```
00000000                                  CPU  "SYMPL64_IL.TBL"
00000000                                  HOF  "bin32"
00000000                                  WDLN 8
                               ; version 2.01   May 22, 2018
                               ; Author:  Jerry D. Harthcock
                               ; Copyright (C) 2018.  All rights reserved.

                               ;private dword storage
00000000         =             bitbucket:  EQU     0x0000                   ;this dword location is reserved.  Don't use it for anything because a lot of garbage can wind up here
00000008         =             work_1:     EQU     0x0008
00000010         =             work_2:     EQU     0x0010
00000018         =             work_3:     EQU     0x0018
00000020         =             capt0_save: EQU     0x0020                  ;alternate delayed exception capture register 0 save location
00000028         =             capt1_save: EQU     0x0028                  ;alternate delayed exception capture register 1 save location
00000030         =             capt2_save: EQU     0x0030                  ;alternate delayed exception capture register 2 save location
00000038         =             capt3_save: EQU     0x0038                  ;alternate delayed exception capture register 3 save location

                               ;for private storage of parameters for 3D transform
00000040         =             ext_vect_start: EQU     0x0040              ;location in external memory where the first triangle x1 may be found
00000048         =             triangles:  EQU     0x0048                  ;storage location of number of triangles in this thread's list to process

                               ;dword storage locations for parameters so it will be easy to change to/from double precision
00000050         =             scaleX:     EQU     0x0050                  ;scale factor X axis
00000058         =             scaleY:     EQU     0x0058                  ;scale factor Y axis
00000060         =             scaleZ:     EQU     0x0060                  ;scale factor Z axis
00000068         =             transX:     EQU     0x0068                  ;translate amount X axis
00000070         =             transY:     EQU     0x0070                  ;translate amount Y axis
00000078         =             transZ:     EQU     0x0078                  ;translate amount Z axis

                               ;word (32-bit) storage for x1, y1, z1, x2, y2, z2, x3, y3, z3 for assembling half-word pieces from little endian external memory file of .stl object
00000080         =             x1:         EQU     0x0080
00000084         =             y1:         EQU     0x0084
00000088         =             z1:         EQU     0x0088
0000008C         =             x2:         EQU     0x008C
00000090         =             Y2:         EQU     0x0090
00000094         =             z2:         EQU     0x0094
00000098         =             x3:         EQU     0x0098
0000009C         =             y3:         EQU     0x009C
000000A0         =             z3:         EQU     0x00A0

000000B0         =             XCUs:       EQU     0x00B0                  ;number of XCUs in this implementation
000000B8         =             result_buf: EQU     0x00B8                  ;this is start of the buffer where results are stored and then read back out to external memory when processing is complete
000000C0         =             remainder_push:  EQU     0x00C0
000000C8         =             remainder_pull:  EQU     0x00C8

0000E5E0         =             sin_thetaX: EQU     sind.0                  ;sine of theta X for rotate X
0000E5C0         =             cos_thetaX: EQU     cosd.0                  ;cosine of theta X for rotate X
0000E5E8         =             sin_thetaY: EQU     sind.1                  ;sine of theta Y for rotate Y
0000E5C8         =             cos_thetaY: EQU     cosd.1                  ;cosine of theta Y for rotate Y
0000E5F0         =             sin_thetaZ: EQU     sind.2                  ;sine of theta X for rotate Z
0000E5D0         =             cos_thetaZ: EQU     cosd.2                  ;cosine of theta X for rotate Z

80000000         =             PROG_START: EQU     0x80000000              ;CPU and XCU program memory can be indirectly accessed starting here
00100000         =             STL_START:  EQU     0x00100000              ;start location of .stl file in external memory space
00010000         =             buf_START:  EQU     0x00010000              ;start location of internal tri-port indirectly addressable RAM, which is where the first triangle x1 will be pushed

00000000                                   org     0x0
```

```
00000000 0000000000000100      Constants:  DFL     0, load_vects                   ;entrypoint for this program

00000001 000000000000021A      prog_len:   DFL     0, progend

                               ;parameters for this particular 3D transform test run
00000002                       xform_3axis_parameters:

00000002 000000000000001D      rotx:       dfl     0, 29                          ;rotate around x axis in integer degrees
00000003 000000000000002C      roty:       dfl     0, 44                          ;rotate around y axis in integer degrees
00000004 000000000000004B      rotz:       dfl     0, 75                          ;rotate around z axis in integer degrees
00000005 0000000040000000      scal_x:     dff     0, 2.0                         ;scale X axis amount real
00000006 0000000040000000      scal_y:     dff     0, 2.0                         ;scale y axis amount real
00000007 0000000040100000      scal_z:     dff     0, 2.25                        ;scale Z axis amount real
00000008 0000000040980000      trans_x:    dff     0, 4.75                        ;translate on X axis amount real
00000009 000000004077AE14      trans_y:    dff     0, 3.87                        ;translate on Y axis amount real
0000000A 00000000400F2B02      trans_z:    dff     0, 2.237                       ;translate on Z axis amount real

                               ;           type    dest = OP:(type:srcA, type:srcB)

00000100                                   org     0x00000100                        ;default interrupt vector locations
00000100                       load_vects:
00000100 12FEF80000020172                  uh      NMI_VECT = uh:#NMI_               ;load of interrupt vectors for faster interrupt response
00000101 12FEF000000201A7                  uh      IRQ_VECT = uh:#IRQ_               ;these registers are presently not visible to app s/w
00000102 12FEE80000020176                  uh      INV_VECT = uh:#INV_
00000103 12FEE0000002017F                  uh      DIVx0_VECT = uh:#DIVx0_
00000104 12FED80000020188                  uh      OVFL_VECT = uh:#OVFL_
00000105 12FED00000020191                  uh      UNFL_VECT = uh:#UNFL_
00000106 12FEC8000002019A                  uh      INEXT_VECT = uh:#INEXT_

00000107 12FF8C0000020C00                          enableInt

00000108 12FF8C0000020300      done:               setDone
00000109 14FF68000004EA60                  uw      TIMER = uw:#60000                 ;load time-out timer with sufficient time to process before timeout
0000010A 14FFA04FF887C000                          BREAK                             ;just sit here and wait for interrupt or pushXCU PC

0000010B 1CFFA04FF887C002      begin:              GOSUB threadStart
0000010C 14FFA04FF887FFFC                          goto done

0000010D                       threadStart:

0000010D 0D7FC74FF9000000                  sw      *SP--[8] = uw:PC_COPY             ;save return address

0000010E 12FF8C0000020200                           clearDone                        ;to signal host CPU or XCU has started (ie, not done)

0000010F 04FFB0400B800000                  uw       AR0    = uw:result_buf           ;load AR0 with pointer to source/destination internal result buffer for XCU X1 of first triangle
00000110 04FFB8400B800000                  uw       AR1   =  uw:result_buf

00000111 24E5E04000200000                  fs       sin_thetaX = sind:(uw:@rotx)     ;calculate sine of theta X and save
00000112 24E5C04000200000                  fs       cos_thetaX = cosd:(uw:@rotx)     ;calculate cosine of theta X and save
00000113 24E5E84000300000                  fs       sin_thetaY = sind:(uw:@roty)     ;calculate sine of theta Y and save
00000114 24E5C84000300000                  fs       cos_thetaY = cosd:(uw:@roty)     ;calculate cosine of theta Y and save
00000115 24E5F04000400000                  fs       sin_thetaZ = sind:(uw:@rotz)     ;calculate sine of theta Z and save
00000116 24E5D04000400000                  fs       cos_thetaZ = cosd:(uw:@rotz)     ;calculate cosine of theta Z and save

00000117 2400504000500000                  fs       scaleX = fs:@scal_x              ;save scale X factor
00000118 2400584000600000                  fs       scaleY = fs:@scal_y              ;save scale Y factor
00000119 2400604000700000                  fs       scaleZ = fs:@scal_z              ;save scale Z factor
0000011A 2400684000800000                  fs       transX = fs:@trans_x             ;save translate X axis amount
0000011B 2400704000900000                  fs       transY = fs:@trans_y             ;save translate Y axis amount
```

```
0000011C 2400784000A00000                  fs       transZ = fs:@trans_z              ;save translate Z axis amount

0000011D 04FF702004800000                         for (LPCNT0 = uh:triangles) (      ;load loop counter 0 with number of triangles

0000011E                        loop:   ;scale on X, Y, Z axis

                                                  ;the following routine performs scaling on all three axis first,
                                                  ;rotate on all three axis second, then translate on all three axis last

                                           ;vertex 1
0000011E 04ED805002040050                  fs        FMUL.0 = multiplication:(fs:*AR0++[4], fs:scaleX)
