = COS(X)	
		50	50	DD	01AF	550	PUSHL R0	; Save COS(X) on stack	
		50	54	44	01B1	551	POLYF R5, #SINLENR-1, SINTBR	; R0 = q(X^2)	
		50	54	40	01BA	552	MULF R4, R0	; R0 = X*q(X^2)	
		51	8E	D0	01BD	553	ADDF R4, R0	; R0 = SIN(X)	
				05	01C0	554	MOVL (SP)+, R1	; R1 = COS(X)	
					01C1	555	RSB		
		51	08	50	01C1	556			
				05	01C4	557	1\$: MOVF #1.0, R1	; R0 = X, R1 = 1.0 = COS(X)	
					01C5	558	RSB		
		50	DD	01C5	559	560	2\$: PUSH R0	; Save IX! on stack	
					01C7	561			
					01C7	562	: Replace		
					01C7	563	CVTFD R0, R3		
					01C7	564	MULD2 R3, R3		
					01C7	565	:		
					01C7	566			
		54	50	99FD	01C7	567	CVTGF R0, R4	; R4/R5 = IX!	
		54	54	44FD	01CB	568	MULG2 R4, R4	; R4/R5 = X^2	
					01CF	569			
					01CF	570	: R3/R4 is needed in its G_floating representation for NEEDS_DOUBLE_SINCOS.		
					01CF	571	: The F floating representation of R3/R4 is also needed after the return		
					01CF	572	from NEEDS_DOUBLE_SINCOS. The following code will accomplish these goals.		
					01CF	573	The only line of code that is being replaced in the D_ to G_ conversion		
					01CF	574	is a PUSH R3.		
					01CF	575	:		
					01CF	576	: Note, the BICL3 used below is necessary since a CVTGF yields a rounded		
					01CF	577	value instead of a desired truncated value is obtained. The rounding bit		
					01CF	578	(bit 12 of the second longword) plus all lower order bits of word 3, that		
					01CF	579	is, the first word of the second longword are zeroed.		
					01CF	580			
		53	54	D0	01CF	581	MOVL R4, R3	; R3 = first longword of G_floating	
					01D2	582		representation of X^2	
		54	00001FFF	8F	CB	01D2	583	BICL3 #^X00001FFF, R5, R4	; Rounding bit and all lower order
					01DA	584		bits are zeroed. Original second	
					01DA	585		longword of G_floating rep of	
					01DA	586		X^2 is saved in R5	
		7E	53	33FD	01DA	587	CVTGF R3, -(SP)	; Save X^2 on the stack	
		54	55	D0	01DE	588	MOVL R5, R4	; R3/R4 is G_floating rep of X^2	
					01E1	589	:		
					01E1	590	PUSHL R3	; Save x^2 on stack	
					01E1	591	:		
		00000778'EF	16	01E1	592	JSB NEEDS_DOUBLE_SINCOS	; Use special logic to obtain overhang		
		55	50	D0	01E7	593	MOVL R0, R5	; Save COS(X) in R5	
		50	8E	D0	01EA	594	MOVL (SP)+, R0	; R0 = X^2	
		03	50	55	01ED	595	POLYF R0, #SINLENR-1, SINTBR	; R0 = q(X^2)	
		50	6E	44	01F3	596	MULF (SP), R0	; R0 = X*q(X^2)	
		50	8E	40	01F6	597	ADDF (SP)+, R0	; R0 = SIN(X)	
		51	55	D0	01F9	598	MOVL R5, R1	; R1 = COS(X)	
				05	01FC	599	RSB		
					01FD	600			
					01FD	601			
					01FD	602	LARGE_SINCOS:		
		000004C1'EF	16	01FD	603	JSB REDUCE_LARGE	; R3/R4 = reduced argument (in cycles)		

		0203	604					
	52	DD	0203	605	PUSHL	R2	: R2 = octant bits	
7E	53	7D	0205	606	MOVQ	R3, -(SP)	; Save octant bits on stack	
		16	0208	607	JSB	L COS	; Save reduced argument on stack	
	53	8E	7D	020E	608	MOVQ	(SP)+, R3	; R0 = COS(X)
	52	6E	DO	0211	609	MOVL	(SP), R2	; Reduced argument in R3/R4
	6E	50	DO	0214	610	MOVL	R0, (SP)	; R2 = octant bits
000002AA'EF		16	0217	611	JSB	L SIN	; R1 = COS(X)	
51	8E	DO	021D	612	MOVL	(SP)+, R1	; R0 = SIN(X)	
		05	0220	613	RSB		; R1 = COS(X)	

```

0221 615 .SBTTL MTH$SIN_R4
0221 616
0221 617 ; This routine computes the SIN of the F-format value of R0. The computation
0221 618 ; is performed one of three ways depending on the size of the input argument,
0221 619 X:
0221 620
0221 621 1) If  $|X| < \pi/4$ , then X is used directly in a polynomial approximation
0221 622 of SIN.
0221 623 2) If  $\pi/4 \leq |X| < 9\pi/4$ , then the subroutine REDUCE_MEDIUM is called
0221 624 to reduce the argument to an equivalent argument in radians, Y, and
0221 625 the octant, I, containing the argument. Y is then evaluated in a
0221 626 polynomial chosen as a function of I to compute SIN(X).
0221 627 3) If  $9\pi/4 \leq |X|$ , then the subroutine REDUCE_LARGE is called to
0221 628 reduce the argument to an equivalent argument in cycles, Y, and the
0221 629 octant, I containing the argument. Y is then evaluated in a
0221 630 polynomial chosen as a function of I to compute SIN(X).
0221 631
0221 632 MTH$SIN_R4:: ; Check the sign of R0
      50 53 0221 633 TSTF   R0
      OF 18 0223 634 BGEQ   POS_SIN
      50 50 52 0225 635 JSB     SIN
      50 50 05 022B 636 MNEGF  R0, R0
      022E 637 RSB
      022F 638
      50 8000 BF AA 022F 639 SIN:
      0234 640 BICW   #^X8000, R0 ; R0 = |X|
      50 0FDB4049 8F 51 0234 641 POS_SIN: ; Compare pi/4 with |X|
      23 14 023B 642 CMPF   #LF PI OV_4, R0 ; No argument reduction is necessary
      50 31D641E2 8F 51 023D 643 BGTR   SMAEL_SIN
      5E 19 0244 644 CMPF   #LF 9-PI OV_4, R0 ; Compare 9*pi/4 with |X|
      0246 645 BLSS   LARGE_SIN ; Use special logic for |X| > 9*pi/4
      0246 646
      0246 647 ; pi/4 < |X| < 9*pi/4
      0246 648
      0246 649 ; Branch to one of four polynomial
      0000044D'EF 16 0246 650 JSB    REDUCE_MEDIUM ; Medium argument reduction routine
      024C 651 ; R3/R4 = Y = reduced argument
      024C 652 ; R2 = octant
      07 00 52 8F 024C 653 M_SIN: CASEB R2, #0, #7 ; Branch to one of four polynomial
      0250 654 ; evaluations depending on the
      05B6' 0250 655 1$: WORD P_SIN_R-1$ ; octant bits.
      0514' 0252 656 WORD P_COS_R-1$ ;
      0514' 0254 657 WORD P_COS_R-1$ ;
      05B1' 0256 658 WORD N_SIN_R-1$ ;
      05B1' 0258 659 WORD N_SIN_R-1$ ;
      0560' 025A 660 WORD N_COS_R-1$ ;
      0560' 025C 661 WORD N_COS_R-1$ ;
      05B6' 025E 662 WORD P_SIN_R-1$ ;
      0260 663
      0260 664 ; Logic for small arguments. |X| < pi/4.
      0260 665
      0260 666
      0260 667
      50 4000 BF B1 0260 668 SMALL_SIN: ; Compare 1/2 with |X|
      1A 19 0265 669 CMPW   #^X4000, R0 ; Sufficient overhang not available
      50 3A80 BF B1 0267 670 BLSS   2$ ; Compare with 2^-12
      0267 671 CMPW   #^X3A80, R0

```

FDCA CF 54 12 18 026C 672 BGEQ 1\$; No polynomial evaluation is needed
 50 50 D0 026E 673 MOVL R0, R4
 50 50 44 0271 674 MULF R0, R0
 03 50 55 0274 675 POLYF R0, #SINLENR-1, SINTBR ; R0 = q(x^2)
 50 54 44 027A 676 MULF R4, R0
 50 54 40 027D 677 ADDF R4, R0 ; R0 = SIN(X)
 50 54 05 0280 678 1\$: RSB
 50 50 DD 0281 679 ;
 50 50 99FD 0283 680 2\$: PUSHL R0 ; Save |x| on stack
 50 50 44FD 0287 681 CVTGF R0, R0 ; R0/R1 = |x|
 50 50 028B 682 MULG2 R0, R0 ; R0/R1 = X^2
 50 50 028B 683 ;
 50 50 028B 684 ; Note, the BICL3 used below is necessary since a CVTGF yields a rounded
 50 50 028B 685 ; value instead of a desired truncated value is obtained. The rounding bit
 50 50 028B 686 ; (bit 12 of the second longword) plus all lower order bits of word 3, that
 50 50 028B 687 ; is, the first word of the second longword are zeroed.
 50 50 028B 688 ;
 51 51 00001FFF 8F CB 028B 689 ; R0 = first longword of G_floating
 51 51 00001FFF 8F CB 028B 690 ; representation of X^2
 50 50 33FD 0293 691 BICL3 #^X00001FFF, R1, R1 ; Rounding bit and all lower order
 03 50 55 0293 692 ; bits are zeroed. Original second
 50 6E 44 0293 693 ; longword of G_floating rep of
 50 8E 40 02A0 694 ; X^2 is saved in R5
 50 8E 05 02A3 695 CVTGF R0, R0 ; R0 = X^2
 50 8E 05 02A3 696 POLYF R0, #SINLENR-1, SINTBR ; R0 = q(X^2)
 50 8E 05 02A3 697 MULF2 (SP), R0 ; R0 = X*q(X^2)
 50 8E 05 02A3 698 ADDF2 (SP)+, R0 ; R0 = SIN(X)
 50 8E 05 02A3 699 RSB ;
 50 8E 05 02A4 700 ;
 000004C1'EF 16 02A4 701 LARGE_SIN:
 53 D5 02AA 702 JSB REDUCE_LARGE ; R3/R4 = reduced argument (in cycles)
 14 13 02AC 703 L_SIN: TSTL R3 ; R2 = octant bits
 53 D5 02AA 704 BEQL DEGENERATE_CASE_SIN ; Check for degenerate case
 07 00 52 8F 02AE 705 ;
 07 00 52 8F 02AE 706 CASEB R2, #0, #7
 0619' 02B2 707 ;
 057B' 02B2 708 1\$: .WORD P_SIN_C-1\$
 057B' 02B4 709 .WORD P_COS_C-1\$
 057B' 02B6 710 .WORD P_COS_C-1\$
 0619' 02B8 711 .WORD P_SIN_C-1\$
 0614' 02BA 712 .WORD P_SIN_C-1\$
 05C5' 02BC 713 .WORD N_SIN_C-1\$
 05C5' 02BE 714 .WORD N_COS_C-1\$
 0614' 02C0 715 .WORD N_COS_C-1\$
 02C2 716 .WORD N_SIN_C-1\$
 02C2 717 ;
 02C2 718 DEGENERATE_CASE_SIN:
 02C2 719 ;
 52 52 52 01 8A 02C2 720 BICB #1, R2 ; Compute index as (R2 - 1)/2
 03 00 FF 8F 9C 02C5 721 ROTL #1, R2, R2
 52 52 52 01 8A 02CA 722 CASEB R2, #0, #3
 0718' 02CE 723 ;
 0724' 02D0 724 1\$: .WORD P_ONE-1\$
 071C' 02D2 725 .WORD UNFL -1\$
 0724' 02D4 726 .WORD N_ONE-1\$
 0724' 02D4 727 .WORD UNFL -1\$
 0724' 02D4 728 ;

