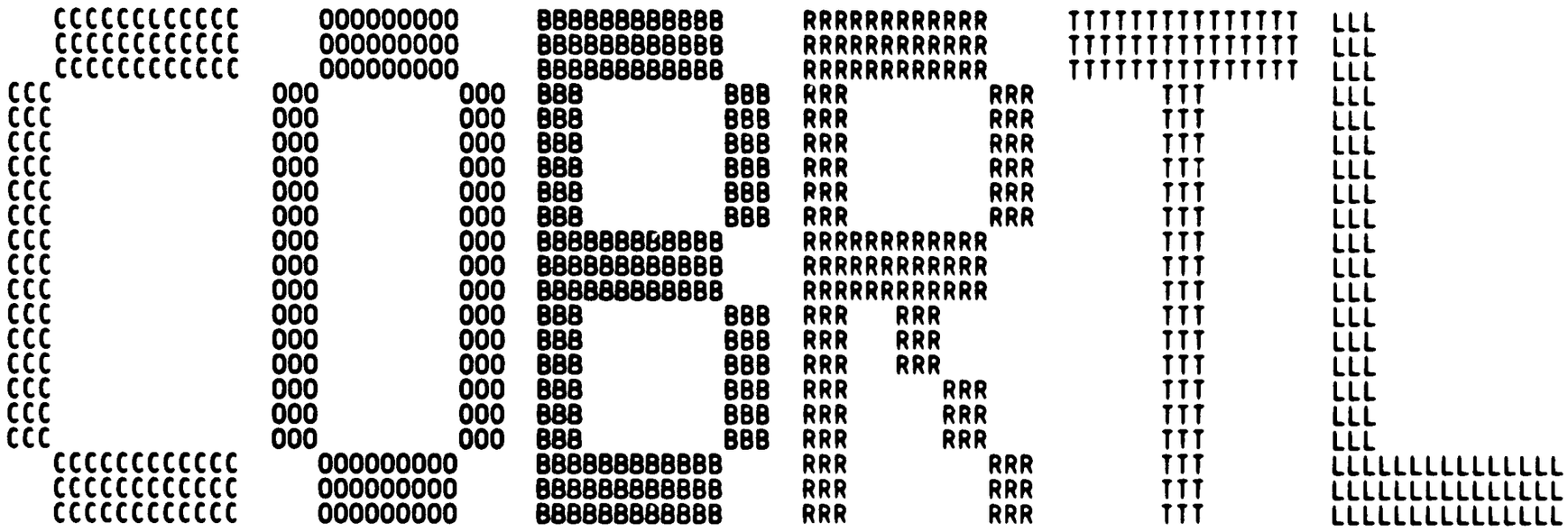
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRRRRRRRRRR TTTTTTTTTTTTTTT LLL
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRRRRRRRRRR TTTTTTTTTTTTTTT LLL
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRRRRRRRRRR TTTTTTTTTTTTTTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBBBBBBBBBB RRRRRRRRRRR TTT LLL
CCC 000 000 BBBBBBBBBBBB RRRRRRRRRRR TTT LLL
CCC 000 000 BBBBBBBBBBBB RRRRRRRRRRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCC 000 000 BBB BBB RRR RRR TTT LLL
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRR RRR TTT LLLLLLLLLLLLLLL
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRR RRR TTT LLLLLLLLLLLLLLL
CCCCCCCCCCC 000000000 BBBBBBBBBBBB RRR RRR TTT LLLLLLLLLLLLLLL

```
  CCCCCCCC    000000    BBBBBBBB    DDDDDDDD      IIIIII    VV      VV    QQQQQQ
  CCCCCCCC    000000    BBBBBBBB    DDDDDDDD      IIIIII    VV      VV    QQQQQQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BBBBBBBB    DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BBBBBBBB    DD      DD      II      VV      VV  QQ      QQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ  QQ  QQ
CC           OO      OO  BB      BB  DD      DD      II      VV      VV  QQ  QQ  QQ
CC           OO      OO  BB      BB  DD      DD      II        VV  VV    QQ    QQ      ....
CC           OO      OO  BB      BB  DD      DD      II        VV  VV    QQ    QQ      ....
  CCCCCCCC    000000    BBBBBBBB    DDDDDDDD      IIIIII        VV        QQQQ  QQ      ....
  CCCCCCCC    000000    BBBBBBBB    DDDDDDDD      IIIIII        VV        QQQQ  QQ      ....

LL             IIIIII      SSSSSSSS
LL             IIIIII      SSSSSSSS
LL               II      SS
LL               II      SS
LL               II      SS
LL               II      SS
LL               II        SSSSSS
LL               II        SSSSSS
LL               II              SS
LL               II              SS
LL               II              SS
LL               II              SS
LLLLLLLLLL     IIIIII    SSSSSSSS
LLLLLLLLLL     IIIIII    SSSSSSSS
```