0000011F 04ED885002040058                  fs        FMUL.1 = multiplication:(fs:*AR0++[4], fs:scaleY)
00000120 04ED905002040060                  fs        FMUL.2 = multiplication:(fs:*AR0++[4], fs:scaleZ)
                                           ;vertex 2
00000121 04ED985002040050                  fs        FMUL.3 = multiplication:(fs:*AR0++[4], fs:scaleX)
00000122 04EDA05002040058                  fs        FMUL.4 = multiplication:(fs:*AR0++[4], fs:scaleY)
00000123 04EDA85002040060                  fs        FMUL.5 = multiplication:(fs:*AR0++[4], fs:scaleZ)
                                           ;vertex 3
00000124 04EDB05002040050                  fs        FMUL.6 = multiplication:(fs:*AR0++[4], fs:scaleX)
00000125 04EDB85002040058                  fs        FMUL.7 = multiplication:(fs:*AR0++[4], fs:scaleY)
00000126 04EDC05002040060                  fs        FMUL.8 = multiplication:(fs:*AR0++[4], fs:scaleZ)


                                ;                    X1 is now in FMUL_0
                                ;                    Y1 is now in FMUL_1
                                ;                    Z1 is now in FMUL_2
                                ;                    X2 is now in FMUL_3
                                ;                    Y2 is now in FMUL_4
                                ;                    Z2 is now in FMUL_5
                                ;                    X3 is now in FMUL_6
                                ;                    Y3 is now in FMUL_7
                                ;                    Z3 is now in FMUL_8

                                  ;rotate around X axis
                                       ;vertex 1
                                           ; (cos(xrot) * Y1) - (sin(xrot) * Z1)
00000127 04EDC84ED884E5C0                  fs        FMUL.9 = multiplication:(fs:FMUL.1, fs:cos_thetaX)      ; FMUL.9 = (cos(xrot) * Y1)
00000128 04EDD04ED904E5E0                  fs        FMUL.10 = multiplication:(fs:FMUL.2, fs:sin_thetaX)     ; FMUL.10 = (sin(xrot) * Z1)
                                           ; (sin(xrot) * Y1) + (cos(xrot) * Z1)
00000129 04EDD84ED884E5E0                  fs        FMUL.11 = multiplication:(fs:FMUL.1, fs:sin_thetaX)     ; FMUL.11 = (sin(xrot) * Y1)
0000012A 04EDE04ED904E5C0                  fs        FMUL.12 = multiplication:(fs:FMUL.2, fs:cos_thetaX)     ; FMUL.12 = (cos(xrot) * Z1)

0000012B 04EE004EDC84EDD0                  fs        FSUB.0 = subtraction:(fs:FMUL.9, fs:FMUL.10)            ; FSUB.0 = (cos(xrot) * Y1) - (sin(xrot) * Z1)
0000012C 04EE804EDD84EDE0                  fs        FADD.0 = addition:(fs:FMUL.11, fs:FMUL.12)              ; FADD.0 = (sin(xrot) * Y1) + (cos(xrot) * Z1)

                                       ;vertex 2
                                           ; (cos(xrot) * Y2) - (sin(xrot) * Z2)
0000012D 04ED884EDA04E5C0                  fs        FMUL.1 = multiplication:(fs:FMUL.4, fs:cos_thetaX)      ; FMUL.1 = (cos(xrot) * Y2)
0000012E 04ED904EDA84E5E0                  fs        FMUL.2 = multiplication:(fs:FMUL.5, fs:sin_thetaX)      ; FMUL.2 = (sin(xrot) * Z2)
                                           ; (sin(xrot) * Y2) + (cos(xrot) * Z2)
0000012F 04EDE84EDA04E5E0                  fs        FMUL.13 = multiplication:(fs:FMUL.4, fs:sin_thetaX)     ; FMUL.13 = (sin(xrot) * Y2)
00000130 04EDF04EDA84E5C0                  fs        FMUL.14 = multiplication:(fs:FMUL.5, fs:cos_thetaX)     ; FMUL.14 = (cos(xrot) * Z2)

00000131 04EE084ED884ED90                  fs        FSUB.1 = subtraction:(fs:FMUL.1, fs:FMUL.2)             ; FSUB.1 = (cos(xrot) * Y2) - (sin(xrot) * Z2)
00000132 04EE884EDE84EDF0                  fs        FADD.1 = addition:(fs:FMUL.13, fs:FMUL.14)              ; FADD.1 = (sin(xrot) * Y2) + (cos(xrot) * Z2)

                                       ;vertex 3
                                           ; (cos(xrot) * Y3) - (sin(xrot) * Z3)
00000133 04EDC84EDB84E5C0                  fs        FMUL.9 = multiplication:(fs:FMUL.7, fs:cos_thetaX)      ; FMUL.9 = (cos(xrot) * Y3)
```

```
00000134 04EDD04EDC04E5E0                   fs        FMUL.10 = multiplication:(fs:FMUL.8, fs:sin_thetaX)     ; FMUL.10 = (sin(xrot) * Z3)
                                            ; (sin(xrot) * Y3) + (cos(xrot) * Z3)
00000135 04EDD84EDB84E5E0                   fs        FMUL.11 = multiplication:(fs:FMUL.7, fs:sin_thetaX)     ; FMUL.11 = (sin(xrot) * Y3)
00000136 04EDE04EDC04E5C0                   fs        FMUL.12 = multiplication:(fs:FMUL.8, fs:cos_thetaX)     ; FMUL.12 = (cos(xrot) * Z3)

00000137 04EE104EDC84EDD0                   fs        FSUB.2 = subtraction:(fs:FMUL.9, fs:FMUL.10)            ; FSUB.2 = (cos(xrot) * Y3) - (sin(xrot) * Z3)
00000138 04EE904EDD84EDE0                   fs        FADD.2 = addition:(fs:FMUL.11, fs:FMUL.12)              ; FADD.2 = (sin(xrot) * Y3) + (cos(xrot) * Z3)

                                            ;         X1 is now in FMUL_0
                                            ;         Y1 is now in FSUB_0
                                            ;         Z1 is now in FADD_0
                                            ;         X2 is now in FMUL_3
                                            ;         Y2 is now in FSUB_1
                                            ;         Z2 is now in FADD_1
                                            ;         X3 is now in FMUL_6
                                            ;         Y3 is now in FSUB_2
                                            ;         Z3 is now in FADD_2

                                  ;rotate around Y axis
                                       ;vertex 1
                                            ; (cos(yrot) * X1) + (sin(yrot) * Z1)
00000139 04ED884ED804E5C8                   fs        FMUL.1 = multiplication:(fs:FMUL.0, fs:cos_thetaY)      ; FMUL.1 = (cos(yrot) * X1)
0000013A 04ED904EE804E5E8                   fs        FMUL.2 = multiplication:(fs:FADD.0, fs:sin_thetaY)      ; FMUL.2 = (sin(yrot) * Z1)
                                            ; (cos(yrot) * Z1) - (sin(yrot) * X1)
0000013B 04EDA04EE804E5C8                   fs        FMUL.4 = multiplication:(fs:FADD.0, fs:cos_thetaY)      ; FMUL.4 = (cos(xrot) * Z1)
0000013C 04EDA84ED804E5E8                   fs        FMUL.5 = multiplication:(fs:FMUL.0, fs:sin_thetaY)      ; FMUL.5 = (sin(xrot) * X1)

0000013D 04EE984ED884ED90                   fs        FADD.3 = addition:(fs:FMUL.1, fs:FMUL.2)                ; FADD.3 = (cos(yrot) * X1) + (sin(yrot) * Z1)
0000013E 04EE184EDA04EDA8                   fs        FSUB.3 = subtraction:(fs:FMUL.4, fs:FMUL.5)             ; FSUB.3 = (cos(yrot) * Z1) - (sin(yrot) * X1)
                                       ;vertex 2
                                            ; (cos(yrot) * X2) + (sin(yrot) * Z2)
0000013F 04EDB84ED984E5C8                   fs        FMUL.7 = multiplication:(fs:FMUL.3, fs:cos_thetaY)      ; FMUL.7 = (cos(yrot) * X2)
00000140 04EDC04EE884E5E8                   fs        FMUL.8 = multiplication:(fs:FADD.1, fs:sin_thetaY)      ; FMUL.8 = (sin(yrot) * Z2)
                                            ; (cos(yrot) * Z2) - (sin(yrot) * X2)
00000141 04EDC84EE884E5C8                   fs        FMUL.9 = multiplication:(fs:FADD.1, fs:cos_thetaY)      ; FMUL.9 = (cos(xrot) * Z2)
00000142 04EDD04ED984E5E8                   fs        FMUL.10 = multiplication:(fs:FMUL.3, fs:sin_thetaY)     ; FMUL.10 = (sin(xrot) * X2)

00000143 04EEA04EDB84EDC0                   fs        FADD.4 = addition:(fs:FMUL.7, fs:FMUL.8)                ; FADD.4 = (cos(yrot) * X2) + (sin(yrot) * Z2)
00000144 04EE204EDC84EDD0                   fs        FSUB.4 = multiplication:(fs:FMUL.9, fs:FMUL.10)         ; FSUB.4 = (cos(yrot) * Z2) - (sin(yrot) * X2)

                                       ;vertex 3
                                            ; (cos(yrot) * X3) + (sin(yrot) * Z3)
00000145 04EDD84EDB04E5C8                   fs        FMUL.11 = multiplication:(fs:FMUL.6, fs:cos_thetaY)     ; FMUL.11 = (cos(yrot) * X3)
00000146 04EDE04EE904E5E8                   fs        FMUL.12 = multiplication:(fs:FADD.2, fs:sin_thetaY)     ; FMUL.12 = (sin(yrot) * Z3)

                                            ; (cos(yrot) * Z3) - (sin(yrot) * X3)
00000147 04EDE84EE904E5C8                   fs        FMUL.13 = multiplication:(fs:FADD.2, fs:cos_thetaY)     ; FMUL.13 = (cos(xrot) * Z3)
00000148 04EDF04EDB04E5E8                   fs        FMUL.14 = multiplication:(fs:FMUL.6, fs:sin_thetaY)     ; FMUL.14 = (sin(xrot) * X3)

00000149 04EEA84EDD84EDE0                   fs        FADD.5 = addition:(fs:FMUL.11, fs:FMUL.12)              ; FADD.5 = (cos(yrot) * X3) + (sin(yrot) * Z3)
0000014A 04EE284EDE84EDF0                   fs        FSUB.5 = subtraction:(fs:FMUL.13, fs:FMUL.14)           ; FSUB.5 = (cos(yrot) * Z3) - (sin(yrot) * X3)

                                            ;         X1 is now in FADD_3
                                            ;         Y1 is now in FSUB_0
                                            ;         Z1 is now in FSUB_3
                                            ;         X2 is now in FADD_4
                                            ;         Y2 is now in FSUB_1
                                            ;         Z2 is now in FSUB_4
                                            ;         X3 is now in FADD_5
```

```
                                             ;         Y3 is now in FSUB_2
                                             ;         Z3 is now in FSUB_5

                                  ;rotate around Z axis
                                       ;vertex 1
                                            ; (cos(zrot) * X1) - (sin(zrot) * Y1)
0000014B 04ED804EE984E5D0                    fs        FMUL.0 = multiplication:(fs:FADD.3, fs:cos_thetaZ)      ; FMUL.0 = (cos(zrot) * X1)
0000014C 04ED884EE004E5F0                    fs        FMUL.1 = multiplication:(fs:FSUB.0, fs:sin_thetaZ)      ; FMUL.1 = (sin(xrot) * Y1)