UVX\$SINCOS
2-005

; Floating Point Sine, Cosine and Sincos ^{B 3}
MTH\$SIN_R4 16-SEP-1984 02:06:12 VAX/VMS Macro V04-00
6-SEP-1984 11:29:16 [MTHRTL.SRC]UVXSINCOS.MAR;1 Page 18
(12)

02D6 729

```

02D6 731
02D6 732
02D6 733 .SBTTL MTHSCOS_R4
02D6 734
02D6 735 ; This routine computes the COS of the F-format value of R0. The computation
02D6 736 ; is performed one of three ways depending on the size of the input argument,
02D6 737 ; X. The processing is the same as described for MTH$SIN_R4.
02D6 738
02D6 739
02D6 740 MTHSCOS_R4:::
50 50 8000 50 53 02D6 741 TSTF R0 ; Check for reserved operand
50 UFDB4049 8F AA 02D8 742 BICW #^X8000, R0 ; R0 = |X|
50 31D641E2 8F 23 14 02DD 743 CMPF #LF PI OV_4, R0 ; Compare pi/4 with |X|
50 3D 19 02E4 744 BGTR SMAEL_COS ; No argument reduction is necessary
50 02E6 745 CMPF #LF 9-PI OV_4, R0 ; Compare 9*pi/4 with |X|
50 02ED 746 BLSS LARGE_COS ; Use special logic for |X| > 9*pi/4
02EF 747
02EF 748 ; pi/4 <= |X| < 9*pi/4
02EF 749
02EF 750 :
0000044D'EF 16 02EF 751 JSB REDUCE_MEDIUM ; Medium argument reduction routine
02F5 752
02F5 753
07 00 52 8F 02F5 754 M_COS: CASEB R2, #0, #7 ; R3/R4 = Y = reduced argument
02F9 755
046B' 02F9 756 1$: .WORD P_COS_R-1$ ; R2 = octant
0508' 02FB 757 .WORD N_SIN_R-1$ ; Branch to one of four polynomial
0508' 02FD 758 .WORD N_SIN_R-1$ ; evaluations depending on the
04B7' 02FF 759 .WORD N_COS_R-1$ ; octant bits.
04B7' 0301 760 .WORD N_COS_R-1$ ; Branch to one of four polynomial
050D' 0303 761 .WORD P_SIN_R-1$ ; evaluations depending on the
050D' 0305 762 .WORD P_SIN_R-1$ ; octant bits.
046B' 0307 763 .WORD P_COS_R-1$ ; Branch to one of four polynomial
0309 764 .WORD P_COS_R-1$ ; evaluations depending on the
0309 765 : Logic for small arguments. |X| < pi/4.
0309 766
0309 767
0309 768
50 4000 8F B1 0309 769 SMALL_COS: ; Small argument reduction
07 18 030E 770 CMPW #^X4000, R0 ; Compare 1/2 with |X|
0310 771 BGEQ 1$ ; Sufficient overhang is available
0310 772 :
0310 773 : CVTFD R0, R3 ; R3/R4 = |X|
0310 774 :
53 50 99FD 0310 775 CVTFG R0, R3 ; R3/R4 = |X|
50 045D 31 0314 776 BRW NEEDS_DOUBLE ; Use special logic to obtain overhang
50 3A80 8F B1 0317 777 1$: CMPW #^X3A80, R0 ; Compare with 2^-12
50 0A 18 031C 778 BGEQ 2$ ; No polynomial evaluation is needed
50 50 44 031E 779 MULF R0,R0 ; R0 = X*X
FCF9 CF 03 50 55 0321 780 POLYF R0, #COSLENR1-1, COSTBR1; R0 = COS(X)
05 0327 781 RSB
0328 782
50 08 50 0328 783 2$: MOVF #1.0, R0 ; R0 = 1.0 = COS(X)
05 0328 784 RSB
032C 785
032C 786
032C 787 LARGE_COS:

```

```

000004C1'EF 16 032C 788 JSB REDUCE_LARGE ; R3/R4 = reduced argument (in cycles)
      53 D5 0332 789 ; R2 = octant bits
      14 13 0334 790 L_COS: TSTL R3 ; Check for degenerate case
      0336 791 BEQL DEGENERATE_CASE_COS

07 00 52 8F 0336 793 CASEB R2, #0, #7
      04F3' 033A 794 1$: .WORD P_COS_C-1$
      0591' 033C 795 .WORD P_SIN_C-1$
      058C' 033E 796 .WORD N_SIN_C-1$
      053D' 0340 797 .WORD N_COS_C-1$
      053D' 0342 798 .WORD N_COS_C-1$
      058C' 0344 799 .WORD N_SIN_C-1$
      0591' 0346 800 .WORD P_SIN_C-1$
      04F3' 0348 801 .WORD P_COS_C-1$

      034A 802
      034A 803
      034A 804 DEGENERATE_CASE_COS:
      034A 805

52 03 52 52 FF 01 8A 034A 806 BICB #1, R2 ; Compute index as (R2 - 1)/2
      00 52 8F 9C 034D 807 ROTL #-1, R2, R2
      0352 808 CASEB R2, #0, #3
      0356 809

      069C' 0356 810 1$: .WORD UNFL -1$
      0694' 0358 811 .WORD N_ONE-1$
      069C' 035A 812 .WORD UNFL -1$
      0690' 035C 813 .WORD P_ONE-1$
      035E 814
  
```

035E 816 .SBTTL MTH\$SINCOSD_R5
 035E 817
 035E 818 ; This routine computes the SIND and COSD of the F-format value of R0. The
 035E 819 computation is performed one of two ways depending on the size of the input
 035E 820 argument, X:
 035E 821
 035E 822 1) If $|X| < 45$, then X is used directly in polynomial approximation
 035E 823 of SIND and COSD.
 035E 824 2) If $45 \leq |X|$, then the subroutine REDUCE_DEGREES is called to reduce
 035E 825 the argument to an equivalent argument in degrees, Y, and the
 035E 826 octant, I containing the argument. Y is then evaluated in two
 035E 827 polynomials chosen as a function of I, to compute SIND(X) and
 035E 828 COSD(X).
 035E 829
 035E 830 MTH\$SINCOSD_R5::
 50 53 035E 831 TSTF R0
 OF 18 0360 832 BGEQ SINCOSD
 00000371'EF 16 0362 833 BICW #^X8000, R0
 50 50 52 0367 834 JSB SINCOSD
 05 036D 835 MNEGF R0, R0
 0370 836 RSB
 0371 837
 0371 838 SINCOSD:
 50 4334 8F B1 0371 839 CMPW #W 45, R0
 1E 14 0376 840 BGTR SMALL_SINCOSD
 000006E2'EF 16 0378 841 JSB REDUCE_DEGREES
 7E 53 7D 037E 842 MOVQ R3, -(SP)
 00000426'EF 16 0381 843 JSB EVAL_COSD
 53 8E 7D 0387 844 MOVQ (SP)+, R3
 50 DD 038A 845 PUSHL R0
 000003CE'EF 16 038C 846 JSB EVAL_SIND
 51 8E D0 0392 847 MOVL (SP)+, R1
 05 0395 848 RSB
 0396 849
 0396 850
 0396 851 SMALL_SINCOSD:
 50 DD 0396 852 PUSHL R0
 0000043A'EF 16 0398 853 JSB SMALL_COSD
 55 50 D0 039E 854 MOVL R0, R5
 50 8E D0 03A1 855 MOVL (SP)+, R0
 000003E2'EF 16 03A4 856 JSB SMALL_SIND
 51 55 D0 03AA 857 MOVL R5, RT
 05 03AD 858 RSB

; R0 = |X|
; R0/R1 = SIND(|X|)/COSD(|X|)
; R0 = -SIND(|X|)

; Compare 45 to |X|
; special processing for small arg
; R3/R4 = octant/reduced argument
; Save octant bits and reduced arg
; R0 = COSD(Y)
; R3/R4 = octant/reduced argument
; Save COSD(Y)
; R0 = SIND(Y)
; R1 = COSD(Y)

; Save argument
; R0 = COSD(|X|)
; R5 = COSD(|X|)
; R0 = argument
; R0 = SIND(X)
; R1 = COSD(|X|)

03AE 860 .SBTLL MTH\$SIND_R4
 03AE 861
 03AE 862 : This routine computes the SIND of the F-format value of R0. The computation
 03AE 863 : is performed one of two ways depending on the size of the input argument, X:
 03AE 864 :
 03AE 865 : 1) If $|X| < 45$, then X is used directly in polynomial approximation
 03AE 866 : of SIND.
 03AE 867 : 2) If $45 \leq |X|$, then the subroutine REDUCE_DEGREES is called to reduce
 03AE 868 : the argument to an equivalent argument in degrees, Y, and the
 03AE 869 : octant, I, containing the argument. Y is then evaluated in two
 03AE 870 : polynomials chosen as a function of I, to compute SIND(X).
 03AE 871
 03AE 872 MTH\$SIND_R4:
 50 53 03AE 873 TSTF R0 ; R0 = X
 OF 18 03B0 874 BGEQ POS_SIND
 000003BC'EF 16 03B2 875 JSB NEG_SIND
 50 50 52 03B8 876 MNEGF R0, R0 ; R0 = SIND(|X|)
 05 03B8 877 RSB ; R0 = -SIND(|X|)
 03BC 878
 03BC 879 NEG_SIND:
 50 8000 8F AA 03BC 880 BICW #^X8000, R0 ; R0 = |X|
 50 4334 8F B1 03C1 881 POS_SIND:
 1A 14 03C6 882 CMPW #W 45, R0 ; Compare 45 to |X|
 000006E2'EF 16 03C8 883 BGTR SMALL_SIND ; special processing for small arg
 03CE 884 JSB REDUCE_DEGREES ; R3/R4 = octant/reduced argument
 03CE 885
 07 00 53 03CE 886 EVAL_SIND:
 05FC' 03D2 887 CASEB R3, #0, #7
 052C' 03D4 888 1\$: WORD P_SIN_D-1\$
 052C' 03D6 889 WORD P_COS_D-1\$
 05FC' 03D8 890 WORD P_COS_D-1\$
 05F9' 03DA 891 WORD P_SIN_D-1\$
 0590' 03DC 892 WORD N_SIN_D-1\$
 0590' 03DE 893 WORD N_COS_D-1\$
 05F9' 03E0 894 WORD N_COS_D-1\$
 03E2 895 WORD N_SIN_D-1\$
 03E2 896
 03E2 897
 03E2 898 SMALL_SIND:
 50 2EE13D65 8F 51 03E2 899 CMPF #LF_SMALLD, R0 ; Compare 180/pi*2^-12 with |X|
 06 14 03E9 900 BGTR 1\$; No polynomial evaluation is
 54 50 00 03EB 901 MOVL R0, R4 necessary
 05DD 31 03EE 902 BRW P_SIN_D
 50 53 03F1 903 1\$: TSTF R0 ; Check for zero
 1C 13 03F3 904 BEQL 3\$; Return if R0 = 0
 50 2EE10365 8F 51 03F5 905 CMPF #LF_SMALLEST_DEG, R0 ; Check for possible underflow on
 03 15 03FC 906 BLEQ 2\$ conversion to radians
 05F1 31 03FE 907 BRW UNFL ; Underflow will occur on conversion
 51 50 A3513BEF 8F 45 0401 908 2\$: MULF3 #LF_CONVERT, R0, R1 ; R1 = (pi/180 - 2^-6)*|X|
 50 0300 8F A2 0409 909 SUBW #^X300, R0 ; R0 = |X|^2^-6
 50 51 40 040E 910 ADDF R1, R0 ; R0 = SIND(|X|) = (pi/180)|X|
 05 0411 911 3\$: RSB

```

        0412 913      .SBTTL MTH$COSD_R4
        0412 914
        0412 915 ; This routine computes the COSD of the F-format value of R0. The computation
        0412 916 ; is performed one of two ways depending on the size of the input argument, X:
        0412 917 ; Details are given in the discussion on MTH$SIND_R4.
        0412 918
        0412 919 MTH$COSD_R4:::
50   8000 50 53 0412 920      TSTF    R0          ; Check for reserved operand
50   4334 8F AA 0414 921      BICW    #^X8000, R0
50   1A 14 0419 922      CMPW    #W 45, R0          ; R0 = !X!
000006E2'EF 16 0420 923      BGTR    SMALL_COSD
                                JSB     REDUCE_DEGREES ; Compare 45 to !X!
                                0426
                                0426 EVAL_COSD:
07   00 53 8F 0426 927      CASEB   R3, #0, #7
04D4' 042A 928 1$:      WORD    P_COS_D-1$
05A4' 042C 929      WORD    P_SIN_D-1$
05A1' 042E 930      WORD    N_SIN_D-1$
0538' 0430 931      WORD    N_COS_D-1$
0538' 0432 932      WORD    N_COS_D-1$
05A1' 0434 933      WORD    N_SIN_D-1$
05A4' 0436 934      WORD    P_SIN_D-1$
04D4' 0438 935      WORD    P_COS_D-1$
                                043A 936
                                043A 937
                                043A 938 SMALL_COSD:
50   2EE13D65 8F 51 043A 939      CMPF    #LF_SMALLD, R0 ; Compare 180/pi*2^-12 with !X!
54   06 14 0441 940      BGTR    1$          ; Check if polynomial evaluation is
      50  D0 0443 941      MOVL    R0, R4
      04B5 31 0446 942      BRW    P_COS_D
50   08 50 0449 943 1$:      MOVF    #T, R0          ; POLY needed
      05 044C 944      RSB
                                044D 945