Table of contents

```
0000     1          .TITLE  COB$DIVQ_R8     COBOL Divide Quadwords
0000     2          .IDENT  /1-012/         ; File: COBDIVQ.MAR EDIT:PDG1012
0000     3
0000     4 ;
0000     5 ;
0000     6 ;****************************************************************************
0000     7 ;*                                                                          *
0000     8 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                *
0000     9 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000    10 ;*  ALL RIGHTS RESERVED.                                                    *
0000    11 ;*                                                                          *
0000    12 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000    13 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
0000    14 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000    15 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000    16 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000    17 ;*  TRANSFERRED.                                                            *
0000    18 ;*                                                                          *
0000    19 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000    20 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
0000    21 ;*  CORPORATION.                                                            *
0000    22 ;*                                                                          *
0000    23 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000    24 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    25 ;*                                                                          *
0000    26 ;*                                                                          *
0000    27 ;****************************************************************************
0000    28 ;
0000    29 ;
0000    30 ;
0000    31 ; FACILITY: COBOL ARITHMETIC
0000    32 ;++
0000    33 ; ABSTRACT:
0000    34 ;       This module contains the routine which divides two
0000    35 ;       quadwords, producing a quadword result.
0000    36 ;
0000    37 ;--
0000    38 ;
0000    39 ; VERSION: 1
0000    40 ;
0000    41 ; HISTORY:
0000    42 ;
0000    43 ; AUTHOR:
0000    44 ;       John Sauter, 26-DEC-78
0000    45 ;
0000    46 ; MODIFIED BY:
0000    47 ;
0000    48 ;
0000    49 ;
```

```
0000     51          .SBTTL  HISTORY         ; Detailed Current Edit History
0000     52
0000     53
0000     54 ; Edit History for Version 1 of COBDIVQ
0000     55 ;
0000     56 ; 1-001 - Original from COBOL-74, MODULE FASB4 ROUTINE FAST_DIV_4
0000     57 ; 1-002 - Make the entry point symbol global.  JBS 03-JAN-1979
0000     58 ; 1-003 - Minor editing cleanups.  JBS 11-JAN-1979
0000     59 ; 1-004 - Complete rewrite.  MLJ 11-Mar-1979
0000     60 ; 1-005 - 19 digit temps and other minor changes.  MLJ 13-Mar-1979
0000     61 ; 1-006 - Correct DIVP round towarrds zero problem.  PDG 12-Jul-1979
0000     62 ; 1-007 - Make it work correctly with overlapping input and output
0000     63 ;         operands.   RKR 24-SEPT-79
0000     64 ; 1-008 - Cosmetic changes.  RKR 21-OCT-79
0000     65 ; 1-009 - Complete rewrite.  PDG 15-Jun-1981
0000     66 ;         This includes several of the ideas from FAST_DIV_4.
0000     67 ; 1-010 - Added EDIT field for use in checkin's audit trail.  LB 28-JUL-81
0000     68 ; 1-011 - Updated copyright date.  LB 30-JUL-81
0000     69 ; 1-012 - Some bug fix.  PDG 9-Aug-1981
```

```
                        .SBTTL  DECLARATIONS

;
; INCLUDE FILES:
;

;
; EXTERNAL SYMBOLS:
;       NONE
;

;
; MACROS:
;       NONE
;

;
; PSECT DECLARATIONS:
;       .PSECT  _COB$CODE       PIC, SHR, LONG, EXE, NOWRT

;
; EQUATED SYMBOLS:
;       NONE
;

;
; OWN STORAGE:
;
;       NONE
;
```

```
                        .SBTTL  COB$DIVQ_R8

                ;++
                ; FUNCTIONAL DESCRIPTION:
                ;
                ;       Divides two quadwords, producing a quadword result.
                ;
                ; CALLING SEQUENCE:
                ;
                ;       JSB COB$DIVQ_R8 (divisor.rq.r, dividend.rq.r, quotient.wq.r)
                ;
                ;       Arguments are passed in R6, R7 and R8.
                ;
                ; INPUT PARAMETERS:
                ;
                ;       DIVISOR.rq.r            The divisonr.
                ;       DIVIDEND.rq.r           The dividend.
                ;
                ; IMPLICIT INPUTS:
                ;
                ;       All of the trap bits in the PSL are assumed off.
                ;
                ; OUTPUT PARAMETERS:
                ;
                ;       QUOTIENT.wq.r           The result of the division; DIVIDEND/DIVISOR.
                ;
                ; IMPLICIT OUTPUTS:
                ;
                ;       NONE