                                            ; (sin(zrot) * X1) + (cos(zrot) * Y1)
0000014D 04ED904EE984E5F0                    fs        FMUL.2 = multiplication:(fs:FADD.3, fs:sin_thetaZ)      ; FMUL.2 = (sin(xrot) * X1)
0000014E 04ED984EE004E5D0                    fs        FMUL.3 = multiplication:(fs:FSUB.0, fs:cos_thetaZ)      ; FMUL.3 = (cos(xrot) * Y1)

0000014F 04EE304ED804ED88                    fs        FSUB.6 = subtraction:(fs:FMUL.0, fs:FMUL.1)             ; FSUB.6 = (cos(zrot) * X1) - (sin(zrot) * Y1)
00000150 04EEB04ED904ED98                    fs        FADD.6 = addition:(fs:FMUL.2, fs:FMUL.3)                ; FADD.6 = (sin(zrot) * X1) + (cos(zrot) * Y1)

                                       ;vertex 2
                                            ; (cos(zrot) * X2) - (sin(zrot) * Y2)
00000151 04EDA04EEA04E5D0                    fs        FMUL.4 = multiplication:(fs:FADD.4, fs:cos_thetaZ)      ; FMUL.4 = (cos(zrot) * X1)
00000152 04EDA84EE084E5F0                    fs        FMUL.5 = multiplication:(fs:FSUB.1, fs:sin_thetaZ)      ; FMUL.5 = (sin(xrot) * Y1)
                                            ; (sin(zrot) * X2) + (cos(zrot) * Y2)
00000153 04EDB04EEA04E5F0                    fs        FMUL.6 = multiplication:(fs:FADD.4, fs:sin_thetaZ)      ; FMUL.6 = (sin(xrot) * X2)
00000154 04EDB84EE084E5D0                    fs        FMUL.7 = multiplication:(fs:FSUB.1, fs:cos_thetaZ)      ; FMUL.7 = (cos(xrot) * Y2)

00000155 04EE384EDA04EDA8                    fs        FSUB.7 = subtraction:(fs:FMUL.4, fs:FMUL.5)             ; FSUB.7 = (cos(zrot) * X2) - (sin(zrot) * Y2)
00000156 04EEB84EDB04EDB8                    fs        FADD.7 = addition:(fs:FMUL.6, fs:FMUL.7)                ; FADD.7 = (sin(zrot) * X2) + (cos(zrot) * Y2)

                                       ;vertex 3
                                            ; (cos(zrot) * X3) - (sin(zrot) * Y3)
00000157 04EDC04EEA84E5D0                    fs        FMUL.8 = multiplication:(fs:FADD.5, fs:cos_thetaZ)      ; FMUL.8 = (cos(zrot) * X3)
00000158 04EDC84EE104E5F0                    fs        FMUL.9 = multiplication:(fs:FSUB.2, fs:sin_thetaZ)      ; FMUL.9 = (sin(xrot) * Y3)
                                            ; (sin(zrot) * X3) + (cos(zrot) * Y3)
00000159 04EDD04EEA84E5F0                    fs        FMUL.10 = multiplication:(fs:FADD.5, fs:sin_thetaZ)     ; FMUL.10 = (sin(xrot) * X3)
0000015A 04EDD84EE104E5D0                    fs        FMUL.11 = multiplication:(fs:FSUB.2, fs:cos_thetaZ)     ; FMUL.11 = (cos(xrot) * Y3)

0000015B 04EE404EDC04EDC8                    fs        FSUB.8 = subtraction:(fs:FMUL.8, fs:FMUL.9)             ; FSUB.8 = (cos(zrot) * X3) - (sin(zrot) * Y3)
0000015C 04EEC04EDD04EDD8                    fs        FADD.8 = addition:(fs:FMUL.10, fs:FMUL.11)              ; FADD.8 = (sin(zrot) * X3) + (cos(zrot) * Y3)

                                             ;         X1 is now in FSUB.6
                                             ;         Y1 is now in FADD.6
                                             ;         Z1 is now in FSUB.3
                                             ;         X2 is now in FSUB.7
                                             ;         Y2 is now in FADD.7
                                             ;         Z2 is now in FSUB.4
                                             ;         X3 is now in FSUB.8
                                             ;         Y3 is now in FADD.8
                                             ;         Z3 is now in FSUB.5

                                    ;now translate on X, Y, Z axis
                                        ;vertex 1
0000015D 04EE804EE3040068                    fs        FADD.0 = addition:(fs:FSUB.6, fs:transX)
0000015E 04EE884EEB040070                    fs        FADD.1 = addition:(fs:FADD.6, fs:transY)
0000015F 04EE904EE1840078                    fs        FADD.2 = addition:(fs:FSUB.3, fs:transZ)
                                        ;vertex 2
00000160 04EEC84EE3840068                    fs        FADD.9 = addition:(fs:FSUB.7, fs:transX)
00000161 04EED04EEB840070                    fs        FADD.10 = addition:(fs:FADD.7, fs:transY)
00000162 04EED84EE2040078                    fs        FADD.11 = addition:(fs:FSUB.4, fs:transZ)
                                        ;vertex 3
00000163 04EEE04EE4040068                    fs        FADD.12 = addition:(fs:FSUB.8, fs:transX)
00000164 04EEE84EEC040070                    fs        FADD.13 = addition:(fs:FADD.8, fs:transY)
```

```
00000165 04EEF04EE2840078                 fs        FADD.14 = addition:(fs:FSUB.5, fs:transZ)

00000166 0500214EE8000000                 fs        *AR1++[4] = fs:FADD.0         ;copy transformed X1 to alignable memory
00000167 0500214EE8800000                 fs        *AR1++[4] = fs:FADD.1         ;copy transformed Y1 to alignable memory
00000168 0500214EE9000000                 fs        *AR1++[4] = fs:FADD.2         ;copy transformed Z1 to alignable memory
00000169 0500214EEC800000                 fs        *AR1++[4] = fs:FADD.9         ;copy transformed X2 to alignable memory
0000016A 0500214EED000000                 fs        *AR1++[4] = fs:FADD.10        ;copy transformed Y2 to alignable memory
0000016B 0500214EED800000                 fs        *AR1++[4] = fs:FADD.11        ;copy transformed Z2 to alignable memory
0000016C 0500214EEE000000                 fs        *AR1++[4] = fs:FADD.12        ;copy transformed X3 to alignable memory
0000016D 0500214EEE800000                 fs        *AR1++[4] = fs:FADD.13        ;copy transformed Y3 to alignable memory
0000016E 0500214EEF000000                 fs        *AR1++[4] = fs:FADD.14        ;copy transformed Z3 to alignable memory

0000016F 14FFA04FF7043FAF                         NEXT LPCNT0 GOTO: loop)        ;continue until done
00000170 14FFA04FF8878000                         nop
00000171 04FFA85004700000                 uw      PC = uw:*SP++[8]                ;return

                               ; interrupt/exception trap service routines
00000172 0D7FC74FF9000000      NMI_:      sw      *SP--[8] = uw:PC_COPY           ;save return address from non-maskable interrupt (time-out timer in this instance)
00000173 14FF68000004EA60                 uw      TIMER = uw:#60000               ;put a new value in the timer
00000174 14FFA04FF8878000                         nop
00000175 0CFFA85004700000                 sw      PC = uw:*SP++[8]                ;return from interrupt

00000176 0D7FC74FF9000000      INV_:      sw      *SP--[8] = uw:PC_COPY           ;save return address from floating-point invalid operation exception, which is maskable
00000177 0600206FF4000000                 ud      capt0_save = ud:CAPTURE0        ;read out CAPTURE0 register and save it
00000178 0600286FF4800000                 ud      capt1_save = ud:CAPTURE1        ;read out CAPTURE1 register and save it
00000179 0600306FF5000000                 ud      capt2_save = ud:CAPTURE2        ;read out CAPTURE2 register and save it
0000017A 0600386FF5800000                 ud      capt3_save = ud:CAPTURE3        ;read out CAPTURE3 register and save it
0000017B 14FF8B0000000001                         lowerSignals(ub:#invalid)       ;lower invalid signal
0000017C 10FF040000000001                         raiseFlags(ub:#invalid)         ;raise invalid flag
0000017D 14FF68000004EA60                 uw      TIMER = uw:#60000               ;put a new value in the timer
0000017E 0CFFA85004700000                 sw      PC = uw:*SP++[8]                ;return from interrupt

0000017F 0D7FC74FF9000000      DIVx0_:    sw      *SP--[8] = uw:PC_COPY           ;save return address from floating-point divide by 0 exception, which is maskable
00000180 0600206FF4000000                 ud      capt0_save = ud:CAPTURE0        ;read out CAPTURE0 register and save it
00000181 0600286FF4800000                 ud      capt1_save = ud:CAPTURE1        ;read out CAPTURE1 register and save it
00000182 0600306FF5000000                 ud      capt2_save = ud:CAPTURE2        ;read out CAPTURE2 register and save it
00000183 0600386FF5800000                 ud      capt3_save = ud:CAPTURE3        ;read out CAPTURE3 register and save it
00000184 14FF8B0000000002                         lowerSignals(ub:#divByZero)     ;lower divByZero signal
00000185 10FF040000000002                         raiseFlags(ub:#divByZero)       ;raise divByZero flag
00000186 14FF68000004EA60                 uw      TIMER = uw:#60000               ;put a new value in the timer
00000187 0CFFA85004700000                 sw      PC = uw:*SP++[8]                ;return from interrupt

00000188 0D7FC74FF9000000      OVFL_:     sw      *SP--[8] = uw:PC_COPY           ;save return address from floating-point overflow exception, which is maskable
00000189 0600206FF4000000                 ud      capt0_save = ud:CAPTURE0        ;read out CAPTURE0 register and save it
0000018A 0600286FF4800000                 ud      capt1_save = ud:CAPTURE1        ;read out CAPTURE1 register and save it
0000018B 0600306FF5000000                 ud      capt2_save = ud:CAPTURE2        ;read out CAPTURE2 register and save it
0000018C 0600386FF5800000                 ud      capt3_save = ud:CAPTURE3        ;read out CAPTURE3 register and save it
0000018D 14FF8B0000000004                         lowerSignals(ub:#overflow)      ;lower overflow signal
0000018E 10FF040000000004                         raiseFlags(ub:#overflow)        ;raise overflow flag
0000018F 14FF68000004EA60                 uw      TIMER = uw:#60000               ;put a new value in the timer
00000190 0CFFA85004700000                 sw      PC = uw:*SP++[8]                ;return from interrupt

00000191 0D7FC74FF9000000      UNFL_:     sw      *SP--[8] = uw:PC_COPY           ;save return address from floating-point underflow exception, which is maskable
00000192 0600206FF4000000                 ud      capt0_save = ud:CAPTURE0        ;read out CAPTURE0 register and save it