```

```

044D 947
044D 948
044D 949
044D 950
044D 951
044D 952 : This routine assumes that the absolute value of the argument, X, is in R0
044D 953 and that pi/4 <= |X| < 9*pi/4. It returns a double precision reduced
044D 954 argument, Y, in R3/R4, and the octant bits in R2.
044D 955
044D 956 : The reduced argument is obtained by locating the octant that X is in through
044D 957 a binary search and then subtracting off a suitable multiple of pi/2
044D 958 :
044D 959
044D 960 REDUCE_MEDIUM:
044D 961 :
044D 962 : Omit the line of code:
044D 963 CLRL R1 ; R0/R1 = |X|
044D 964 :
50 53D1417B 8F 51 044D 965 CMPF #LF_5_PI_OV_4, R0 ; |X| >= 5*pi/4
50 CBE44116 8F 51 0454 966 BLEQ 5$ ; |X| >= 3*pi/4
      15 15 0456 967 CMPF #LF_3_PI_OV_4, R0
      15 15 045D 968 BLEQ 3$ ; |X| >= 3*pi/4
      045F 969 :
      045F 970 :
      045F 971 : SUBD3 D_PI_OV_2, R0, R3 ; R3/R4 = |X| - pi/2
      045F 972 CVTFG R0, R0 ; R0/R1 = |X|
      0463 973 SUBG3 G_PI_OV_2, R0, R3 ; R3/R4 = |X| - pi/2
      046A 974 BGEQ 2$ ; Octant bits = 001
      52 01 D0 046C 975 MOVL #1, R2
      05 046F 976 RSB
      0470 977
      52 02 D0 0470 978 2$: MOVL #2, R2 ; Octant bits = 010
      05 0473 979 RSB
      0474 980
      0474 981 : old SUBD3 D_PI,R0,R3
      0474 982 :
      0474 983
53 50 50 50 99FD 0474 984 3$: CVTFG R0, R0 ; R0/R1 = |X|
      FB98 CF 43FD 0478 985 SUBG3 G_PI, R0, R3 ; R3/R4 = |X| - pi
      04 18 047F 986 BGEQ 4$ ; Octant bits = 011
      52 03 D0 0481 987 MOVL #3, R2
      05 0484 988 RSB
      0485 989
      52 04 D0 0485 990 4$: MOVL #4, R2 ; Octant bits = 100
      05 0488 991 RSB
      0489 992
50 EDDF41AF 8F 51 0489 993 5$: CMPF #LF_7_PI_OV_4, R0 ; |X| >= 7*pi/4
      15 15 0490 994 BLEQ 7$ ; R3/R4 = |X| - 3*pi/2
      0492 995 :
      0492 996 : SUBD3 D_3_PI_OV_2, R0, R3
      0492 997 :
53 50 50 50 99FD 0492 998 CVTFG R0, R0 ; R0/R1 = |X|
      FB75 CF 43FD 0496 999 SUBG3 G_3_PI_OV_2, R0, R3 ; R3/R4 = |X| - 3*pi/2
      049D 1000 BGEQ 6$ ; Octant bits = 101
      52 04 18 049D 1001 MOVL #5, R2
      05 049F 1002 RSB
      04A2 1003

```

52 06 D0 04A3 1004
05 04A3 1005 6\$: MOVL #6, R2 ; Octant bits = 110
04A6 1006
04A7 1007
04A7 1008 ; old
04A7 1009 ; SUBD3 D_2_PI, R0, R3 ; R3/R4 = |X| - 2*pi
04A7 1010
53 50 50 99FD 04A7 1011 7\$: CVTFG R0, R0 ; R0/R1 = |X|
FB68 CF 43FD 04AB 1012 SUBG3 G_2_PI, R0, R3 ; R3/R4 = |X| - 2*pi
52 07 D0 04B2 1013 MOVL #7, R2 ; Octant bits = 111
05 04B5 1014 RSB
04B6 1015
52 D4 04B6 1016 8\$: CLRL R2 ; Octant bits = 000
05 04B8 1017 RSB
04B9 1018
04B9 1019
04B9 1020
04B9 1021
04B9 1022

```

04B9 1024 .SBTTL REDUCE_LARGE
04B9 1025
04B9 1026
04B9 1027 : This routine is used to reduce large arguments (|X| >= 9*pi/4) modulo pi/4.
04B9 1028 : It returns the reduced argument, Y, in R3/R4 in units of cycles, and returns
04B9 1029 : the octant bits, I, in R2.
04B9 1030
04B9 1031 : The method of reduction is as follows:
04B9 1032
04B9 1033 :  $x*(4/\pi) = 2^n*f*(4/\pi)$  where n is an integer and  $1/2 \leq f < 1$ 
04B9 1034 :  $= 2^{(n-24)} * (2^{24}*f) * (4/\pi)$ 
04B9 1035 :  $= (2^{24}*f) * (2^{(n-24)} * 4/\pi)$ 
04B9 1036 :  $= K*C$ , where K =  $2^{24}*f$  is an integer and C =  $2^{(n-24)} * 4/\pi$ 
04B9 1037 : Let L = K*C modulo 8, where  $0 \leq L < 8$ , and let I = the integer(L) and
04B9 1038 : h = fract(L), then if I is even Y = h, otherwise Y = 1-h
04B9 1039
04B9 1040 : CONSTANTS:
04B9 1041
00001E80 04B9 1042 L_INT_WEIGHT = ^X1E80 ; weights exponent by 61
00001000 04B9 1043 W_TERM_WEIGHT = ^X1000 ; weights exponent by 32
00000200 04B9 1044 G0_TERM_WEIGHT = ^X0200 ; Weights exponent by 32 in G_floating
00004000 04B9 1045 W_MAX_WEIGHT = ^X4000 ; maximum unbiased exponent in F and G
04B9 1046
00000019 04B9 1047 W_ADJUST = ^X19 ; Used to locate binary point in
04B9 1048 MTH$AL_4_OV_PI table
04B9 1049 : D_2_TO_64: .QUAD ^X0000000000006080 ; 2^64
04B9 1050
04B9 1051 G_2_TO_64:
04B9 1052 .QUAD ^X0000000000004410 ; 2^64
04C1 1053
04C1 1054 REDUCE_LARGE:
04C1 1055
04C1 1056 : The first step is to obtain the location of the binary point in the representation of C =  $2^{(n-p)} * (4/\pi)$  in two parts - the number of longwords from
04C1 1057 : the start and the number of bits from the most significant bit of the next
04C1 1058 : longword. Also K =  $2^{24}*f$  must be obtained.
04C1 1059
04C1 1060
52 50 F9 8F 9C 04C1 1061 ROTL #7, R0, R2 ; Get exp field in first byte of R2
52 52 19 A2 04C6 1062 SUBW #W_ADJUST, R2 ; Unbias exp and adjust for leading
04C9 1063 zeroes. R2 = location of binary
04C9 1064 point
53 52 FD 8F 9C 04C9 1065 ROTL #3, R2, R3 ; Divide R2 by 32 and mul by 4 to get
53 FFFFFFE3 8F CA 04CE 1066 BICL #^XFFFFFE3, R3 R3 = # of longwords (in bytes) to
04D5 1067
51 00000000'EF DE 04D5 1068 MOVAL MTH$AL_4_OV_PI, R1 binary point.
04DC 1069
52 51 53 C2 04DC 1070 SUBL R3, R1 ; Get base address of MTH$AL_4_OV_PI
52 E0 8F 8A 04DF 1071 BICB #^XE0, R2 table
04E3 1072
50 7F80 8F AA 04E3 1073 BICW #^X7F80, R0 ; R1 points to 1st quadword of interest
50 4C00 8F A8 04E8 1074 BISW #^X4C00, R0 ; R2(7:0) = # of bits within longword
50 50 4A 04ED 1075 CVTFL R0, R0 ; Clear exponent field
04F0 1076
04F0 1077
04F0 1078 : The next step is to generate an approximation to C, call it C'' to be used
04F0 1079 : in computing x*4/pi. C'' will consist of the first three integer bits of
04F0 1080 : C and the first 61 fraction bits of C. These bits will be obtained from a