                ;
                ; COMPLETION CODES:
                ;
                ;       NONE
                ;
                ; SIDE EFFECTS:
                ;
                ;       Destroys R0 through R8.
                ;
                ;       If the divisor equals zero, a "divide by zero" exception is raised.
                ;       Integer overflow occurs if only if the largest negative integer is
                ;       divided by -1.  The result is the largest negative integer.
                ;
                ; NOTES:
                ;
                ;       In comments below, the following conventions are used:
                ;
                ;       Equations describe the current state of the registers.
                ;       A prefix of "S" indicates the longword is to be considered signed.
                ;       A prefix of "U" indicates the longword is to be considered unsigned.
                ;       A primed value represents the an value of a register.
                ;--

                COB$DIVQ_R8::
04 A6 66 01 1F EC       CMPV    #31, #1, (R6), 4(R6)    ; Divisor in longword range?
                60 12   BNEQ    200$                    ; No, do slow code
                ;+
                ;
```

```
                                0008   159 ; The divisor is a longword, so (hopefully), a single EDIV will work
                                0008   160 ;
                                0008   161 ;-
     50   51   67   66   7B    0008   162         EDIV    (R6), (R7), R1, R0      ; Try the EDIV
                    09   1D    000D   163         BVS     100$                    ; Branch if failed
                88  51   D0    000F   164         MOVL    R1, (R8)+               ; Store quotient
     88   51   01   1F   EE    0012   165         EXTV    #31, #1, R1, (R8)+      ; Store high-order longword
                         05    0017   166         RSB                             ; Return
                               0018   167 100$:
                               0018   168 ;+
                               0018   169 ;
                               0018   170 ; The divisor is a longword, but a single EDIV doesn't work
                               0018   171 ;
                               0018   172 ;-
                52   67   7D    0018   173         MOVQ    (R7), R2                ; Grab dividend
                56   66   D0    001B   174         MOVL    (R6), R6                ; Grab divisor
                               001E   175         ;
                               001E   176         ; We want to compute:
                               001E   177         ;
                               001E   178         ; SR3x2^32 + UR2 = SR6 x (SR1x2^32 + UR0) + remainder
                               001E   179         ;
                               001E   180         ; where the remainder is of the same sign as the numerator,
                               001E   181         ; and is less (in absolute value) than the divisor.
                               001E   182         ;
     54   53   01   1F   EE    001E   183         EXTV    #31, #1, R3, R4         ; Sign extend dividend to R4-R3-R2
     53   51   53   56   7B    0023   184         EDIV    R6, R3, R1, R3          ; First EDIV (can't overflow)
                               0028   185         ;
                               0028   186         ; SR4x2^32 + UR3' = SR3' = SR6 x SR1 + SR3
                               0028   187         ;
                               0028   188         ; Now try the second EDIV
                               0028   189         ;
     55   50   52   56   7B    0028   190         EDIV    R6, R2, R0, R5
                    17   1C    002D   191         BVC     107$
                               002F   192         ;
                               002F   193         ; We know that the quotient will fit into 32 bits, but because, (in
                               002F   194         ; the true quad result of the divide), bit 32 doesn't equal bit 31,
                               002F   195         ; EDIV sets overflow and trashes the results.  We borrow (carry?) from
                               002F   196         ; the high longword of the quotient so that bits 32 and 31 of the true
                               002F   197         ; quadword quotient will be equal.  The signs of the remainder and the
                               002F   198         ; dividend may differ, but that's okay (for now).
                               002F   199         ;
          55   53   56   CD    002F   200         XORL3   R6, R3, R5              ; Determine sign of true result
                    07   19    0033   201         BLSS    103$                    ; Branch if true result is negative