00000193 0600286FF4800000                 ud      capt1_save = ud:CAPTURE1        ;read out CAPTURE1 register and save it
00000194 0600306FF5000000                 ud      capt2_save = ud:CAPTURE2        ;read out CAPTURE2 register and save it
00000195 0600386FF5800000                 ud      capt3_save = ud:CAPTURE3        ;read out CAPTURE3 register and save it
00000196 14FF8B0000000008                         lowerSignals(ub:#underflow)     ;lower underflow signal
00000197 10FF040000000008                         raiseFlags(ub:#underflow)       ;raise underflow flag
```

```
00000198 14FF68000004EA60              uw      TIMER = uw:#60000               ;put a new value in the timer
00000199 0CFFA85004700000              sw      PC = uw:*SP++[8]                ;return from interrupt

0000019A 0D7FC74FF9000000      INEXT_:   sw      *SP--[8] = uw:PC_COPY           ;save return address from floating-point inexact exception, which is maskable
0000019B 0600206FF4000000              ud      capt0_save = ud:CAPTURE0        ;read out CAPTURE0 register and save it
0000019C 0600286FF4800000              ud      capt1_save = ud:CAPTURE1        ;read out CAPTURE1 register and save it
0000019D 0600306FF5000000              ud      capt2_save = ud:CAPTURE2        ;read out CAPTURE2 register and save it
0000019E 0600386FF5800000              ud      capt3_save = ud:CAPTURE3        ;read out CAPTURE3 register and save it
0000019F 14FF8B0000000010                      lowerSignals(ub:#inexact)       ;lower inexact signal
000001A0 10FF040000000010                      raiseFlags(ub:#inexact)         ;raise inexact flag
000001A1 14FF68000004EA60              uw      TIMER = uw:#60000               ;put a new value in the timer
000001A2 0CFFA85004700000              sw      PC = uw:*SP++[8]                ;return from interrupt

000001A3 0D7FC74FF9000000      IRQ_XCU:  sw      *SP--[8] = uw:PC_COPY           ;save return address (general-purpose, maskable interrupt)
000001A4 14FF68000004EA60              uw      TIMER = uw:#60000               ;put a new value in the timer
000001A5 14FFA04FF8878000                      nop
000001A6 0CFFA85004700000              sw      PC = uw:*SP++[8]                ;return from interrupt

000001A7                       thread_end:


000001A7 0D7FC74FF9000000      IRQ_:     sw      *SP--[8] = uw:PC_COPY

000001A8 12FF8C0000020200                      clearDone

000001A9                       push_thread:
000001A9 12FDD0000002FFFF                      forceReset(uh:#{XCU15 | XCU14 | XCU13 | XCU12 | XCU11 | XCU10 | XCU9 | XCU8 | XCU7 | XCU6 | XCU5 | XCU4 | XCU3 | XCU2 | XCU1 | XCU0})
000001AA 12FDD2000002FFFF                      forceBreak(uh:#{XCU15 | XCU14 | XCU13 | XCU12 | XCU11 | XCU10 | XCU9 | XCU8 | XCU7 | XCU6 | XCU5 | XCU4 | XCU3 | XCU2 | XCU1 | XCU0})
000001AB 12FDD00000020000                      forceReset(uh:#0)         ;release all target XCU resets.  Note that releasing reset does not affect forceBreak

                                               ;at this point all XCUs should be in h/w break state doing absolutely nothing

000001AC 24FFD84000100000              uw      AR5 = uw:@prog_len

000001AD 34FFB04080000000              uw      AR0 = uw:#0x80000000                   ;load AR0 with pointer to location of beginning of thread to be pushed into XCU program memories
                                                                                      ;be sure to set MSB of pointer to access program memory indirecly
000001AE 34FFB84080000000              uw      AR1 = uw:#0x80000000                   ;place the thread starting at 0x00000000 in XCU program memory (setting MSB of address)
                                                                                      ;forces data to be written to program memory instead of data memory
000001AF 02FF801800500000                      REPEAT [AR5]                           ;push the the 3D transform thread into each XCU program memory--simultaneously
000001B0 00FDD87000870009                          pushAll ud:*AR1++[1], ud:*AR0++[1] ;the entire thread is pushed into XCU using this instruction sequence

000001B1 34FFB04000100000              uw      AR0 = uw:#STL_START                    ;load AR0 with address of external RAM location where raw STL file begins
000001B2 3400404000100060              uw      ext_vect_start = uw:#STL_START + 96    ;this is the location of the first triangle X1 in external RAM
000001B3 3400B84000010000              uw      result_buf = uw:#buf_START
000001B4 0C00485828000000              sw      triangles = uw:*AR0[80]                ;set destination sign extend bit to signal reverse endian-ness and get number of triangles



000001B5 14FF986FDF0FC003      _16_XCUs:         if (ud:XCU_STATUS_REG:[bit63]==0) GOTO: _8_XCUs     ;test DONE bit for XCU15
000001B6 1000B00000000010              ub      XCUs   = ub:#16
000001B7 14FFA04FF887C00F                      goto push_XCUs

000001B8 14FF986FDF0DC003      _8_XCUs:          if (ud:XCU_STATUS_REG:[bit55]==0) GOTO: _4_XCUs     ;test DONE bit for XCU7
000001B9 1000B00000000008              ub      XCUs   = ub:#8
000001BA 14FFA04FF887C00C                      goto push_XCUs

000001BB 14FF986FDF0CC003      _4_XCUs:          if (ud:XCU_STATUS_REG:[bit51]==0) GOTO: _2_XCUs     ;test DONE bit for XCU3
000001BC 1000B00000000004              ub      XCUs   = ub:#4
000001BD 14FFA04FF887C009                      goto push_XCUs
```

```
000001BE 14FF986FDF0C4003      _2_XCUs:             if (ud:XCU_STATUS_REG:[bit49]==0) GOTO: _1_XCU      ;test DONE bit for XCU1
000001BF 1000B00000000002                   ub      XCUs   = ub:#2
000001C0 14FFA04FF887C006                           goto push_XCUs

000001C1 14FF986FDF0C0003      _1_XCU:              if (ud:XCU_STATUS_REG:[bit48]==0) GOTO: NO_XCUs     ;test DONE bit for XCU0
000001C2 1000B00000000001                   ub      XCUs   = ub:#1
000001C3 14FFA04FF887C003                           goto push_XCUs

000001C4 1000B00000000000      NO_XCUs:     ub      XCUs = ub:#0
000001C5 14FFA04FF887C03E                           GOTO solo_process  ;the CPU has to do the 3D transform solo


000001C6                       push_XCUs:
000001C6 14FFC80000040100                   uw      AR3 = uw:#load_vects                          ;each XCU PC will be initialized to begin executing here
000001C7 14FFD0000004010B                   uw      AR4 = uw:#begin                               ;this is the PC address from which all threads begin processing (ie, exit out of SW break)

000001C8 140008000002FDE0                   uw      work_1 = uh:#{XCU_MON_REQUEST}                ;get base address of pushXCU operator
000001C9 04DE0040008000B0                   uw      add.0  = add:(uw:work_1, ub:XCUs)             ;add number of XCUs to get most significant address +1
000001CA 04DB8040048000B0                   uw      div.0  = div:(uw:triangles, ub:XCUs)          ;div.0 now contains number of triangles per XCU (not counting any remainder)
000001CB 04DC004DB80000B0                   uw      mul.0  = mul:(uw:div.0, ub:XCUs)              ;determine any remainder
000001CC 04DD00400484DC00                   uw      sub.0  = sub:(uw:triangles, uw:mul.0)         ;sub.0 now contains any remainder
000001CD 0400C04DD0000000                   uw      remainder_push = uw:sub.0                     ;copy result of remainder calc into remainder so it can be used later
000001CE 0400C84DD0000000                   uw      remainder_pull = uw:sub.0                     ;copy result of remainder calc into remainder so it can be used later

000001CF 00FDD84FFC84FFA8                           pushAll uw:PC, uw:AR3                         ;preset PCs of all XCUs at once to point to entrypoint of initialization sequence
000001D0 00FDD8400B8400B8                           pushAll uw:result_buf, uw:result_buf          ;push the location of the beginning of XCU input/result buffer

000001D1 04FFC02DE0000000                   uw      AR2    = uh:add.0                             ;current XCU base address for that XCU
000001D2 04FFB84004000000                   uw      AR1 = uw:ext_vect_start                       ;address in external RAM of where the first triangle X1 is located

000001D3 04FF78200B000000                           for (LPCNT1 = uh:XCUs) (                      ;for the number of XCUs ...
000001D4                       push_outer:
000001D4 02000037FFA00000                   uh          0x0000 = uh:*AR2--[1]                     ;bumb by -1 XCU number
000001D5 04FFB0400B800000                   uw          AR0    = uw:result_buf                    ;load AR0 with pointer to destination result buffer for XCU X1 of first triangle

000001D6 14DE084DB8040000                   uw          add.1 = add:(uw:div.0, uw:#0)             ;copy calculated triangles/XCUs  into add.1 for future use

000001D7 10FF89400C020000                               compare(uw:remainder_push, uh:#0x0)       ;see if there was any remainder from original triangles/XCU calculation
000001D8 14FFA04FF8800004                               IF (A==B) GOTO: no_remainder_push         ;if no remainder, skip over a push of one more triangle for the current XCU

000001D9 14DD00400C000001                   uw          sub.0  = sub:(uw:remainder_push, ub:#1)   ;decrement any remainder by 1
000001DA 14DE084DB8040001                   uw          add.1 = add:(uw:div.0, uw:#1)             ;add.1 now contains the number of triangles this particular XCU is to process
000001DB 0400C04DD0000000                   uw          remainder_push = uw:sub.0

000001DC                       no_remainder_push:
000001DC 0100022DE0820048                               pushXCU *AR2++[0]:uh:triangles, uh:add.1 ;poke the triangle batch size for this XCU into its "trangles" location

000001DD 04FF702DE0800000                               for (LPCNT0 = uh:add.1) (                 ;for the number of triangles per XCU ...