```

				04F0 1081 : constant stored in the integer array MTH\$AL_4_OV_PI.
				04F0 1082 :
				04F0 1083 : NOTE: The ASHQ, ADDL, and MULL instructions in the follow sections may
				04F0 1084 : result in an integer overflow trap. The overflow incurred is intentional,
				04F0 1085 : so that the IV bit must be turned off. The IV bit is not restored until
				04F0 1086 : after all of the necessary fraction bits have been generated.
				04F0 1087 :
6E	FFFFFD	7E	DC	04F0 1088 : MOVPSL -(SP)
		8F	CA	04F2 1089 : BICL #^C<PSLSM_IV>, (SP) ; Put current PSL on stack
		20	B9	04F9 1090 : BICPSW #PSLSM_IV ; (SP) = current IV bit
				04FB 1091 : Clear integer overflow bit
				04FB 1092 :
				04FB 1093 : Get C'' = C(0):C(1) in R3/R4. C(0) and C(1) are unsigned integers generated
				04FB 1094 : from the binary representation of C. The high three bits of C(0)
				04FB 1095 : are the the first three bits to the left of the binary point of C. Bits 28:0
				04FB 1096 : of C(0) and bits 31:0 are the first 61 bits to the right of the binary point
				04FB 1097 : of C.
				04FB 1098 :
				04FB 1099 :
7E	61	52	79	04FB 1100 : ASHQ R2, (R1), -(SP) ; Shift the proper quadword so that
				04FF 1101 : (SP)+4 has C(0) in it
	51	04	C2	04FF 1102 : SUBL #4, R1 ; R1 points to next quadword in
				0502 1103 : MTH\$AL_4_OV_PI table
53	61	52	79	0502 1104 : ASHQ R2, (R1), R3 ; Shift quadword to R3 so that R4 has
				0506 1105 : C(1) in it
				0506 1106 :
				0506 1107 : Generate the low 64 bits of the product K*C'' = L. This product is
				0506 1108 : equivalent to multiplying K times C'' modulo 8. The result of the
				0506 1109 : product is in R3:R4 with bits 63:61 the octant bits, 60:24 the valid fraction
				0506 1110 : bits of the product and 23:0 non-valid fraction bits that will be used later
				0506 1111 : if more fraction bits need to be generated.
				0506 1112 :
				0506 1113 :
53	00	54	50	7A 0506 1114 : EMUL R0, R4, #0, R3 ; R3/R4 = K*C(1)
		03	18	050B 1115 : BGEQ 1\$; Check if R4 was negative
	54	50	C0	050D 1116 : ADDL R0, R4 ; Adjust for signed mult. error
6E	50	04	AE	0510 1117 : 1\$: MULL3 4(SP), R0, (SP) ; (SP) = Low 32 bits of K*C(0)
	54	6E	C0	0515 1118 : ADDL (SP), R4 ; R3/R4 = L = low order bits of K*C''
				0518 1119 :
				0518 1120 :
				0518 1121 : At this point there may or may not be enough valid bits in R3/R4 to generate
				0518 1122 : Y. If the first 7 fraction bits are all 1's or 0's, there a possibility of
				0518 1123 : loss of significance when computing Y. Consequently, we must check for loss
				0518 1124 : of significance before converting R3/R4 to Y and I.
				0518 1125 :
				0518 1126 :
				0518 1127 :
6E	54	00200000	8F	C1 0518 1128 : ADDL3 #^X200000, R4, (SP) ; If the first 8 fraction bits are 1's
6E	3FC00000	8F	D3 0520 1129 : BITL #^X3FC00000, (SP) ; and the reduced arg = 1-f or the	
	47	12	0527 1130 : BNEQ CONVERT ; first 8 bit are 0 and the reduced	
				0529 1131 : arg = f, then (and only then) bits
				0529 1132 : 29:22 are 0 and significance will
				0529 1133 : be lost.
				0529 1134 :
				0529 1135 :
				0529 1136 : More bits need to be generated to cover the loss of significance. There are
				0529 1137 : not enough registers to hold all the potential extra bits, so that the bits

```

      0529 1138 ; already generated must be put on the stack.
      0529 1139 ;
      0529 1140 ;
      5E 04 C2 0529 1141      SUBL #4, SP          ; Allocate an extra longwords on stack
      6E 53 7D 052C 1142      MOVQ R3, (SP)       ; for later use
      000006A0'EF 16 052F 1143      JSB GEN_MORE_BITS ; Put L onto stack
      6E 54 C0 0535 1145      ADDL R4, (SP)       ; R3/R4 contain 56 additional bits
      04 05 1E 0538 1146      BCC 2$              ; Add new high order bits to old low
      04 AE D6 053A 1149      INCL 4(SP)        ; order bits
      2C 11 053D 1150      BRB 4$              ; Check for possible carry and adjust
      14 04 AE 1D E0 053F 1152      BRW LEADING_ZEROS ; high order bits accordingly
      0544 1153 2$: BBS #29, 4(SP), 3$ ; A carry indicates a minor loss of
      0544 1154 ; Lost significance due to leading zeros ; significance has occurred
      0544 1155 ; Check for leading ones or zeros
      54 04 AE 14 00 EA 0544 1156      FFS #0, #20, 4(SP), R4 ; If at least one bit is set. This
      1F 12 054A 1157      BNEQ 4$              ; means lost significance was minor.
      6E 1FFFFFFF 8F D1 054C 1158      CMPL #^X1FFFFFFF, (SP) ; One of the three high bits is set.
      16 15 0553 1159      BLEQ 4$              ; lost significance was minor.
      00EC 31 0555 1160      BRW LEADING_ZEROS
      0558 1161 ;
      0558 1162 ;
      0558 1163 ; Lost significance due to leading ones
      54 04 AE 14 00 EB 0558 1164 3$: FFC #0, #20, 4(SP), R4 ; If at least one bit is clear. This
      0B 12 055E 1165      BNEQ 4$              ; means lost significance was minor.
      6E E0000000 8F D1 0560 1166      CMPL #^XE0000000, (SP) ; One of the three high bits is clear.
      02 1E 0567 1167      BGEOU 4$             ; lost significance was minor.
      50 11 0569 1168      BRB LEADING_ONES
      056B 1169 ;
      056B 1170 ;
      53 6E 7D 056B 1171 4$: MOVQ (SP), R3 ; Move bits to registers
      8E D5 056E 1172      TSTL (SP)+        ; Remove extra longword from stack
      0570 1173 ;
      0570 1174 ;
      0570 1175 CONVERT:
      5E 08 C0 0570 1176      ADDL #8, SP          ; Clear stack
      0573 1177 ;
      0573 1178 ; Convert bit string to double precision reduced argument.
      0573 1179 ;
      0573 1180 ;
      0573 1181 ; Isolate octant bits and convert low order bits to double precision.
      0573 1182 ;
      52 54 03 1D EF 0573 1183      EXTZV #29, #3, R4, R2 ; R2 = octant bits
      0578 1184 ;
      0578 1185 ;
      0578 1186 ;
      50 53 4EFD 0578 1187      CVTLG R3, R0          ; R0/R1 = 2^61*FL0
      02 18 057C 1188      BGEOQ 1$             ; Check for signed conversion error
      54 D6 057E 1189      INCL R4             ; and adjust R5 accordingly
      1C 52 E9 0580 1190 1$: BLBC R2, 2$ ; Check for odd or even octant bits
      0583 1191 ;
      0583 1192 ; Octant bits are odd. Reduced argument equals 1 - f.
      0583 1193 ;
      54 E0000000 8F C8 0583 1194      BISL #^XE0000000, R4 ; Set octant bits
  
```

```

      058A 1195 : CVTLD R4, R3 ; R3/R4 = -2^29*(1 - FHI)
      058A 1196 : ADDW2 #W_TERM_WEIGHT, R3
      058A 1197 : CVTLG R4, R3 ; R3/R4 = -2^29*(1 - FHI)
      058A 1198 : ADDW2 #GW_TERM_WEIGHT, R3 ; R3/R4 = -2^61*(1 - FHI)
      53 53 54 4EFD 058A 1199 : ADDD R0, R3
      0200 8F A0 058E 1200 : SUBW #^X9E80, R3
      0593 1201 : ADDG2 R0, R3 ; R3/R4 = -2^61*(1 - FHI - FLO)
      0593 1202 : SUBW2 #^X83D0, R3 ; R3/R4 = 1 - F
      0593 1203 : BRW RESTORE
      53 53 50 40FD 0593 1204 : Octant bits are even. Reduced argument equals f
      83D0 8F A2 0597 1205 : 059F 1208
      00FB 31 059C 1206 : 059F 1210 : 059F 1211 2$: BICL #^XE0000000, R4 ; Clear octant bits
      05A6 1212 : 05A6 1213 : CVTLD R4, R3 ; R3/R4 = 2^29*FHI
      05A6 1214 : ADDW #W_TERM_WEIGHT, R3 ; R3/R4 = 2^61*FHI
      05A6 1215 : ADDD R0, R3 ; R3/R4 = 2^61*(FHI + FLO)
      05A6 1216 : SUBW #^X1E80, R3 ; R3/R4 = F = FHI + FLO
      05A6 1217 : CVTLG R4, R3 ; R3/R4 = 2^29*FHI
      53 53 54 4EFD 05A6 1218 : 05AA 1219 : ADDW #GW_TERM_WEIGHT, R3 ; R3/R4 = 2^61*FHI
      0200 8F A0 05AF 1220 : ADDG2 R0, R3 ; R3/R4 = 2^61*(FHI + FLO)
      53 03D0 8F A2 05B3 1221 : SUBW2 #^X03D0, R3 ; R3/R4 = F = FHI + FLO
      00DF 31 05B8 1222 : BRW RESTORE
      05BB 1223
      05BB 1224
      05BB 1225
      05BB 1226 : At this point it has been determined that there is a major loss of
      05BB 1227 : significance and the processing begins a looping phase. Each iteration of
      05BB 1228 : the loop will generate additional extra bits of K*C until enough significant
      05BB 1229 : bits to compute Y are available. During this time the stack will contain
      05BB 1230 : four longwords allocated as follows:
      05BB 1231 : four longwords allocated as follows:
      05BB 1232 : SP ----> L2 : L1 and L2 are the last 2 longwords worth of
      05BB 1233 : L1 : significant bits generated
      05BB 1234 : L0 : 1st three bits of L0 are the octant bits
      05BB 1235 : W : Weighting factor - keeps track of the
      05BB 1236 : number of iterations that have failed
      05BB 1237 : to produce enough significant bits.
      05BB 1238 :
      05BB 1239 :
      05BB 1240 :
      05BB 1241 LEADING_ONES:
      05BB 1242 :
      05BB 1243 : If processing continues here it is known that the loss of significance is due
      05BB 1244 : to a string of leading ones.
      05BB 1245 :
      OC AE 7E 53 D0 05BB 1246 : MOVL R3, -(SP) ; Put L2 onto stack
      00001E80 8F D0 05BE 1247 : MOVL #L_INT_WEIGHT, 12(SP) ; W = exp bias for last longword
      05C6 1248 : of the product K*C
      05C6 1249 :
      04 AE FFE00000 8F D1 05C6 1250 LOOP_1: CMPL #^XFFE00000, 4(SP) ; Check L2 for enough significant bits
      3F 1A 05CE 1251 BGTRU CONVERT_1B ; Enough bits. Convert to floating.
  