                53   56   C2    0035   202         SUBL2   R6, R3                  ; Subtract SR6x2^32 from dividend
                    51   D6    0038   203         INCL    R1                      ; Increase quotient by 2^32
                    05   11    003A   204         BRB     104$
                53   56   C0    003C   205 103$:   ADDL2   R6, R3                  ; Add SR6x2^32 to dividend
                    51   D7    003F   206         DECL    R1                      ; Decrease quotient by 2^32
     55   50   52   56   7B    0041   207 104$:   EDIV    R6, R2, R0, R5          ; Re-try the second EDIV
                               0046   208 107$:   ;
                               0046   209         ; SR3x2^32 + UR2 = SR6 x SR0 + SR5
                               0046   210         ;
                               0046   211         ; SR3'x2^32 + UR2  = SR6 x SR1 x 2^32 + SR3x2^32 + UR2
                               0046   212         ;       = SR6 x SR1 x 2^32 + SR6 x SR0 + SR5
                               0046   213         ;       = SR6 x (SR1x2^32 + SR0) + SR5          (getting close)
                               0046   214         ;
                    02   18    0046   215         BGEQ    108$                    ; Make R0 unsigned
```

```
                51   D7  0048   216          DECL    R1
                         004A   217 108$:    ;
                         004A   218          ; SR3'x2^32 + UR2 = SR6 x (SR1x2^32 + UR0) + SR5
                         004A   219          ;
                         004A   220          ; Fix up the result if the remainder has the wrong sign
                         004A   221          ;
          53    54   55  CD  004A   222          XORL3   R5, R4, R3              ; Signs same?
                     14  18  004E   223          BGEQ    102$                    ; If so, OK.
                     55  D5  0050   224          TSTL    R5                      ; See if remainder is zero
                     10  13  0052   225          BEQL    102$                    ; If so, the quotient is correct
                     54  D5  0054   226          TSTL    R4                      ; Was original dividend plus or minus?
                     07  18  0056   227          BGEQ    105$
                     50  D6  0058   228          INCL    R0                      ; Increment the result
                51   00  D8  005A   229          ADWC    #0, R1
                     05  11  005D   230          BRB     102$
                     50  D7  005F   231 105$:    DECL    R0                      ; Decrement the result
                51   00  D9  0061   232          SBWC    #0, R1
                68   50  7D  0064   233 102$:    MOVQ    R0, (R8)                ; Store result
                     05  0067   234              RSB                             ; And return (remainder = psuedo R5)
                         0068   235 200$:
                         0068   236 ;+
                         0068   237 ;
                         0068   238 ; The divisor doesn't fit into a signed longword
                         0068   239 ;
                         0068   240 ;-
                     7E  D4  0068   241          CLRL    -(SP)                   ; Clear negate flag
                52   67  7D  006A   242          MOVQ    (R7), R2                ; Grab dividend into R3-R2
                     0B  18  006D   243          BGEQ    201$                    ; Skip if positive
                     6E  96  006F   244          INCB    (SP)                    ; Toggle negate flag
                52   52  CE  0071   245          MNEGL   R2, R2                  ; Negate dividend
                53   00  D8  0074   246          ADWC    #0, R3
                53   53  CE  0077   247          MNEGL   R3, R3
                56   66  7D  007A   248 201$:    MOVQ    (R6), R6                ; Grab divisor into R7-R6
                     0B  18  007D   249          BGEQ    202$                    ; Skip if positive
                     6E  96  007F   250          INCB    (SP)                    ; Toggle negate flag
                56   56  CE  0081   251          MNEGL   R6, R6                  ; Negate divisor
                57   00  D8  0084   252          ADWC    #0, R7
                57   57  CE  0087   253          MNEGL   R7, R7
                         008A   254 202$:    ;
                         008A   255          ; Normalize the divisor
                         008A   256          ;
          50    57   01  C9  008A   257          BISL3   #1, R7, R0              ; Make sure high longword not zero
                50   50  4E  008E   258          CVTLF   R0, R0                  ; To find most significant bit
                         0091   259 ;;;;;;;  BGTR    207$                    ; Branch if everything okay
                     5C  19  0091   260          BLSS    203$                    ; Branch if divisor still negative!