000001DE 12FF800000020011      push_inner:                  REPEAT    uh:#17                      ;push 18 half-words into target XCU (for a total of 9 32-bit floats per triangle)
000001DF 0900023001130010                                       pushXCU.endi *AR2++[0]:uh:*AR0++[2], uh:*AR1++[2]  ;reverse endian-ness just before push (AR2 contains the current XCU number)
000001E0 0200003007100000                   uh              0x0000  = uh:*AR1++[14]               ;bump source pointer by 14 to skip over .STL attribute and NORM fields
000001E1 14FFA04FF7043FFD                               NEXT LPCNT0 GOTO: push_inner)             ;decrement and jump if result not zero
000001E2                       push_next_XCU:
000001E2 14FFA04FF7843FF2                           NEXT LPCNT1 GOTO: push_outer)                 ;decrement number of XCUs in LPCNT0 and jump if not zero

000001E3 12FDD20000020000                           forceBreak(uh:#0)                             ;clear all h/w breakpoints
000001E4 12FDD4000002FFFF                           sstep(uh:#{XCU15 | XCU14 | XCU13 | XCU12 | XCU11 | XCU10 | XCU9 | XCU8 | XCU7 | XCU6 | XCU5 | XCU4 | XCU3 | XCU2 | XCU1 | XCU0})
000001E5 12FDD40000020000                           sstep(uh:#0)                                  ;each XCU must be single-stepped out of a h/w break to begin running freely
```

```
                                                    ;like the CPU before it was interrupted to invoke this process, the XCU's will now encounter a "s/w" breakpoint
                                                    ;at which point the CPU will change their PC's to threadStart to begin processing

 000001E6 14FF986FDF080000      waitForXCUbreak0:   if (ud:XCU_STATUS_REG:[bit32]==0) GOTO: waitForXCUbreak0  ;wait for XCU_0 to hit s/w breakpoint
 000001E7 14FFA04FF8878000                          nop                                                       ;since push and pull ops occur immediatly, two nops must be inserted to prevent triggering
if branch taken
 000001E8 14FFA04FF8878000                          nop
 000001E9 00FDD84FFD04FFA8                          pushAll uw:PC, uw:AR4                                     ;push "begin" into all XCU PCs simultaneously
 000001EA 12FDD4000002FFFF                          sstep(uh:#{XCU15 | XCU14 | XCU13 | XCU12 | XCU11 | XCU10 | XCU9 | XCU8 | XCU7 | XCU6 | XCU5 | XCU4 | XCU3 | XCU2 | XCU1 | XCU0})
 000001EB 12FDD40000020000                          sstep(uh:#0)                                              ;each XCU must be single-stepped out of a h/w break to begin running freely

 000001EC 14FFA06FDF0C0000      waitForNotDone0:    if (ud:XCU_STATUS_REG:[bit48]==1) GOTO: waitForNotDone0   ;wait for XCU_0 to bring its DONE bit low, indicating processing has started
                                                    ;
                                                    ;  XCUs are busy processing here
                                                    ;
 000001ED 14FF986FDF0C0000      waitForDone0:       if (ud:XCU_STATUS_REG:[bit48]==0) GOTO: waitForDone0      ;wait for XCU_0 to bring its DONE bit high, indicating completion
                                                    ;
                                                    ;  now that XCU0 is done processing its triangles, it's time to start pull them out and pushing them
                                                    ;  back into external memory
                                                    ;
 000001EE 04FFC02DE0000000                  uw      AR2 = uh:add.0                              ;previously calculated current XCU base address for that XCU
 000001EF 04FFB84004000000                  uw      AR1 = uw:ext_vect_start                     ;address in external RAM of where the first triangle X1 is located
 000001F0 04FF78200B000000                          for (LPCNT1 = uh:XCUs) (                    ;for the number of XCUs ...
 000001F1                       pull_outer:
 000001F1 02000037FFA00000                  uh          0x0000 = uh:*AR2--[1]                   ;bumb by -1 XCU number
 000001F2 04FFB0400B800000                  uw          AR0 = uw:result_buf                     ;load AR0 with pointer to destination result buffer for XCU X1 of first triangle
 000001F3 14DE084DB8040000                  uw          add.1 = add:(uw:div.0, uw:#0)           ;copy calculated triangles/XCUs  into add.1 for future use

 000001F4 10FF89400C820000                              compare(uw:remainder_pull, uh:#0x0)     ;see if there was any remainder from original triangles/XCU calculation
 000001F5 14FFA04FF8800004                              IF (A==B) GOTO: no_remainder_pull       ;if no remainder, skip over a push of one more triangle for the current XCU

 000001F6 14DD00400C800001                  uw          sub.0  = sub:(uw:remainder_pull, ub:#1) ;decrement any remainder by 1
 000001F7 14DE084DB8040001                  uw          add.1 = add:(uw:div.0, uw:#1)           ;add.1 now contains the number of triangles this particular XCU is to process
 000001F8 0400C84DD0000000                  uw          remainder_pull = uw:sub.0

 000001F9                       no_remainder_pull:
 000001F9 04FF702DE0800000                              for (LPCNT0 = uh:add.1) (               ;for the number of triangles per XCU ...
 000001FA 12FF800000020011      pull_inner:                 REPEAT    uh:#17                    ;push 18 half-words into target XCU (for a total of 9 32-bit floats per triangle)
 000001FB 0B00111000230010                                      pullXCU.endi  uh:*AR1++[2], *AR2++[0]:uh:*AR0++[2]  ;reverse endian-ness just before pull (AR2 contains the current XCU number)
 000001FC 14FFA04FF8878000                                  nop
 000001FD 14FFA04FF8878000                                  nop
 000001FE 0200003007100000                  uh              0x0000  = uh:*AR1++[14]             ;bump source pointer by 14 to skip over .STL attribute and NORM fields
 000001FF 14FFA04FF7043FFB                              NEXT LPCNT0 GOTO: pull_inner)

 00000200                       pull_next_XCU:
 00000200 14FFA04FF7843FF1                          NEXT LPCNT1 GOTO: pull_outer)
 00000201 12FF8C0000020300                          setDone
 00000202 0CFFA85004700000                  sw      PC = uw:*SP++[8]                            ;return from interrupt--we are done

 00000203                       solo_process:
 00000203 34FFB04000100000                  uw      AR0 = uw:#STL_START                         ;load AR0 with address of external RAM location where raw STL file begins
 00000204 3400404000100060                  uw      ext_vect_start = uw:#STL_START + 96         ;this is the location of the first triangle X1 in external RAM
 00000205 0C00485828000000                  sw      triangles = uw:*AR0[80]                     ;set destination sign extend bit to signal reverse endian-ness and get number of triangles
 00000206 34FFB04000010000                  uw      AR0 = uw:#buf_START                         ;load AR0 with pointer to destination result buffer for XCU X1 of first triangle
 00000207 04FFB84004000000                  uw      AR1 = uw:ext_vect_start                     ;address in external RAM of where the first triangle X1 is located
 00000208 04FF702004800000                          for (LPCNT0 = uh:triangles) (               ;pull triangles in from external memory into internal working memory
 00000209 12FF800000020011      pull_solo:              REPEAT    uh:#17
 0000020A 0B00103001100000                  sh              *AR0++[2] = uh:*AR1++[2]            ;reverse endian-ness just before push (by setting destination sign extend bit (ie, "sh")
0000020B 0200003007100000                 uh          0x0000  = uh:*AR1++[14]                 ;bump source pointer by 14 to skip over .STL attribute and NORM fields
0000020C 14FFA04FF7043FFD                         NEXT LPCNT0 GOTO: pull_solo)

0000020D 1CFFA04FF887FF00                         gosub threadStart                           ;compute the transform of entire 3D object--solo
0000020E 14FFA04FF8878000                         nop
0000020F 04FFB84004000000                 uw      AR1 = uw:ext_vect_start
00000210 34FFB04000010000                 uw      AR0 = uw:#buf_START

00000211 04FF702004800000                         for (LPCNT0 = uh:triangles) (                ;push computed transform result back out to external memory
00000212 12FF800000020011      push_solo:             REPEAT    uh:#17
00000213 0B00113001000000                 sh              *AR1++[2] = uh:*AR0++[2]
00000214 14FFA04FF8878000                             nop
00000215 14FFA04FF8878000                             nop
00000216 0200003007100000                 uh          0x0000  = uh:*AR1++[14]
00000217 14FFA04FF7043FFB                         NEXT LPCNT0 GOTO: push_solo)
00000218 12FF8C0000020300                         setDone
00000219 0AFFA83004700000                 sh      PC = uh:*SP++[8]

0000021A                       progend:
00000000                                  end
```

```
00000000  ABS                0000E620  ABS.0              0000E628  ABS.1
0000E630  ABS.2              0000E638  ABS.3              00000000  ADD
0000DE00  ADD.0              0000DE08  ADD.1              0000DE50  ADD.10