```

000006A0'EF	16	05D0	1252		JSB GEN_MORE_BITS	R3/R4 contains an 56 additional 56 bits of K*C'
54 6E C0	05D6	1253			ADDL (SP), R4	Add low order bits of existing product, L2, to high order bits of new product, R4.
08 0B 1E	05D9	1254			BCC 1\$	Check for carry on previous ADDL
04 AE D6	05DB	1255			INCL 4(SP)	Add carry into L1
06 0E 1E	05DE	1256			BCC 1\$	Check for carry on previous ADDL
08 AE D6	05E0	1257			INCL 8(SP)	Add carry into L0. If carry propagates to L0 enough fraction bits are available to produce equivalent argument
0090 31	05E3	1258			BRW CONVERT_0A	
		05E6	1260			
		05E6	1261			
		05E6	1262			
		05E6	1263			
		05E6	1264			
04 AE FFFFFFFF	8F	D1	05E6	1265	1\$: CMPL #1, 4(SP)	Check for L1 all 1's
1C	1A	05EE	1266		BGTRU CONVERT_1A	Not all 1's. Enough precision bits to compute Y
FFC8 0C AE 1000	8F	6E 53	7D	05F0	MOVQ R3, (SP)	Compress representation of L
4000 8F	3D	05F3	1268		ACBW #W_MAX_WEIGHT, #W_TERM_WEIGHT, 12(SP), LOOP 1	
		05FE	1269			Increment weighting factor. If weighting factor is greater than 128 then no more bits need to be generated.
		05FE	1270			
		05FE	1271			
		05FE	1272			
		05FE	1273			
		05FE	1274			
		05FE	1275			
		05FE	1276			The weighting factor is greater than 128. This means that the reduced argument is either not distinguishable from 1 or too small to be represented in F_format (i.e. underflow.). Zero is returned in R3 for the reduced argument to signal this occurrence. Note that under these conditions the correct function value is one of the values 0, +/-1. The correct choice is determined by the calling program based on the octant bits returned in R1.
		05FE	1277			
		05FE	1278			
		05FE	1279			
		05FE	1280			
		05FE	1281			
		05FE	1282			
		05FE	1283			
52 08 AE	03 53	D4	05FE	1284	CLRL R3	Reduced argument is zero
	5E 1D	EF	0600	1285	EXTZV #29, #3, 8(SP), R2	R2 = octant bits
	10 C0		0606	1286	ADDL #16, SP	
	008E 31		0609	1287	BRW RESTORE	Clear stack
			060C	1288		
			060C	1289	CONVERT_1A:	
	6E 53	D0	060C	1290	MOVL R3, (SP)	Put L2 onto stack
			060F	1291	CONVERT_1B:	
			060F	1292	:	
			060F	1293	CVTLD (SP)+, R0	$R0/R1 = 2^W \cdot L2$
			060F	1294	CVTLG (SP)+, R0	$R0/R1 = 2^W \cdot L2$
50 8E 4EFD	02 18	0613	1295		BGEQ 1\$	Check for signed conversion error
	6E D6	0615	1296		INCL (SP)	and adjust L1 accordingly.
		0617	1297			
		0617	1298			
		0617	1299	1\$: CVTLD (SP), R3		
		0617	1300			
		0617	1301			
53 6E 4EFD	07 18	0617	1302	1\$: CVTLG (SP), R3	Convert L1 to double and check for signed conversion error.	
		061B	1303	BGEQ 2\$		
		061D	1304	:		
		061D	1305	ADDW #^X9000, R3		
		061D	1306	:		
53 8200 8F	OC A0	061D	1307	ADDW #^X8200, R3	$R3/R4 = 2^W \cdot (2^{64} - L1)$	
	11	0622	1308	BRB 3\$		

			0624 1309			
			0624 1310			
			0624 1311	2\$: ADDW #1000, R3		
			0624 1312	SUBD3 R3, D_2_TO_64, R3		
			0624 1313			
			0624 1314			
53	53 FE8A CF	0200 8F A0	0624 1315	2\$: ADDW #^X0200, R3	; R3/R4 = 2^W*L1	
			0629 1316	SUBG3 R3, G_2_TO_64, R3	; R3/R4 = 2^W*(2^64 - L1)	
			0630 1317			
			0630 1318			
			0630 1319	3\$: SUBD R0, R3	; R3/R4 = 2^W*(2^64 - L1 - L2)	
			0630 1320	EXTZV #29, #3, (SP)+, R2	; R2 = octant bits	
			0630 1321	SUBL (SP)+, R3	; R3/R4 = Y	
			0630 1322	BRW RESTORE		
			0630 1323			
			0630 1324			
52	BE 53 03 7E	50 1D 8E	0630 1325	3\$: SUBG2 R0, R3	; R3/R4 = 2^W*(2^64 - L1 - L2)	
		42FD EF	0634 1326	EXTZV #29, #3, (SP)+, R2	; R2 = octant bits	
		4EFD	0639 1327	CVTLG (SP)+, -(SP)		
	53 0056	42FD 31	063D 1328	SUBG2 (SP)+, R3	; R3/R4 = Y	
			0641 1329	BRW RESTORE		
			0644 1330			
			0644 1331			
			0644 1332	LEADING_ZEROS:		
			0644 1333			
			0644 1334			
			0644 1335	If processing continues here it is known that the loss of significance is due		
			0644 1336	to a string of leading zeros. Note that it is known that the loop for		
			0644 1337	leading zeros will terminate before an underflow condition occurs so that the		
			0644 1338	loop does not include a test for underflow.		
			0644 1339			
0C AE	00001E80	7E 53 03	0644 1340	MOVL R3, -(SP)	; Put L2 onto stack	
		BF D0	0647 1341	MOVL #L_INT_WEIGHT, 12(SP)	; W = exp bias for last longword	
			064F 1342		of the product K*C'	
			064F 1343			
04 AE	001FFFFF	8F D1	064F 1344	LOOP_0: CMPL #^X001FFFFF, 4(SP)	; Check L2 for enough fraction bits	
		20 19	0657 1345	BLSS CONVERT_0B	Enough bits. Convert to floating	
	000006A0'EF	EF 16	0659 1346	JSB GEN_MORE_BITS	R3/R4 contain an additional 56 bits	
			065F 1347		of K*C'.	
54	6E C0	065F 1348		ADDL (SP), R4	Add low order bits of existing	
		0662 1349			product, (SP), to high order bits	
		0662 1350			of new product, R4.	
			0662 1351	BCC 1\$	Check for carry on previous ADDL	
04 AE	03 D6	0664 1352		INCL 4(SP)	Add in carry. Note: High bits of R4	
		0667 1353			are 0 so no carry possible on INCL	
	54 0B	D5 12	0667 1354	1\$: TSTL R4	Check for all 0's	
		0669 1355		BNEQ CONVERT_0A	Not all 0's. Enough precision bits.	
0C AE	6E 53	7D A0	066B 1356	MOVQ R3, (SP)	Compress representation of L	
	1000 8F	066E 1357		ADDW #W_TERM_WEIGHT, 12(SP)	Increment weighting factor.	
	D9	11 0674	0674 1358	BRB LOOP_0		
		0676 1359				
			0676 1360	CONVERT_0A:		
6E	54 D0	0676 1361		MOVL R4, (SP)	; Put L2 onto stack	
		0679 1362	CONVERT_0B:			
		0679 1363				
		0679 1364		CVTLD (SP), R0	; R0/R1 = 2^W*L2	
		0679 1365				

50 6E 4EFD 0679 1366 02 18 067D 1367 6E D6 067F 1368 0681 1369 0681 1370 0681 1371 : 1\$: CVTLD (SP), R3 0681 1372 : ADDW #W_TERM_WEIGHT, R3 0681 1373 : ADDD R0, R3 0681 1374 : EXTZV #29, #3, (SP)+, R2 0681 1375 : SUBL (SP)+, R3 0681 1376 53 53 6E 4EFD 0681 1377 1\$: CVTLG (SP), R3 0200 8F A0 0685 1378 ADDW2 #GW_TERM_WEIGHT, R3 53 50 40FD 068A 1379 ADDG2 R0, R3 03 1D EF 068E 1380 EXTZV #29, #3, (SP)+, R2 7E 8E 4EFD 0693 1381 CVTLG (SP)+, -(SP) 53 8E C2 0697 1382 SUBL2 (SP)+, R3 069A 1383 069A 1384 069A 1385 RESTORE: 5E 6E B8 069A 1386 BISPSW (SP) 04 C0 069C 1387 ADDL #4, SP 05 069F 1388 RSB 06A0 1389 06A0 1390 GEN_MORE_BITS: 06A0 1391 06A0 1392 : 06A0 1393 : This subroutine generates 56 extra fraction bits and stores them in R3/R4 06A0 1394 : 06A0 1395 : 53 53 51 04 C2 06A0 1396 SUBL #4, R1 61 52 79 06A3 1397 ASHQ R2, (R1), R3 54 50 7A 06A7 1398 EMUL R0, R4, #0, R3 03 18 06AC 1399 BGEQ 1\$: 54 50 C0 06AE 1400 ADDL R0, R4 05 06B1 1401 1\$: RSB 06B2 1402 06B2 1403 06B2 1404 06B2 1405	; R0/R1 = 2^W*L2 ; If L2 is negative, adjust L1 to ; to reflect signed rather than ; unsigned conversion ; Convert L1 to double ; R3/R4 = 2^W*L1 ; R3/R4 = 2^W*(L1 + L2) ; R2 = octant bits ; R3/R4 = Y ; Convert L1 to double ; R3/R4 = 2^W*L1 ; R3/R4 = 2^W*(L1 + L2) ; R2 = octant bits ; R3/R4 = Y ; Restore IV bit and exit ; Remove mask from stack ; Adjust pointer to next quadword ; Put next coefficient in R4 ; Generate the fraction bits in R3/R4 ; Check for signed arithmetic error ; Negative factor, adjust R4
---	---

```

06B2 1407 .SBTTL REDUCE_DEGREES
06B2 1408
06B2 1409 ; This routine assumes that the absolute value of the argument is in R0.
06B2 1410 ; The reduction process is performed in two stages. The first stage of
06B2 1411 ; the reduction reduces the argument modulo 360 to a value less than 2^24,
06B2 1412 ; and the second stage reduces the argument to less than 45.
06B2 1413
06B2 1414 ; Constants used in this reduction:
06B2 1415 :
06B2 1416
06B2 1417 POWER_MOD_360_0: ; Powers of 2 modulo 360 for t1 = 0
06B2 1418 .WORD 1, 2, 4, 8
06B2 1419 .WORD 16, 32, 64, 128
06C2 1420 .WORD 256, 152, 304, 248
06CA 1421
06CA 1422 POWER_MOD_360_1: ; Powers of 2 modulo 360 for t1 <> 0
06CA 1423 .WORD 136, 272, 184, 8
06D2 1424 .WORD 16, 32, 64, 128
06DA 1425 .WORD 256, 152, 304, 248
06E2 1426
06E2 1427 REDUCE_DEGREES:
50 4C80 8F B1 06E2 1428 CMPW #^X4C80, R0 ; Compare |x| with 2^24
45 14 06E7 1429 BGTR LAST_STEP ; Branch to special logic for med arg
06E9 1430
06E9 1431
06E9 1432 ; It is assumed here that the argument is greater than 2^24.
06E9 1433
06E9 1434 ; The argument is reduced as follows:
06E9 1435 Let x = 2^t*f, where t > 24 and 1/2 <= f < 1. And let J = 2^24*f =
06E9 1436 = 2^15*j1 + j2 and K = 2^(t-24). Since 2^15 = 8 modulo 360, we have that
06E9 1437 J = 8*j1 + j2 modulo 360. Now let t' = t - 24 = 12*t1 + t2. Note that
06E9 1438 (2^12)^2 = 2^24 = (2^9)*(2^15) = (2^9)*(2^3) = 2^12 modulo 360. Hence,
06E9 1439 if t1 is not zero, K = 2^t' = 2^(12*t1+t2) = [2^(12*t1)]*[2^t2] =
06E9 1440 [2^12]*[2^t2] = 136*2^t2 modulo 360. For t1 = 0, K = 2^t2. Consequently
06E9 1441 define K' congruent to 2^t2 if t1=0 and congruent to 136*2^t2 otherwise,
06E9 1442 where 0 <= K' < 360. Then x' = K'*(8*j1 + j2) is congruent to x modulo
06E9 1443 360 and x' < 2^24
06E9 1444
06E9 1445
51 50 00007F80 8F CB 06E9 1446 BICL3 #^X7F80, R0, R1 ; R1 = J
51 4C00 8F A8 06F1 1447 BISW #^X4C00, R1 ; R0 = t'*2^7
50 51 C2 06F6 1448 SUBL R1, R0
06F9 1449
52 51 7FFF0000 8F CB 06F9 1450 BICL3 #^X7FFF0000, R1, R2 ; R2 = J1*2^15
51 52 42 0701 1451 SUBF R2, R1 ; R1 = J2
52 0600 8F A2 0704 1452 SUBW #^X600, R2 ; R2 = J1
51 52 40 0709 1453 ADDF R2, R1 ; R1 = 8*j1 + j2 = J modulo 360
070C 1454
50 50 F9 8F 9C 070C 1455 ROTL #-7, R0, R0 ; R0 = t'
52 50 0C 87 0711 1456 DIVB3 #12, R0, R2 ; R2 = t1
52 0C 84 0715 1457 MULB #12, R2 ; R2 = 12*t1
50 52 82 0718 1458 SUBB R2, R0 ; R0 = t2
071B 1459
52 95 071B 1460 TSTB R2 ; Check for t1 = 0 and choose K'
07 12 071D 1461 BNEQ 1$ ; accordingly
50 8F AF40 4D 071F 1462 CVTWF POWER_MOD_360_0[R0], R0 ; R0 = K'
05 11 0724 1463 BRB 2$
```