                     54  D4  0093   261 207$:    CLRL    R4      ; needed?       ; Clear R4 in case we don't shift
50        50   08  07  EF  0095   262          EXTZV   #7, #8, R0, R0          ; Move the exponent into low byte
  50        9F  8F  50  83  009A   263          SUBB3   R0, #159, R0            ; Un-normalize the exponent
                     11  19  009F   264          BLSS    213$    ; needed?       ; Don't shift right! (loses accuracy)
          56   56  50  79  00A1   265          ASHQ    R0, R6, R6              ; Shift the divisor into R7-R6
                         00A5   266          ;
                         00A5   267          ; Shift dividend by same amount, into R4-R3-R2
                         00A5   268          ;
          51   20  50  C3  00A5   269          SUBL3   R0, #32, R1             ; Get offset in bits for R4
54        53   50  51  EF  00A9   270          EXTZV   R1, R0, R3, R4          ; Grab portion for R4 (must be extZv!)
          52   52  50  79  00AE   271          ASHQ    R0, R2, R2              ; Shift lower two
                         00B2   272 213$:    ;
```

```
                                00B2    273          ; Divide R4-R3-R2 by R7-R6 giving R5-R4 remainder R3-R2
                                00B2    274          ;
                                00B2    275          ; First, divide R4-R3 by R7 giving R4 remainder R3
                                00B2    276          ;
                          55    D4    00B2    277          CLRL    R5                      ; Clear high longword of quotient
          50    51    53    57    7B    00B4    278          EDIV    R7, R3, R1, R0          ; R4-R3 / R7 = R5-R4 rem R3
                                00B9    279 ;;;;;;;  BVC     222$                    ; Branch if no overflow
                          3F    1D    00B9    280          BVS     291$                    ; Branch if we had overflow
                    53    50    7D    00BB    281 222$:    MOVQ    R0, R3                  ; Quotient in R5-R4, remainder in R3
                                00BE    282          ;
                                00BE    283          ;       SR7 x SR4 + SR3 = SR4'x2^32 + UR3'
                                00BE    284          ;
                                00BE    285          ; Multiply back and subtract to compute remainder by using the formula:
                                00BE    286          ;
                                00BE    287          ;       SR4'x2^64 + UR3'x2^32 + UR2 - (SR7x2^32 + UR6) x SR4
                                00BE    288          ;    =  SR3x2^32 + UR2 - UR6 x SR4
                                00BE    289          ;
          50    00    54    56    7A    00BE    290 295$:    EMUL    R6, R4, #0, R0          ; R1-R0 = UR6 x SR4
                03    56    1F    E1    00C3    291          BBC     #31, R6, 209$           ; Consider R6 unsigned
                      51    54    C0    00C7    292          ADDL2   R4, R1
                      52    50    C2    00CA    293 209$:    SUBL2   R0, R2                  ; Subtract the product
                      53    51    D9    00CD    294          SBWC    R1, R3
                            0D    18    00D0    295          BGEQ    205$                    ; Was estimate too big?
                            54    D7    00D2    296 204$:    DECL    R4                      ; Yes, decrement estimate
                      55    00    D9    00D4    297          SBWC    #0, R5
                      52    56    C0    00D7    298          ADDL2   R6, R2                  ; And add back divisor
                      53    57    D8    00DA    299          ADWC    R7, R3
                            F3    19    00DD    300          BLSS    204$                    ; And continue until positive
                                00DF    301 205$:    ;
                                00DF    302          ; See if we must negate the quotient
                                00DF    303          ;
                      09    8E    E9    00DF    304          BLBC    (SP)+, 206$             ; Skip if no negate required
                54    54    CE    00E2    305          MNEGL   R4, R4                  ; Negate result
                55    00    D8    00E5    306          ADWC    #0, R5
                55    55    CE    00E8    307          MNEGL   R5, R5
                68    54    7D    00EB    308 206$:    MOVQ    R4, (R8)                ; Store quotient
                            05    00EE    309          RSB                             ; Return
                                00EF    310 ;+
                                00EF    311 ;
                                00EF    312 ; This code was moved out of line so that 'usual' cases don't cause branches.
                                00EF    313 ;