0000DE58  ADD.11             0000DE60  ADD.12             0000DE68  ADD.13
0000DE70  ADD.14             0000DE78  ADD.15             0000DE10  ADD.2
0000DE18  ADD.3              0000DE20  ADD.4              0000DE28  ADD.5
0000DE30  ADD.6              0000DE38  ADD.7              0000DE40  ADD.8
0000DE48  ADD.9              00000000  ADDC               0000DD80  ADDC.0
0000DD88  ADDC.1             0000DDD0  ADDC.10            0000DDD8  ADDC.11
0000DDE0  ADDC.12            0000DDE8  ADDC.13            0000DDF0  ADDC.14
0000DDF8  ADDC.15            0000DD90  ADDC.2             0000DD98  ADDC.3
0000DDA0  ADDC.4             0000DDA8  ADDC.5             0000DDB0  ADDC.6
0000DDB8  ADDC.7             0000DDC0  ADDC.8             0000DDC8  ADDC.9
00000000  ADDITION           00000035  AFLAGRAISED        0000000C  ALTIMMDIVBYZERO
0000000F  ALTIMMINEXACT      0000000B  ALTIMMINVALID      0000000D  ALTIMMOVERFLOW
0000000E  ALTIMMUNDERFLOW    0000001F  ALWAYS             00000000  AND
0000DF80  AND.0              0000DF88  AND.1              0000DFD0  AND.10
0000DFD8  AND.11             0000DFE0  AND.12             0000DFE8  AND.13
0000DFF0  AND.14             0000DFF8  AND.15             0000DF90  AND.2
0000DF98  AND.3              0000DFA0  AND.4              0000DFA8  AND.5
0000DFB0  AND.6              0000DFB8  AND.7              0000DFC0  AND.8
0000DFC8  AND.9              0000FFB0  AR0                0000FFB8  AR1
0000FFC0  AR2                0000FFC8  AR3                0000FFD0  AR4
0000FFD8  AR5                0000FFE0  AR6                0000003E  AWAY
00000000  BCLR               0000D900  BCLR.0             0000D908  BCLR.1
0000D950  BCLR.10            0000D958  BCLR.11            0000D960  BCLR.12
0000D968  BCLR.13            0000D970  BCLR.14            0000D978  BCLR.15
0000D910  BCLR.2             0000D918  BCLR.3             0000D920  BCLR.4
0000D928  BCLR.5             0000D930  BCLR.6             0000D938  BCLR.7
0000D940  BCLR.8             0000D948  BCLR.9             00000000  BCND
0000010B  BEGIN              00000000  BIT0               00000001  BIT1
0000000A  BIT10              0000000B  BIT11              0000000C  BIT12
0000000D  BIT13              0000000E  BIT14              0000000F  BIT15
00000010  BIT16              00000011  BIT17              00000012  BIT18
00000013  BIT19              00000002  BIT2               00000014  BIT20
00000015  BIT21              00000016  BIT22              00000017  BIT23
00000018  BIT24              00000019  BIT25              0000001A  BIT26
0000001B  BIT27              0000001C  BIT28              0000001D  BIT29
00000003  BIT3               0000001E  BIT30              0000001F  BIT31
00000020  BIT32              00000021  BIT33              00000022  BIT34
00000023  BIT35              00000024  BIT36              00000025  BIT37
00000026  BIT38              00000027  BIT39              00000004  BIT4
00000028  BIT40              00000029  BIT41              0000002A  BIT42
0000002B  BIT43              0000002C  BIT44              0000002D  BIT45
0000002E  BIT46              0000002F  BIT47              00000030  BIT48
00000031  BIT49              00000005  BIT5               00000032  BIT50
00000033  BIT51              00000034  BIT52              00000035  BIT53
00000036  BIT54              00000037  BIT55              00000038  BIT56
00000039  BIT57              0000003A  BIT58              0000003B  BIT59
00000006  BIT6               0000003C  BIT60              0000003D  BIT61
0000003E  BIT62              0000003F  BIT63              00000007  BIT7
00000008  BIT8               00000009  BIT9               00000000  BITBUCKET
00000000  BSET               0000D980  BSET.0             0000D988  BSET.1
0000D9D0  BSET.10            0000D9D8  BSET.11            0000D9E0  BSET.12
0000D9E8  BSET.13            0000D9F0  BSET.14            0000D9F8  BSET.15
0000D990  BSET.2             0000D998  BSET.3             0000D9A0  BSET.4
0000D9A8  BSET.5             0000D9B0  BSET.6             0000D9B8  BSET.7
0000D9C0  BSET.8             0000D9C8  BSET.9             0000FF98  BTBC
0000FFA0  BTBS               00000000  BUBL               0000D800  BUBL.0
```

```
0000D808  BUBL.1             0000D850  BUBL.10            0000D858  BUBL.11
0000D860  BUBL.12            0000D868  BUBL.13            0000D870  BUBL.14
0000D878  BUBL.15            0000D810  BUBL.2             0000D818  BUBL.3
0000D820  BUBL.4             0000D828  BUBL.5             0000D830  BUBL.6
0000D838  BUBL.7             0000D840  BUBL.8             0000D848  BUBL.9
00010000  BUF_START          00000001  C                  00000032  CANONICAL
00000020  CAPT0_SAVE         00000028  CAPT1_SAVE         00000030  CAPT2_SAVE
00000038  CAPT3_SAVE         0000FF40  CAPTURE0           0000FF48  CAPTURE1
0000FF50  CAPTURE2           0000FF58  CAPTURE3           0000FF08  CLAS
00000000  CLASS              0000FF1E  CMPQE              0000FF1A  CMPQG
0000FF18  CMPQGE             0000FF0C  CMPQGU             0000FF16  CMPQL
0000FF14  CMPQLE             0000FF10  CMPQLU             0000FF1C  CMPQNE
0000FF12  CMPQNG             0000FF0E  CMPQNL             0000FF0A  CMPQO
0000FF0B  CMPQU              0000FF1F  CMPSE              0000FF1B  CMPSG
0000FF19  CMPSGE             0000FF0D  CMPSGU             0000FF17  CMPSL
0000FF15  CMPSLE             0000FF11  CMPSLU             0000FF1D  CMPSNE
0000FF13  CMPSNG             0000FF0F  CMPSNL             0000D780  CNVFBTA.0
0000D788  CNVFBTA.1          0000D7D0  CNVFBTA.10         0000D7D8  CNVFBTA.11
0000D7E0  CNVFBTA.12         0000D7E8  CNVFBTA.13         0000D7F0  CNVFBTA.14
0000D7F8  CNVFBTA.15         0000D790  CNVFBTA.2          0000D798  CNVFBTA.3
0000D7A0  CNVFBTA.4          0000D7A8  CNVFBTA.5          0000D7B0  CNVFBTA.6
0000D7B8  CNVFBTA.7          0000D7C0  CNVFBTA.8          0000D7C8  CNVFBTA.9
0000E500  CNVFDCS.0          0000E508  CNVFDCS.1          0000E550  CNVFDCS.10
0000E558  CNVFDCS.11         0000E560  CNVFDCS.12         0000E568  CNVFDCS.13
0000E570  CNVFDCS.14         0000E578  CNVFDCS.15         0000E510  CNVFDCS.2
0000E518  CNVFDCS.3          0000E520  CNVFDCS.4          0000E528  CNVFDCS.5
0000E530  CNVFDCS.6          0000E538  CNVFDCS.7          0000E540  CNVFDCS.8
0000E548  CNVFDCS.9          0000E400  CNVFHCS.0          0000E408  CNVFHCS.1
0000E450  CNVFHCS.10         0000E458  CNVFHCS.11         0000E460  CNVFHCS.12
0000E468  CNVFHCS.13         0000E470  CNVFHCS.14         0000E478  CNVFHCS.15
0000E410  CNVFHCS.2          0000E418  CNVFHCS.3          0000E420  CNVFHCS.4
0000E428  CNVFHCS.5          0000E430  CNVFHCS.6          0000E438  CNVFHCS.7
0000E440  CNVFHCS.8          0000E448  CNVFHCS.9          0000D700  CNVTBFA.0
0000D708  CNVTBFA.1          0000D750  CNVTBFA.10         0000D758  CNVTBFA.11
0000D760  CNVTBFA.12         0000D768  CNVTBFA.13         0000D770  CNVTBFA.14
0000D778  CNVTBFA.15         0000D710  CNVTBFA.2          0000D718  CNVTBFA.3
0000D720  CNVTBFA.4          0000D728  CNVTBFA.5          0000D730  CNVTBFA.6
0000D738  CNVTBFA.7          0000D740  CNVTBFA.8          0000D748  CNVTBFA.9
0000E480  CNVTDCS.0          0000E488  CNVTDCS.1          0000E4D0  CNVTDCS.10
0000E4D8  CNVTDCS.11         0000E4E0  CNVTDCS.12         0000E4E8  CNVTDCS.13
0000E4F0  CNVTDCS.14         0000E4F8  CNVTDCS.15         0000E490  CNVTDCS.2
0000E498  CNVTDCS.3          0000E4A0  CNVTDCS.4          0000E4A8  CNVTDCS.5
0000E4B0  CNVTDCS.6          0000E4B8  CNVTDCS.7          0000E4C0  CNVTDCS.8
0000E4C8  CNVTDCS.9          0000E380  CNVTHCS.0          0000E388  CNVTHCS.1
0000E3D0  CNVTHCS.10         0000E3D8  CNVTHCS.11         0000E3E0  CNVTHCS.12
0000E3E8  CNVTHCS.13         0000E3F0  CNVTHCS.14         0000E3F8  CNVTHCS.15
0000E390  CNVTHCS.2          0000E398  CNVTHCS.3          0000E3A0  CNVTHCS.4
0000E3A8  CNVTHCS.5          0000E3B0  CNVTHCS.6          0000E3B8  CNVTHCS.7
0000E3C0  CNVTHCS.8          0000E3C8  CNVTHCS.9          0000FF2F  COMPARE
00000036  COMPARETRUE        00000000  CONSTANTS          0000E980  CONV.0
0000E988  CONV.1             0000E9D0  CONV.10            0000E9D8  CONV.11
0000E9E0  CONV.12            0000E9E8  CONV.13            0000E9F0  CONV.14
0000E9F8  CONV.15            0000E990  CONV.2             0000E998  CONV.3
0000E9A0  CONV.4             0000E9A8  CONV.5             0000E9B0  CONV.6
0000E9B8  CONV.7             0000E9C0  CONV.8             0000E9C8  CONV.9
00000000  CONVERTFORMAT      00000000  CONVERTFROMBINARYTOASCII  00000000  CONVERTFROMDECIMALCHARACTER
00000000  CONVERTFROMHEXCHARACTER  00000000  CONVERTFROMINT     00000000  CONVERTTOBINARYFROMASCII
00000000  CONVERTTODECIMALCHARACTER  00000000  CONVERTTOHEXCHARACTER  00000000  CONVERTTOINTEGEREXACTTIESTOAWAY
00000000  CONVERTTOINTEGEREXACTTIESTOEVEN  00000000  CONVERTTOINTEGEREXACTTOWARDNEGATIVE  00000000  CONVERTTOINTEGEREXACTTOWARDPOSITIVE
```

```
00000000  CONVERTTOINTEGEREXACTTOWARDZERO  00000000  CONVERTTOINTEGERTIESTOAWAY  00000000  CONVERTTOINTEGERTIESTOEVEN
00000000  CONVERTTOINTEGERTOWARDNEGATIVE  00000000  CONVERTTOINTEGERTOWARDPOSITIVE  00000000  CONVERTTOINTEGERTOWARDZERO
00000000  COPY               0000E660  COPY.0             0000E668  COPY.1
0000E670  COPY.2             0000E678  COPY.3             00000000  COPYSIGN
0000E600  COPYSIGN.0         0000E608  COPYSIGN.1         0000E610  COPYSIGN.2