50 A0 AF40 4D 0726 1464 1\$: 50 51 44 072B 1465 2\$: 072E 1466 072E 1467 072E 1468 LAST_STEP: 072E 1469 : 072E 1470 : Argument reduction scheme for arguments with absolute value less than 2^24 072E 1471 : 072E 1472 : The reduced argument Y is computed as follows: 072E 1473 : Let I = int(X/45) 072E 1474 : if I is even 072E 1475 : then Y = X - 45*I 072E 1476 : else Y = (I+1)*45 - x 072E 1477 : 53 50 B6 8F 0B603DB6 8F 54 072E 1478 EMODF #LF_1_OV_45, #X_1_OV_45, R0, R3, R4 0738 0739 1479 : R3 = I = integer part of x /45 54 16 53 E9 0739 1480 BLBC R3, EVEN : Branch if octant bits are even 53 01 C1 073C 1481 ADDL3 #1, R3, R4 : R4 = I + 1 54 54 54 4E 0740 1482 CVTLF R4, R4 : 00004334 8F 44 0743 1483 MULF #LF_45, R4 : R4 = 45*(I+1) 54 50 42 074A 1484 SUBF R0, R4 : R4 = Y 53 F8 8F 8A 074D 1485 BICB #^XF8, R3 : Save only last three octant bits 05 0751 1486 RSB : 0752 1487 : 54 53 4E 0752 1488 EVEN: CVTLF R3, R4 : R4 = I 0000C334 8F 44 0755 1489 MULF #LF_M45, R4 : R4 = -45*I 54 50 40 075C 1490 ADDF R0, R4 : R4 = Y 53 F8 8F 8A 075F 1491 BICB #^XF8, R3 : Save only last three octant bits 05 0763 1492 RSB : 0764 1493

```

0764 1495
0764 1496 .SBTTL RADIANT_POLYNOMIALS ; Polynomials for arguments in radians
0764 1497
0764 1498
0764 1499
0764 1500
0764 1501 ; Polynomial evaluation for COS(Y) for Y in radians
0764 1502 ;
0764 1503
0764 1504 P_COS_R:
53 8000 8F AA 0764 1505 BICW #^X8000, R3 ; R3/R4 = |Y|
0769 1506 CMPW #^X4000, R3 ; Compare 1/2 with |Y|
0769 1507 ; Using CMPL for G does not yield enough signif fraction bits
0769 1508
50 4000 53 33FD 0769 1509 CVTGF R3, R0 ; Need to conform to old code.
2E B1 076D 1510 CMPW #^X4000, R0 ; Compare 1/2 with |Y|
0772 1511 BGEQ LEQL_HALF ; Sufficient overhang is available
0774 1512 NEEDS_DOUBLE:
0774 1513 ;
0774 1514 MULD R3, R3 ; R3/R4 = Y*Y
0774 1515 ;
0774 1516
53 53 44FD 0774 1517 MULG2 R3, R3 ; R3/R4 = Y*Y
0778 1518
0778 1519 NEEDS_DOUBLE_SINCOS:
0778 1520 ; Save high half of Y*Y
0778 1521 PUSHC R3
0778 1522 MOVQ R3, -(SP)
0778 1523 BICW2 #^X1FFF, R4 ; Save G rep of Y^2
0780 1524 CVTGF R3, R3 ; Zero the rounding bit (bit 12) and
0784 1525 ; all lower order bits of word 4.
0784 1526 POLYF R3, #COSLENR2-1, COSTBR2; R0 = Y^4*Q(Y*Y)
078A 1527 SUBL3 #^X80 (SP)+, R3 ; R3/R4 = Y*Y/2
078A 1528 SUBD #1, R3 ; R3/R4 = Y^2/2 - 1
078A 1529 SUBD R3, R0 ; R0/R1 = COS(Y)
078A 1530 CVTDF R0, R0 ; R0 = COS(Y)
078A 1531
53 8E 10 C3 078A 1532 SUBL3 #^X0010, (SP)+, R3 ; R3/R4 = Y^2
54 8E D0 078E 1533 MOVL (SP)+, R4 ; R3/R4 = Y^2/2 - 1
53 08 42FD 0791 1534 SUBG2 #1, R3
50 50 99FD 0795 1535 CVTGF R0, R0
50 53 42FD 0799 1536 SUBG2 R3, R0 ; R0/R1 = COS(Y)
50 50 33FD 079D 1537 CVTGF R0, R0 ; R0 = COS(Y)
05 07A1 1538 RSB
07A2 1539 LEQL_HALF:
07A2 1540 CVTDF R3, R4 ; R4 = |Y|
07A2 1541 CVTGF R3, R4 ; R4 = |Y|
07A6 1542 MULF R4, R4 ; R4 = Y*Y
07A9 1543 POLYF R4, #COSLENR1-1, COSTBR1; R0 = COS(Y)
05 07AF 1544 RSB
07B0 1545
07B0 1546
07B0 1547 ; Polynomial evaluation for -COS(Y)
07B0 1548
07B0 1549
07B0 1550
07B0 1551 N_COS_R:

```

53	8000	8F	AA	07B0	1552	BICW #^X8000, R3	: R3/R4 = Y	
				07B5	1553	CMPW #^X4000, R3	: Compare 1/2 with Y	
				07B5	1554	: Using CMPL for G does not yield enough significant fraction bits		
				07B5	1555			
50	4000	53	33FD	07B5	1556	CVTGF R3, R0	: Need to conform to old code.	
				07B9	1557	CMPW #^X4000, R0	: Compare 1/2 with Y	
				07BE	1558	BGEQ 1\$: Sufficient overhang is available	
				07C0	1559	MULD R3, R3	: R3/R4 = Y*Y	
				07C0	1560	PUSHL R3	: Save high half of Y*Y	
				07C0	1561			
54	1FFF	8F	AA	07C0	1562	MULG2 R3, R3	: R3/R4 = Y*Y	
	7E	53	7D	07C4	1563	MOVQ R3, -(SP)	: Save G rep of Y^2	
				07C7	1564	BICW2 #^X1FFF, R4	: Zero the rounding bit (bit 12) and all lower order bits of word 4.	
				07CC	1565			
				07CC	1566	CVTGF R3, R3	: R3 = truncated Y^2	
F85A	CF	04	53	55	07D0	1567		
				07D0	1568	POLYF R3, #COSLENR2-1, COSTBR2; R0 = Y^4*Q(Y*Y)		
				07D6	1569			
				07D6	1570	SUBL3 #^X80, (SP)+, R3	: R3/R4 = Y*Y/2	
				07D6	1571	SUBD #1, R3	: R3/R4 = -(1 - Y^2/2)	
				07D6	1572	SUBD R0, R3	: R3/R4 = -COS(Y)	
				07D6	1573	CVTDF R3, R0	: R0 = -COS(Y)	
				07D6	1574			
53	8E	10	C3	07D6	1575	SUBL3 #^X0010, (SP)+, R3		
54	8E	D0		07DA	1577	MOVL (SP)+, R4	: R3/R4 = Y^2/2	
53	08	42FD		07DD	1578	SUBG2 #1, R3	: R3/R4 = -(1 - Y^2/2)	
50	50	99FD		07E1	1579	CVTGF R0, R0		
53	50	42FD		07E5	1580	SUBG2 R0, R3	: R3/R4 = -COS(Y)	
50	53	33FD		07E9	1581	CVTGF R3, R0	: R0 = -COS(Y)	
				07ED	1582			
		05		07ED	1583	RSB		
				07EE	1584			
				07EE	1585			
				07EE	1586	; 1\$: CVTDF R3, R4	: R4 = Y	
				07EE	1587			
F825	CF	54	53	33FD	07EE	1588	1\$: CVTGF R3, R4	: R4 = Y
	54	54	44	07F2	1589	MULF R4, R4	: R4 = Y*Y	
	03	54	55	07F5	1590	POLYF R4, #COSLENR1-1, COSTBR1; R0 = COS(Y)		
50	8000	8F	AC	07FB	1591	XORW #^X8000, R0	: R0 = -COS(Y)	
			05	0800	1592	RSB		
				0801	1593			
				0801	1594			
				0801	1595	: Polynomial evaluation for -SIN(Y)		
				0801	1596			
				0801	1597			
53	8000	8F	AC	0801	1598	N_SIN_R:		
				0801	1599	XORW #^X8000, R3		
				0806	1600			
				0806	1601			
				0806	1602	: Polynomial evaluation for SIN(Y)		
				0806	1603			
				0806	1604			
				0806	1605	P_SIN_R:		
				0806	1606	PUSHL R3	: Save high half of Y	
54	1FFF	7E	53	7D	0806	1607	MOVQ R3, -(SP)	: Save R3 R4 = Y
				0809	1608	BICW2 #^X1FFF, R4	: Zero the rounding bit (bit 12) and	