                                00EF    314 ;-
                                00EF    315 203$:    ;
                                00EF    316          ; The divisor was still negative!  This is because the original divisor
                                00EF    317          ; was equal to -2^63.  The only possible quotients are zero and one,
                                00EF    318          ; depending on whether the dividend also equals 2^32.
                                00EF    319          ;
    55    53    01    1F    EF    00EF    320          EXTZV   #31, #1, R3, R5         ; Get low longword of result
                68    55    7D    00F4    321          MOVQ    R5, (R8)                ; Store quotient (R6 = 0, remember?)
                      8E    D5    00F7    322          TSTL    (SP)+                   ; Throw away the negate flag
                            05    00F9    323          RSB                             ; And return
                                00FA    324 ;+
                                00FA    325 ;
                                00FA    326 ; This code was moved out of line so that 'usual' cases don't cause branches.
                                00FA    327 ;
                                00FA    328 ;-
                                00FA    329 291$:    ;
```

```
                                00FA   330          ; Since we had overflow, we know that, before normalizing:
                                00FA   331          ;
                                00FA   332          ; 2^63   >= dividend >= 2^62
                                00FA   333          ; 2^32+7 >= divisor  >= 2^31
                                00FA   334          ; 2^32   >= quotient >= 2^31    (true quotient, that is)
                                00FA   335          ;
                                00FA   336          ; Also, on overflow, EDIV sets the remainder to zero.
                                00FA   337          ;
                                00FA   338          ; We can compute the desired results via:
                                00FA   339          ;
                                00FA   340          ; ( (R4-R3) - R7x2^31 ) / R7 + 2^31 = R1 rem R0
                                00FA   341          ;
                                00FA   342          ; Unless this also causes an overflow, in which case, we know that
                                00FA   343          ; quotient must equal 2^32.
                                00FA   344          ;
 50   00   57   80000000 8F   7A 00FA   345          EMUL    #1@31, R7, #0, R0       ; Multiply by -2^31
                     50   53   C0 0103   346          ADDL2   R3, R0                  ; Compute the difference
                     51   00   D8 0106   347          ADWC    #0, R1
                     51   54   C0 0109   348          ADDL2   R4, R1
           50   51   50   57   7B 010C   349          EDIV    R7, R0, R1, R0          ; Try the EDIV one more time
                          05   1D 0111   350          BVS     292$                    ; Oh, damn!
                  A4 51   1F   E3 0113   351          BBCS    #31, R1, 222$           ; Add 2^31 to the quotient
                          00   0117   352          HALT                            ; We should NEVER get here
                50   54   57   C3 0118   353 292$:   SUBL3   R7, R4, R0             ; Compute the remainder
                          51   D4 011C   354          CLRL    R1                      ; Set low longword of quotient
                          55   D6 011E   355          INCL    R5                      ; Set high longword of quotient
                          99   11 0120   356          BRB     222$                    ; Join common code
                               0122   357
                               0122   358          .END
```

COB$DIVQ_R8 00000000 RG 01

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
_COB$CODE	00000122 (290.)	01 (1.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	41	00:00:00.05	00:00:00.62
Command processing	132	00:00:00.39	00:00:07.02
Pass 1	75	00:00:00.55	00:00:03.74
Symbol table sort	0	00:00:00.00	00:00:00.00
Pass 2	72	00:00:00.46	00:00:02.04
Symbol table output	1	00:00:00.01	00:00:00.01
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	325	00:00:01.47	00:00:13.60

The working set limit was 1050 pages.
5550 bytes (11 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 1 non-local and 21 local symbols.
358 source lines were read in Pass 1, producing 8 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	0

0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:COBDIVQ/OBJ=OBJ$:COBDIVQ MSRC$:COBDIVQ/UPDATE=(ENH$:COBDIVQ)

COBDIVQ
LIS

COBFINDNA
LIS

COBDBEXCE
LIS

COBEXPI
LIS

COBDEEDIT
LIS

COBDISPLA
LIS

COBESCGEN
LIS

COBERROR
LIS

COBDHANDL
LIS