0000E618  COPYSIGN.3         00000000  COSD               0000E5C0  COSD.0
0000E5C8  COSD.1             0000E5D0  COSD.2             0000E5D8  COSD.3
0000E5C0  COS_THETAX         0000E5C8  COS_THETAY         0000E5D0  COS_THETAZ
00000000  COTD               0000E580  COTD.0             0000E588  COTD.1
0000E590  COTD.2             0000E598  COTD.3             0000FF60  CREG
00000000  DBNZ               00000000  DIV                0000DB80  DIV.0
0000DB88  DIV.1              0000DBD0  DIV.10             0000DBD8  DIV.11
0000DBE0  DIV.12             0000DBE8  DIV.13             0000DBF0  DIV.14
0000DBF8  DIV.15             0000DB90  DIV.2              0000DB98  DIV.3
0000DBA0  DIV.4              0000DBA8  DIV.5              0000DBB0  DIV.6
0000DBB8  DIV.7              0000DBC0  DIV.8              0000DBC8  DIV.9
00000007  DIVBY0FLAG         00000016  DIVBY0SIGNAL       00000002  DIVBYZERO
00000000  DIVISION           0000017F  DIVX0_             0000FEE0  DIVX0_VECT
00000108  DONE               00000004  DONE_BIT           00000000  ENDI
0000D880  ENDI.0             0000D888  ENDI.1             0000D8D0  ENDI.10
0000D8D8  ENDI.11            0000D8E0  ENDI.12            0000D8E8  ENDI.13
0000D8F0  ENDI.14            0000D8F8  ENDI.15            0000D890  ENDI.2
0000D898  ENDI.3             0000D8A0  ENDI.4             0000D8A8  ENDI.5
0000D8B0  ENDI.6             0000D8B8  ENDI.7             0000D8C0  ENDI.8
0000D8C8  ENDI.9             00000005  EXCSOURCE          00000000  EXP
0000EA00  EXP.0              0000EA08  EXP.1              0000EA50  EXP.10
0000EA58  EXP.11             0000EA60  EXP.12             0000EA68  EXP.13
0000EA70  EXP.14             0000EA78  EXP.15             0000EA10  EXP.2
0000EA18  EXP.3              0000EA20  EXP.4              0000EA28  EXP.5
0000EA30  EXP.6              0000EA38  EXP.7              0000EA40  EXP.8
0000EA48  EXP.9              00000040  EXT_VECT_START     0000EE80  FADD.0
0000EE88  FADD.1             0000EED0  FADD.10            0000EED8  FADD.11
0000EEE0  FADD.12            0000EEE8  FADD.13            0000EEF0  FADD.14
0000EEF8  FADD.15            0000EE90  FADD.2             0000EE98  FADD.3
0000EEA0  FADD.4             0000EEA8  FADD.5             0000EEB0  FADD.6
0000EEB8  FADD.7             0000EEC0  FADD.8             0000EEC8  FADD.9
00000003  FD                 0000EC00  FDIV.0             0000EC08  FDIV.1
0000EC50  FDIV.10            0000EC58  FDIV.11            0000EC60  FDIV.12
0000EC68  FDIV.13            0000EC70  FDIV.14            0000EC78  FDIV.15
0000EC10  FDIV.2             0000EC18  FDIV.3             0000EC20  FDIV.4
0000EC28  FDIV.5             0000EC30  FDIV.6             0000EC38  FDIV.7
0000EC40  FDIV.8             0000EC48  FDIV.9             00000001  FH
0000002C  FINITE             0000EB00  FMA.0              0000EB08  FMA.1
0000EB50  FMA.10             0000EB58  FMA.11             0000EB60  FMA.12
0000EB68  FMA.13             0000EB70  FMA.14             0000EB78  FMA.15
0000EB10  FMA.2              0000EB18  FMA.3              0000EB20  FMA.4
0000EB28  FMA.5              0000EB30  FMA.6              0000EB38  FMA.7
0000EB40  FMA.8              0000EB48  FMA.9              0000ED80  FMUL.0
0000ED88  FMUL.1             0000EDD0  FMUL.10            0000EDD8  FMUL.11
0000EDE0  FMUL.12            0000EDE8  FMUL.13            0000EDF0  FMUL.14
0000EDF8  FMUL.15            0000ED90  FMUL.2             0000ED98  FMUL.3
0000EDA0  FMUL.4             0000EDA8  FMUL.5             0000EDB0  FMUL.6
0000EDB8  FMUL.7             0000EDC0  FMUL.8             0000EDC8  FMUL.9
00000002  FS                 0000EE00  FSUB.0             0000EE08  FSUB.1
0000EE50  FSUB.10            0000EE58  FSUB.11            0000EE60  FSUB.12
0000EE68  FSUB.13            0000EE70  FSUB.14            0000EE78  FSUB.15
0000EE10  FSUB.2             0000EE18  FSUB.3             0000EE20  FSUB.4
0000EE28  FSUB.5             0000EE30  FSUB.6             0000EE38  FSUB.7
0000EE40  FSUB.8             0000EE48  FSUB.9             0000EC80  FTOI.0
```

```
0000EC88  FTOI.1             0000ECD0  FTOI.10            0000ECD8  FTOI.11
0000ECE0  FTOI.12            0000ECE8  FTOI.13            0000ECF0  FTOI.14
0000ECF8  FTOI.15            0000EC90  FTOI.2             0000EC98  FTOI.3
0000ECA0  FTOI.4             0000ECA8  FTOI.5             0000ECB0  FTOI.6
0000ECB8  FTOI.7             0000ECC0  FTOI.8             0000ECC8  FTOI.9
00000000  FUSEDMULTIPLYADD   00000010  INEXACT            0000019A  INEXT_
0000FEC8  INEXT_VECT         0000002F  INFINITE           00000001  INVALID
00000006  INVFLAG            00000015  INVSIGNAL          00000176  INV_
0000FEE8  INV_VECT           0000001B  IRQ                0000001A  IRQEN
000001A7  IRQ_               0000FEF0  IRQ_VECT           000001A3  IRQ_XCU
0000FF09  IS                 00000000  ISCANONICAL        00000000  ISFINITE
00000000  ISINFINITE         00000000  ISNAN              00000000  ISNORMAL
00000000  ISSIGNALING        00000000  ISSIGNMINUS        00000000  ISSUBNORMAL
00000000  ISZERO             0000ED00  ITOF.0             0000ED08  ITOF.1
0000ED50  ITOF.10            0000ED58  ITOF.11            0000ED60  ITOF.12
0000ED68  ITOF.13            0000ED70  ITOF.14            0000ED78  ITOF.15
0000ED10  ITOF.2             0000ED18  ITOF.3             0000ED20  ITOF.4
0000ED28  ITOF.5             0000ED30  ITOF.6             0000ED38  ITOF.7
0000ED40  ITOF.8             0000ED48  ITOF.9             00000100  LOAD_VECTS
00000000  LOG                0000EA80  LOG.0              0000EA88  LOG.1
0000EAD0  LOG.10             0000EAD8  LOG.11             0000EAE0  LOG.12
0000EAE8  LOG.13             0000EAF0  LOG.14             0000EAF8  LOG.15
0000EA90  LOG.2              0000EA98  LOG.3              0000EAA0  LOG.4
0000EAA8  LOG.5              0000EAB0  LOG.6              0000EAB8  LOG.7
0000EAC0  LOG.8              0000EAC8  LOG.9              00000000  LOGB
0000E800  LOGB.0             0000E808  LOGB.1             0000E850  LOGB.10
0000E858  LOGB.11            0000E860  LOGB.12            0000E868  LOGB.13
0000E870  LOGB.14            0000E878  LOGB.15            0000E810  LOGB.2
0000E818  LOGB.3             0000E820  LOGB.4             0000E828  LOGB.5
0000E830  LOGB.6             0000E838  LOGB.7             0000E840  LOGB.8
0000E848  LOGB.9             0000011E  LOOP               00000000  LOWERFLAGS
0000FF05  LOWFLG             0000FF70  LPCNT0             0000FF78  LPCNT1
00000000  MAX                0000DA80  MAX.0              0000DA88  MAX.1
0000DAD0  MAX.10             0000DAD8  MAX.11             0000DAE0  MAX.12
0000DAE8  MAX.13             0000DAF0  MAX.14             0000DAF8  MAX.15
0000DA90  MAX.2              0000DA98  MAX.3              0000DAA0  MAX.4
0000DAA8  MAX.5              0000DAB0  MAX.6              0000DAB8  MAX.7
0000DAC0  MAX.8              0000DAC8  MAX.9              00000000  MAXNUM
0000E6A0  MAXNUM.0           0000E6A8  MAXNUM.1           0000E6B0  MAXNUM.2
0000E6B8  MAXNUM.3           0000E6E0  MAXNUMMAG.0        0000E6E8  MAXNUMMAG.1
0000E6F0  MAXNUMMAG.2        0000E6F8  MAXNUMMAG.3        00000000  MIN
0000DA00  MIN.0              0000DA08  MIN.1              0000DA50  MIN.10
0000DA58  MIN.11             0000DA60  MIN.12             0000DA68  MIN.13
0000DA70  MIN.14             0000DA78  MIN.15             0000DA10  MIN.2
0000DA18  MIN.3              0000DA20  MIN.4              0000DA28  MIN.5
0000DA30  MIN.6              0000DA38  MIN.7              0000DA40  MIN.8
0000DA48  MIN.9              00000000  MINNUM             0000E680  MINNUM.0
0000E688  MINNUM.1           0000E690  MINNUM.2           0000E698  MINNUM.3
00000000  MINNUMMAG          0000E6C0  MINNUMMAG.0        0000E6C8  MINNUMMAG.1
0000E6D0  MINNUMMAG.2        0000E6D8  MINNUMMAG.3        0000FE00  MONITR_REG
00000000  MOV                00000000  MUL                0000DC00  MUL.0
0000DC08  MUL.1              0000DC50  MUL.10             0000DC58  MUL.11
0000DC60  MUL.12             0000DC68  MUL.13             0000DC70  MUL.14
0000DC78  MUL.15             0000DC10  MUL.2              0000DC18  MUL.3
0000DC20  MUL.4              0000DC28  MUL.5              0000DC30  MUL.6
0000DC38  MUL.7              0000DC40  MUL.8              0000DC48  MUL.9
00000000  MULTIPLICATION     00000002  N                  00000030  NAN
00000000  NEGATE             0000E640  NEGATE.0           0000E648  NEGATE.1
0000E650  NEGATE.2           0000E658  NEGATE.3           00000022  NEGATIVEINFINITY
```

```
00000023  NEGATIVENORMAL     00000024  NEGATIVESUBNORMAL  00000025  NEGATIVEZERO
0000001E  NEVER              00000000  NEXTDOWN           0000E700  NEXTDOWN.0
0000E708  NEXTDOWN.1         0000E710  NEXTDOWN.2         0000E718  NEXTDOWN.3
0000E720  NEXTDOWN.4         0000E728  NEXTDOWN.5         0000E730  NEXTDOWN.6
0000E738  NEXTDOWN.7         00000000  NEXTUP             0000E740  NEXTUP.0
0000E748  NEXTUP.1           0000E750  NEXTUP.2           0000E758  NEXTUP.3
0000E760  NEXTUP.4           0000E768  NEXTUP.5           0000E770  NEXTUP.6
0000E778  NEXTUP.7           00000172  NMI_               0000FEF8  NMI_VECT
0000002B  NORMAL             0000001C  NOTZANDV           000001F9  NO_REMAINDER_PULL
000001DC  NO_REMAINDER_PUSH  000001C4  NO_XCUS            0000000A  NXACTFLAG
00000019  NXACTSIGNAL        00000000  OR                 0000DF00  OR.0