	082D	1629	.SBTTL CYCLE_POLYNOMIALS	: Polynomials for arguments in cycles	
	082D	1630			
	082D	1631			
	082D	1632			
	082D	1633			
	082D	1634	: Polynomial evaluation for COS(Y) for Y in cycles		
	082D	1635			
	082D	1636			
	082D	1637	P_COS_C:		
53	00000000 5F304004	8F 51FD	082D 1638	: CMPF #LF_2_OV_PI, R3	: Compare 2/pi with Y
	2E 18	082D 1639	CMPG #G_2_OV_PI, R3	: Compare 2/pi with Y	
		0839 1640	BGEQ LEQL_2_OV_PI	: Sufficient overhang is available	
		083B 1641	MULD R3, R3	: R3/R4 = Y*Y	
		083B 1642	PUSHL R3	: Save high half of Y*Y	
		083B 1643			
54	53 53 44FD	083B 1644	MULG2 R3, R3	R3/R4 = Y*Y	
	7E 53 7D	083F 1645	MOVQ R3, -(SP)	SAVE Y^2	
	1FFF 8F AA	0842 1646	BICW2 #^X1FFF, R4	Zero the rounding bit (bit 12) and all lower order bits of word 4.	
F817 CF	53 53 33FD	0847 1647	CVTGF R3, R3	R3 = Y^2 truncated.	
	04 53 55	084B 1648	POLYF R3, #COSLENC2-1, COSTBC2	R0 = Y^4*Q(Y*Y)	
		0851 1650	SUBL3 #^X100, (SP)+, R3	R3/R4 = Y*Y/4	
		0851 1651	SUBD #1, R3	R3/R4 = Y^2/4 - 1	
		0851 1652	SUBD R3, R0	R0/R1 = COS(Y)	
		0851 1653	CVTDF R0, R0	R0 = COS(Y)	
		0851 1654			
53	8E 20 C3	0851 1655	SUBL3 #^X0020, (SP)+, R3		
54	8E D0	0855 1656	MOVL (SP)+, R4	R3/R4 = Y^2/4	
53	08 42FD	0858 1657	SUBG2 #1, R3	R3/R4 = Y^2/4 - 1	
50	50 99FD	085C 1658	CVTGF R0, R0		
50	53 42FD	0860 1659	SUBG2 R3, R0	R0/R1 = COS(Y)	
50	50 33FD	0864 1660	CVTGF R0, R0	R0 = COS(Y)	
	05	0868 1661	RSB		
		0869 1662	LEQL_2_OV_PI:		
		0869 1663	: CVTDF R3, R4	R4 = Y	
F7DE CF	53 33FD	0869 1664	CVTGF R3, R4	R4 = Y	
	54 44	086D 1665	MULF R4, R4	R4 = Y*Y	
	04 54 55	0870 1666	POLYF R4, #COSLENC1-1, COSTBC1	R0 = COS(Y)	
	05	0876 1667	RSB		
		0877 1668			
		0877 1669			
		0877 1670			
		0877 1671	: Polynomial evaluation for -COS(Y)		
		0877 1672			
		0877 1673			
		0877 1674	N_COS_C:		
53	00000000 5F304004	8F 51FD	0877 1675	: CMPF #LF_2_OV_PI, R3	: Compare 2/pi with Y
	2E 18	0877 1676	CMPG #G_2_OV_PI, R3	: Compare 2/pi with Y	
		0883 1677	BGEQ 1S	: Sufficient overhang is available	
		0885 1678	MULD R3, R3	: R3/R4 = Y*Y	
		0885 1679	PUSHL R3	: Save high half of Y*Y	
		0885 1680			
54	53 53 44FD	0885 1681	MULG2 R3, R3	R3/R4 = Y^2	
	7E 53 7D	0889 1682	MOVQ R3, -(SP)	Save Y^2	
	1FFF 8F AA	088C 1683	BICW2 #^X1FFF, R4	Zero the rounding bit (bit 12) and all lower order bits of word 4.	
	53 53 33FD	0891 1684	CVTGF R3, R3	R3 = truncated Y^2.	
		0891 1685			

F7CD CF 04 53 55 0895 1686 : POLYF R3, #COSLENC2-1, COSTBC2; R0 = Y^4*Q(Y*Y)
 089B 1687 : SUBL3 #^X100, (SP)+, R3 ; R3/R4 = Y*Y/4
 089B 1688 : SUBD #1, R3 ; R3/R4 = -(1 - Y^2/4)
 089B 1689 : SUBD R0, R3 ; R3/R4 = -COS(Y)
 089B 1690 : CVTDF R3, R0 ; R0 = -COS(Y)

53 8E 20 C3 089B 1691 :
 54 8E D0 089F 1693 :
 53 08 42FD 08A2 1694 :
 50 50 99FD 08A6 1695 :
 53 50 42FD 08AA 1696 :
 50 53 33FD 08AE 1697 :
 05 08B2 1698 :
 08B3 1699 :
 08B3 1700 :
 54 53 33FD 08B3 1701 :
 08B7 1702 :
 F794 CF 54 54 44 08B7 1703 :
 50 04 54 55 08BA 1704 :
 8000 8F AC 08C0 1705 :
 05 08C5 1706 :
 08C6 1707 :
 08C6 1708 :
 08C6 1709 : Polynomial evaluation for -SIN(Y)
 08C6 1710 :
 08C6 1711 :
 53 8000 8F AC 08C6 1712 N_SIN_C:
 08C6 1713 XORW #^X8000, R3 ; R3/R4 = - Y
 08CB 1714 :
 08CB 1715 :
 08CB 1716 : Polynomial evaluation for SIN(Y)
 08CB 1717 :
 08CB 1718 :
 08CB 1719 P_SIN_C:
 7E 53 7D 08CB 1720 : MOVQ R3, -(SP) ; Save argument
 08CE 1721 : MULF R3, R3
 08CE 1722 :
 54 1FFF 8F AA 08CE 1723 : BICW2 #^X1FFF, R4 ; Zero the rounding bit (bit 12) and
 08D3 1724 : all lower order bits of word 4.
 53 53 33FD 08D3 1725 : CVTGF R3, R3 ; R3 = Truncated Y
 53 DD 08D7 1726 : PUSHL R3 ; Save Y truncated

F79A CF 53 53 44 08D9 1728 :
 03 53 55 08DC 1729 :
 08E2 1730 :
 08E2 1731 :
 08E2 1732 :
 08E2 1733 :
 08E2 1734 :
 08E2 1735 :
 08E2 1736 :
 08E2 1737 :
 50 8E 44 08E2 1738 :
 54 6E 20 C3 08E5 1738 :
 54 04 AE D0 08E9 1739 :
 53 8E 42FD 08ED 1740 :
 50 50 99FD 08F1 1741 :
 50 53 42FD 08F5 1742 :
 MULF2 (SP)+, R0 ; R0 = truncated Y * P(Y^2)
 SUBL3 #^X0020, (SP), R3 ; Form the quantity
 MOVL 4(SP), R4 ; R3/R4 = Y/4
 SUBG2 (SP)+, R3 ; R3/R4 = 3/4*Y
 CVTGF R0, R0 ; R0/R1 = SIN(Y)
 SUBG2 R3, R0 ; R0 = SIN(Y)

UVX\$SINCOS
2-005

: Floating Point Sine, Cosine and Sincos ^{K 4}
CYCLE_POLYNOMIALS ; Polynomials for argu 16-SEP-1984 02:06:12 VAX/VMS Macro V04-00
6-SEP-1984 11:29:16 [MTHRTL.SRC]UVXSINCOS.MAR;1 Page 40 (24)

50 50 33FD 08F9 1743 CVTGF R0, R0 ; R0 = SIN(Y)
05 08FD 1744 RSB
08FE 1745

				08FE	1747	.SBTTL DEGREE_POLYNOMIALS		
				08FE	1748			
				08FE	1749			
				08FE	1750 P_COS_D:			
54	2EE142E5	8F	51	08FE	1751	CMPF #LF_90_OV_PI, R4	; Compare 90/pi with Y	
			51	0905	1752	BGEQ 2\$; Double precision isn't needed	
				0907	1753	CVTFD R4, R3	; R3/R4 = Y	
				0907	1754	MULD R3, R3	; R3/R4 = Y*Y	
				0907	1755	PUSHL R3	; Save high half Y*Y	
				0907	1756			
		53	54 99FD	0907	1757	CVTFG R4, R3	; R3/R4 = Y	
		53	53 44FD	090B	1758	MULG2 R3, R3	; R3/R4 = Y^2	
		7E	53 7D	090F	1759	MOVQ R3, -(SP)	; Save R3/R4 = Y^2	
54	1FFF	8F	AA	0912	1760	BICW2 #^X1FFF, R4	; Zero the rounding bit (bit 12) and all lower order bits of word 4.	
		53	53 33FD	0917	1761		; R3 = Y^2 truncated.	
				091B	1762	CVTGF R3, R3		
				091B	1763			
F77B	CF	04	53 55	091B	1764	POLYF R3, #COSDLN2, COSDTB2	; R0 = r(Y*Y) - NOTE: POLYF leaves R1=0	
		50	50 99FD	0921	1765	CVTFG R0, R0	; Need R0 in G floating format.	
				0925	1766	SUBL3 #^X680, (SP)+, R3	; R3/R4 = Y*Y/2^13	
				0925	1767	CMPL #^X1000, R4	; Check for loss of significance when	
53	8E	000000D0	8F	C3	0925	1768		
		54	8E D0	092D	1770	SUBL3 #^X0000, (SP)+, R3	; R3/R4 = Y^2/2^13	
54	00000200	8F	D1	0930	1771	MOVL (SP)+, R4	; Check for loss of significance when	
				0937	1772	CMPL #^X0200, R4	; Subtracting 1 and adjust.	
				0937	1773		; D_float used ^X1000. This converts to	
				0937	1774		; ^X0200 in G_floating.	
				0937	1775			
		7E	53 7D	0937	1776	BGTRU 1\$		
				0939	1777	MOVQ R3, -(SP)	; (SP) = Y^2/2^13	
				093C	1778	BICL #^XFFFF0000, R4	; R3/R4 = high bits of Y^2/2^13	
				093C	1779	SUBD R3, (SP)	; (SP) = Low bits of Y^2/2^13	
				093C	1780	ADDD (SP)+, R0	; R0/R1 = r(Y^2) + low bits of Y^2/2^13	
				093C	1781	SUBD #1, R3	; R3/R4 = -(1 - Y*Y/2^13)	
				093C	1782	SUBD R3, R0	; R0/R1 = COS(Y)	
				093C	1783	CVTDF R0, R0	; R0 = COS(Y)	
				093C	1784			
54	1FFF0000	8F	CA	093C	1785	BICL2 #^X1FFF0000, R4	; R3/R4 = high 40 bits of Y^2/2^13	
		6E	53 42FD	0943	1786	SUBG2 R3, (SP)	; (SP) = low 13 bits of Y^2/2^13	
		50	8E 40FD	0947	1787	ADDG2 (SP)+, R0	; R0/R1 = r(Y^2) + low 13 bits of Y^2/2^13	
		53	08 42FD	094B	1788	1\$: SUBG2 #1, R3	; R3/R4 = -(1 - Y^2/2^13)	
		50	53 42FD	094F	1789	SUBG2 R3, R0	; R0/R1 = COS(Y)	
		50	50 33FD	0953	1790	CVTGF R0, R0	; R0 = COS(Y)	
				0957	1791			
				0957	1792	RSB		
				0958	1793			
F72B	CF	54	54 44	0958	1794	2\$: MULF2 R4, R4	; R4 = Y*Y	
		03	54 55	0958	1795	POLYF R4, #COSDLN1, COSDTB1	; R0 = COS(Y)	
			05	0961	1796	RSB		
				0962	1797			
				0962	1798			
54	2EE142E5	8F	51	0962	1799 N_COS_D:	CMPF #LF_90_OV_PI, R4	; Compare 90/pi with Y	
		51	18	0969	1800	BGEQ 2\$; Double precision isn't needed	
				096B	1801	CVTFD R4, R3	; R3/R4 = Y	
				096B	1802	MULD R3, R3	; R3/R4 = Y*Y	
				096B	1803			

				PUSHL R3	: Save high half of Y*Y
				CVTGF R4, R3	: R3/R4 = Y
				MULG2 R3, R3	: R3/R4 = Y^2
				MOVQ R3, -(SP)	: Save R3/R4 = Y^2
				BICW2 #^X1FFF, R4	: Zero the rounding bit (bit 12) and all lower order bits of word 4.
				CVTGF R3, R3	: R3 = Y^2 truncated.
				POLYF R3, #COSDLN2, COSDTB2	: R0 = r(Y*Y)
				CVTGF R0, R0	: Need R0 in G_floating format.
				SUBL3 #^X680, (SP)+, R3	: R3/R4 = Y*Y/2^13
				CMPL #^X1000, R4	: Check for loss of significance when
				SUBL3 #^X0000, (SP)+, R3	: R3/R4 = Y^2/2^13
				MOVL (SP)+, R4	: Check for loss of significance when
				CMPL #^X0200, R4	: subtracting 1 and adjust. D_float used ^X1000. This converts to ^X0200 in G_floating.
				BGTRU 1\$: D
				MOVQ R3, -(SP)	: (SP) = Y^2/2^13
				BICL #^XFFFF0000, R4	: R3/R4 = high bits of Y^2/2^13
				SUBD R3, (SP)	: (SP) = Low bits of Y^2/2^13
				ADDD (SP)+, R0	: R0/R1 = r(Y^2) + low bits of Y^2/2^13
				SUBD #1, R3	: R3/R4 = -(1 - Y*Y/2^13)
				SUBD R0, R3	: R3/R4 = -COS(Y)
				CVTDF R3, R0	: R0 = -COS(Y)
				BICL2 #^X1FFF0000, R4	: R3/R4 = high 40 bits of Y^2/2^13
				SUBG2 R3, (SP)	: (SP) = low 13 bits of Y^2/2^13
				ADDG2 (SP)+, R0	: R0/R1 = r(Y^2) + low 13 bits of Y^2/2^13
				SUBG2 #1, R3	: R3/R4 = -(1 - Y^2/2^13)
				SUBG2 R0, R3	: R3/R4 = -COS(Y)
				CVTGF R3, R0	: R0 = -COS(Y)
				RSB	
				MULF2 R4, R4	: R4 = Y*Y
				POLYF R4, #COSDLN1, COSDTB1	: R0 = COS(Y)
				XORW #^X8000, R0	: R0 = -COS(Y)
				RSB	
				N_SIN_D:	
				MNEGF R4, R4	: R4 = -Y
				P_SIN_D:	
				MULF3 R4, R4, R0	: R0 = Y*Y
				BEQL RETURN	
				POLYF R0, #SINDLN, SINDTB	: R0 = P(Y*Y)
				MULF2 R4, R0	: R0 = Y*P(Y*Y)
				SUBW2 #^X300, R4	: R4 = Y*2^-6
				ADDF2 R4, R0	: R0 = SIN(Y)
				RETURN: RSB	
				09E5 1859	
				09E6 1860	

				09E6 1862			
				09E6 1863			
				09E6 1864	.SBTTL DEGENERATE_SOLUTIONS		
				09E6 1865			
				09E6 1866 P_ONE:			
50 08	50	09E6 1867		MOVF #1, R0			
	05	09E9 1868		RSB			
		09EA 1869					
		09EA 1870					
50 0000C080 8F	50	09EA 1871 N_ONE:		MOVF #-1, R0			
	05	09F1 1872		RSB			
		09F2 1873					
		09F2 1874					
		09F2 1875					
		09F2 1876 UNFL:					
		09F2 1877 :					
		09F2 1878 ; Underflow; if user has FU set, signal error. Always return 0.0					
		09F2 1879 :					
00000000'GF 00	52 DC	09F2 1880		MOVPSL R2			
52 04 50	FB E9	09F4 1881		CALLS #0, G^MTH\$\$JACKET_TST			
	AD 3C	09FB 1882		BLBC R0, 10\$			
	50 D4	09FE 1883		MOVZWL SF\$W_SAVE_PSW(FP), R2			
		0A02 1884	10\$:	CLRL R0			
OD 52 06	E1 0A04	1885					
	6E DD	0A08 1886					
7E 00'8F	9A	0A0A 1887		BBC #6, R2, 20\$			
		0A0E 1888		PUSHL (SP)			
		0A0E 1889		MOVZBL #MTH\$K_FLOUNDMAT, -(SP)			
00000000'GF 02	FB 0A0E	1890					
	05 0A15	1891					
		0A16 1892		CALLS #2, G^MTH\$\$\$SIGNAL			
		0A16 1893	20\$:	RSB			
		0A16 1894					
		0A16 1895		.END			

; R2 = user's or jacket routine's PSL
 ; R0 = TRUE if JSB from jacket routine
 ; branch if user did JSB
 ; get user PSL saved by CALL
 ; R0 = result. LIB\$SIGNAL will save in
 ; CHF\$L_MCH_R0/R1 so any handler can
 ; fixup
 ; has user enabled floating underflow?
 ; yes, return PC from special routine
 ; trap code for hardware floating
 ; underflow convert to MTH\$_FLOUNDMAT
 ; (32-bit VAX-11 exception code)
 ; signal (condition, PC)
 ; return

CONVERT	00000570	R	02	L_SIN	000002AA	R	02
CONVERT_0A	00000676	R	02	MTHSSJACKET_HND	*****	X	02
CONVERT_0B	00000679	R	02	MTHSSJACKET_TST	*****	X	00
CONVERT_1A	0000060C	R	02	MTHSSSIGNAL	*****	X	00
CONVERT_1B	0000060F	R	02	MTHSAL_4_OV_PI	*****	X	00
COSD	= 0000000C			MTHSCOS	000000F0	RG	02
COSDLN1	= 00000003			MTH\$COSD	00000134	RG	02
COSDLN2	= 00000004			MTH\$COSD_R4	00000412	RG	02
COSDTB1	0000008C	R	02	MTHSCOS_R4	000002D6	RG	02
COSDTB2	0000009C	R	02	MTHSK_FOUNDMAT	*****	X	00
COSINE	= 0000000C			MTHSSIN	000000DC	RG	02
COSLENC1	= 00000005			MTHSSINCOS	000000C0	RG	02
COSLENC2	= 00000005			MTHSSINCOSD	00000104	RG	02
COSLENR1	= 00000004			MTHSSINCOSD_R5	0000035E	RG	02
COSLENR2	= 00000005			MTHSSINCOS_R5	00000148	RG	02
COSTBC1	00000054	R	02	MTHSSIND	00000120	RG	02
COSTBC2	00000068	R	02	MTHSSIND_R4	000003AE	RG	02
COSTBR1	00000020	R	02	MTHSSIN_R4	00000221	RG	02
COSTBR2	00000030	R	02	M_COS	000002F5	R	02
DEGENERATE_CASE_COS	0000034A	R	02	M_SIN	0000024C	R	02
DEGENERATE_CASE_SIN	000002C2	R	02	NEEDS_DOUBLE	00000774	R	02
EVAL_COSD	00000426	R	02	NEEDS_DOUBLE_SINCOS	00000778	R	02
EVAL_SIND	000003CE	R	02	NEG_SIND	000003BC	R	02
EVEN	00000752	R	02	N_COS_C	00000877	R	02
GEN_MORE_BITS	000006A0	R	02	N_COS_D	00000962	R	02
GW_TERM_WEIGHT	= 00000200			N_COS_R	000007B0	R	02
G_2_OV_PI	= 5F304004			N_ONE	000009EA	R	02
G_2_PI	00000018	R	02	N_SIN_C	000008C6	R	02
G_2_TO_64	000004B9	R	02	N_SIN_D	000009CB	R	02
G_3_PI_OV_2	00000010	R	02	N_SIN_R	00000801	R	02
G_PI	00000008	R	02	POS_SIN	00000234	R	02
G_PI_OV_2	00000000	R	02	POS_SINCOS	0000015C	R	02
LARGE_COS	0000032C	R	02	POS_SIND	000003C1	R	02
LARGE_SIN	000002A4	R	02	POWER_MOD_360_0	000006B2	R	02
LARGE_SINCOS	000001FD	R	02	POWER_MOD_360_1	000006CA	R	02
LAST_STEP	0000072E	R	02	PSL\$M_IV	= 00000020		
LEADING_ONES	000005BB	R	02	P_COS_C	0000082D	R	02
LEADING_ZEROS	00000644	R	02	P_COS_D	000008FE	R	02
LEQL_2_OV_PI	00000869	R	02	P_COS_R	00000764	R	02
LEQL_HALF	000007A2	R	02	P_ONE	000009E6	R	02
LF_1_OV_45	= 0B603DB6			P_SIN_C	000008CB	R	02
LF_3_PI_OV_4	= CBE44116			P_SIN_D	000009CE	R	02
LF_45	= 00004334			P_SIN_R	00000806	R	02
LF_5_PI_OV_4	= 53D1417B			REDUCE_DEGREES	000006E2	R	02
LF_7_PI_OV_4	= EDDF41AF			REDUCE_LARGE	000004C1	R	02
LF_90_OV_PI	= 2EE142E5			REDUCE_MEDIUM	0000044D	R	02
LF_9_PI_OV_4	= 31D641E2			RESTORE	0000069A	R	02
LF_CONVERT	= A3513BEF			RETURN	000009E5	R	02
LF_M45	= 0000C334			SFSW_SAVE_PSW	= 00000004		
LF_PI_OV_4	= OFDB4049			SIN	0000022F	R	02
LF_SMALLD	= 2EE13D65			SINCOS	00000157	R	02
LF_SMALLEST_DEG	= 2EE10365			SINCOSD	00000371	R	02
LONG	= 00000004			SIND	= 00000008		
LOOP_0	0000064F	R	02	SINDLN	= 00000003		
LOOP_1	000005C6	R	02	SINDTB	000000B0	R	02
L_COS	00000332	R	02	SINE	= 00000008		
L_INT_WEIGHT	= 00001EB0			SINLENC	= 00000004		

```

SINLENR          = 00000004
SINTBC           = 0000007C R  02
SINTBR           = 00000044 R  02
SMALL_COS        = 00000309 R  02
SMALL_COSD       = 0000043A R  02
SMALL_SIN         = 00000260 R  02
SMALL_SINCOS      = 00000194 R  02
SMALL_SINCOSD     = 00000396 R  02
SMALL_SIND        = 000003E2 R  02
UNFL             = 000009F2 R  02
W_45              = 00004334
= 00000019
= 00004000
= 00001000
= 00000004
= 000000B6

```

```

+-----+
! Psect synopsis !
+-----+

```

PSECT name	Allocation	PSECT No.	Attributes
ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_MTHSCODE	00000A16 (2582.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

```

+-----+
! Performance indicators !
+-----+

```

Phase	Page faults	CPU Time	Elapsed Time
Initialization	36	00:00:00.10	00:00:00.47
Command processing	126	00:00:00.69	00:00:04.65
Pass 1	211	00:00:05.61	00:00:14.68
Symbol table sort	0	00:00:00.24	00:00:00.36
Pass 2	327	00:00:03.96	00:00:12.02
Symbol table output	17	00:00:00.15	00:00:00.97
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	721	00:00:10.80	00:00:33.19

The working set limit was 1500 pages.
 37124 bytes (73 pages) of virtual memory were used to buffer the intermediate code.
 There were 20 pages of symbol table space allocated to hold 196 non-local and 49 local symbols.
 1955 source lines were read in Pass 1, producing 34 object records in Pass 2.
 10 pages of virtual memory were used to define 9 macros.

Macro library name

\$_\$255\$DUA28:[SYSLIB]STARLET.MLB;2

Macros defined

5

```

+-----+
! Macro library statistics !
+-----+

```

UVX\$SINCOS
VAX-11 Macro Run Statistics

; Floating Point Sine, Cosine and Sincos 16-SEP-1984 02:06:12 VAX/VMS Macro V04-00
0 5
6-SEP-1984 11:29:16 [MTHRTL.SRC]UVXSINCOS.MAR;1 Page 46 (27)

131 GETS were required to define 5 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:UVXSINCOS/OBJ=OBJ\$:UVXSINCOS MSRC\$:MTHJACKET/UPDATE=(ENH\$:MTHJACKET)+MS

0266 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