0000DF08  OR.1               0000DF50  OR.10              0000DF58  OR.11
0000DF60  OR.12              0000DF68  OR.13              0000DF70  OR.14
0000DF78  OR.15              0000DF10  OR.2               0000DF18  OR.3
0000DF20  OR.4               0000DF28  OR.5               0000DF30  OR.6
0000DF38  OR.7               0000DF40  OR.8               0000DF48  OR.9
00000004  OVERFLOW           00000008  OVFLFLAG           00000017  OVFLSIGNAL
00000188  OVFL_              0000FED8  OVFL_VECT          0000FFA8  PC
0000FF98  PCC                0000FFA0  PCS                0000FF90  PC_COPY
0000FFF8  PC_REL             00000029  POSITIVEINFINITY   00000028  POSITIVENORMAL
00000027  POSITIVESUBNORMAL  00000026  POSITIVEZERO       00000000  POW
0000E300  POW.0              0000E308  POW.1              0000E350  POW.10
0000E358  POW.11             0000E360  POW.12             0000E368  POW.13
0000E370  POW.14             0000E378  POW.15             0000E310  POW.2
0000E318  POW.3              0000E320  POW.4              0000E328  POW.5
0000E330  POW.6              0000E338  POW.7              0000E340  POW.8
0000E348  POW.9              00000000  POWN               00000000  POWR
0000021A  PROGEND            00000001  PROG_LEN           80000000  PROG_START
000001FA  PULL_INNER         00000200  PULL_NEXT_XCU      000001F1  PULL_OUTER
00000209  PULL_SOLO          000001DE  PUSH_INNER         000001E2  PUSH_NEXT_XCU
000001D4  PUSH_OUTER         00000212  PUSH_SOLO          000001A9  PUSH_THREAD
000001C6  PUSH_XCUS          00000021  QUIETNAN           00000000  RADIX
0000FE10  RADIX_ADDRS        00000000  RAISEFLAGS         0000FF04  RASFLG
0000E780  REM.0              0000E788  REM.1              0000E7D0  REM.10
0000E7D8  REM.11             0000E7E0  REM.12             0000E7E8  REM.13
0000E7F0  REM.14             0000E7F8  REM.15             0000E790  REM.2
0000E798  REM.3              0000E7A0  REM.4              0000E7A8  REM.5
0000E7B0  REM.6              0000E7B8  REM.7              0000E7C0  REM.8
0000E7C8  REM.9              00000000  REMAINDER          000000C8  REMAINDER_PULL
000000C0  REMAINDER_PUSH     00000000  RESTOREFLAGS       000000B8  RESULT_BUF
0000003C  RM0                0000003D  RM1                0000003F  RM_ATTRIB
0000FE18  RNDDIR_REG         00000011  RNF_DIVBY0         00000010  RNF_INV
00000014  RNF_NXACT          00000012  RNF_OVFL           00000013  RNF_UNFL
00000002  ROTX               00000003  ROTY               00000004  ROTZ
00000000  ROUNDTOINTEGRALEXACT  00000000  ROUNDTOINTEGRALTIESTOAWAY  00000000  ROUNDTOINTEGRALTIESTOEVEN
00000000  ROUNDTOINTEGRALTOWARDNEGATIVE  00000000  ROUNDTOINTEGRALTOWARDPOSITIVE  00000000  ROUNDTOINTEGRALTOWARDZERO
0000FF80  RPT                0000FF01  RSTFLG             0000E900  RTOI.0
0000E908  RTOI.1             0000E950  RTOI.10            0000E958  RTOI.11
0000E960  RTOI.12            0000E968  RTOI.13            0000E970  RTOI.14
0000E978  RTOI.15            0000E910  RTOI.2             0000E918  RTOI.3
0000E920  RTOI.4             0000E928  RTOI.5             0000E930  RTOI.6
0000E938  RTOI.7             0000E940  RTOI.8             0000E948  RTOI.9
00000000  SAVEALLFLAGS       0000FF00  SAVEDFLAGS         0000FE08  SAVEDMODES
00000000  SAVEMODES          00000004  SB                 00000000  SCALEB
0000E880  SCALEB.0           0000E888  SCALEB.1           0000E8D0  SCALEB.10
0000E8D8  SCALEB.11          0000E8E0  SCALEB.12          0000E8E8  SCALEB.13
0000E8F0  SCALEB.14          0000E8F8  SCALEB.15          0000E890  SCALEB.2
0000E898  SCALEB.3           0000E8A0  SCALEB.4           0000E8A8  SCALEB.5
0000E8B0  SCALEB.6           0000E8B8  SCALEB.7           0000E8C0  SCALEB.8
```

```
0000E8C8  SCALEB.9            00000050  SCALEX              00000058  SCALEY
00000060  SCALEZ              00000005  SCAL_X              00000006  SCAL_Y
00000007  SCAL_Z              0000FF30  SCHEDCMP            0000FF38  SCHEDULER
00000007  SD                  00000005  SH                  00000000  SHFT
00000000  SHIFT               0000DB00  SHIFT.0             0000DB08  SHIFT.1
0000DB50  SHIFT.10            0000DB58  SHIFT.11            0000DB60  SHIFT.12
0000DB68  SHIFT.13            0000DB70  SHIFT.14            0000DB78  SHIFT.15
0000DB10  SHIFT.2             0000DB18  SHIFT.3             0000DB20  SHIFT.4
0000DB28  SHIFT.5             0000DB30  SHIFT.6             0000DB38  SHIFT.7
0000DB40  SHIFT.8             0000DB48  SHIFT.9             00000031  SIGNALING
00000020  SIGNALINGNAN        0000002A  SIGNMINUS           00000000  SIND
0000E5E0  SIND.0              0000E5E8  SIND.1              0000E5F0  SIND.2
0000E5F8  SIND.3              0000E5E0  SIN_THETAX          0000E5E8  SIN_THETAY
0000E5F0  SIN_THETAZ          00000203  SOLO_PROCESS        0000FFE8  SP
0000FFF0  SP_TOS              0000EB80  SQRT.0              0000EB88  SQRT.1
0000EBD0  SQRT.10             0000EBD8  SQRT.11             0000EBE0  SQRT.12
0000EBE8  SQRT.13             0000EBF0  SQRT.14             0000EBF8  SQRT.15
0000EB90  SQRT.2              0000EB98  SQRT.3              0000EBA0  SQRT.4
0000EBA8  SQRT.5              0000EBB0  SQRT.6              0000EBB8  SQRT.7
0000EBC0  SQRT.8              0000EBC8  SQRT.9              00000000  SQUAREROOT
0000FF88  STATUS              00100000  STL_START           00000000  SUB
0000DD00  SUB.0               0000DD08  SUB.1               0000DD50  SUB.10
0000DD58  SUB.11              0000DD60  SUB.12              0000DD68  SUB.13
0000DD70  SUB.14              0000DD78  SUB.15              0000DD10  SUB.2
0000DD18  SUB.3               0000DD20  SUB.4               0000DD28  SUB.5
0000DD30  SUB.6               0000DD38  SUB.7               0000DD40  SUB.8
0000DD48  SUB.9               00000000  SUBB                0000DC80  SUBB.0
0000DC88  SUBB.1              0000DCD0  SUBB.10             0000DCD8  SUBB.11
0000DCE0  SUBB.12             0000DCE8  SUBB.13             0000DCF0  SUBB.14
0000DCF8  SUBB.15             0000DC90  SUBB.2              0000DC98  SUBB.3
0000DCA0  SUBB.4              0000DCA8  SUBB.5              0000DCB0  SUBB.6
0000DCB8  SUBB.7              0000DCC0  SUBB.8              0000DCC8  SUBB.9
0000002E  SUBNORMAL           00000038  SUBS_DIVBY0         00000037  SUBS_INV
0000003B  SUBS_NXACT          00000039  SUBS_OVFL           0000003A  SUBS_UNFL
00000000  SUBTRACTION         00000006  SW                  00000000  TAND
0000E5A0  TAND.0              0000E5A8  TAND.1              0000E5B0  TAND.2
0000E5B8  TAND.3              00000000  TESTFLAGS           00000000  TESTSAVEDFLAGS
0000010D  THREADSTART         000001A7  THREAD_END          0000FF68  TIMER
0000FF07  TORD                0000FF06  TORDM               00000033  TOTLORDER
00000034  TOTLORDERMAG        00000068  TRANSX              00000070  TRANSY
00000078  TRANSZ              00000008  TRANS_X             00000009  TRANS_Y
0000000A  TRANS_Z             00000048  TRIANGLES           0000FF03  TSTFLG
0000FF02  TSTSFLG             00000000  UB                  00000003  UD
00000001  UH                  00000008  UNDERFLOW           00000009  UNFLFLAG
00000018  UNFLSIGNAL          00000191  UNFL_               0000FED0  UNFL_VECT
00000002  UW                  00000003  V                   000001ED  WAITFORDONE0
000001EC  WAITFORNOTDONE0     000001E6  WAITFORXCUBREAK0    00000008  WORK_1
00000010  WORK_2              00000018  WORK_3              00000080  X1
0000008C  X2                  00000098  X3                  00000000  XCU.0
00000001  XCU.1               0000000A  XCU.10              0000000B  XCU.11
0000000C  XCU.12              0000000D  XCU.13              0000000E  XCU.14
0000000F  XCU.15              00000002  XCU.2               00000003  XCU.3
00000004  XCU.4               00000005  XCU.5               00000006  XCU.6
00000007  XCU.7               00000008  XCU.8               00000009  XCU.9
00000001  XCU0                00000002  XCU1                00000400  XCU10
00000800  XCU11               00001000  XCU12               00002000  XCU13
00004000  XCU14               00008000  XCU15               00000004  XCU2
00000008  XCU3                00000010  XCU4                00000020  XCU5
00000040  XCU6                00000080  XCU7                00000100  XCU8
```

```
00000200  XCU9               000000B0  XCUS               0000FDF8  XCU_CNTRL_REG
0000FDE0  XCU_MON_REQUEST    0000FDD8  XCU_PUSH_ALL       0000FDF0  XCU_STATUS_REG
00000007  XFD                00000005  XFH                00000002  XFORM_3AXIS_PARAMETERS
00000006  XFS                00000000  XOR                0000DE80  XOR.0
0000DE88  XOR.1              0000DED0  XOR.10             0000DED8  XOR.11
0000DEE0  XOR.12             0000DEE8  XOR.13             0000DEF0  XOR.14
0000DEF8  XOR.15             0000DE90  XOR.2              0000DE98  XOR.3
0000DEA0  XOR.4              0000DEA8  XOR.5              0000DEB0  XOR.6
0000DEB8  XOR.7              0000DEC0  XOR.8              0000DEC8  XOR.9
00000084  Y1                 00000090  Y2                 0000009C  Y3
00000000  Z                  00000088  Z1                 00000094  Z2
000000A0  Z3                 0000002D  ZERO               0000001D  ZORV
000001B5  _16_XCUS           000001C1  _1_XCU             000001BE  _2_XCUS
000001BB  _4_XCUS            000001B8  _8_XCUS
```