```
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

```
_$2

Sym
---
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI

PLI
PLI
PLI

PLI
PLI
PLI
PLI
PLI
PLI
PLI

PLI
PLI
PLI
PLI

PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
```

F 2

```
**FILE**ID**PLICONVRT
```

```
PPPPPPPP    LL            IIIIII      CCCCCCCC    OOOOOO    NN      NN  VV      VV  RRRRRRRR    TTTTTTTTTT
PPPPPPPP    LL            IIIIII      CCCCCCCC    OOOOOO    NN      NN  VV      VV  RRRRRRRR    TTTTTTTTTT
PP      PP  LL              II      CC          OO      OO  NN      NN  VV      VV  RR      RR      TT
PP      PP  LL              II      CC          OO      OO  NN      NN  VV      VV  RR      RR      TT
PP      PP  LL              II      CC          OO      OO  NNNN    NN  VV      VV  RR      RR      TT
PP      PP  LL              II      CC          OO      OO  NNNN    NN  VV      VV  RR      RR      TT
PPPPPPPP    LL              II      CC          OO      OO  NN  NN  NN  VV      VV  RRRRRRRR        TT
PPPPPPPP    LL              II      CC          OO      OO  NN  NN  NN  VV      VV  RRRRRRRR        TT
PP          LL              II      CC          OO      OO  NN    NNNN  VV      VV  RR  RR          TT
PP          LL              II      CC          OO      OO  NN    NNNN  VV      VV  RR  RR          TT
PP          LL              II      CC          OO      OO  NN      NN    VV  VV    RR    RR        TT          ....
PP          LL              II      CC          OO      OO  NN      NN    VV  VV    RR    RR        TT          ....
PP          LLLLLLLLLL    IIIIII      CCCCCCCC    OOOOOO    NN      NN      VV      RR      RR      TT          ....
PP          LLLLLLLLLL    IIIIII      CCCCCCCC    OOOOOO    NN      NN      VV      RR      RR      TT          ....

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
```

PLI
1-0

Table of contents

```
0000          .title pli$convert - pl1 general purpose data type conversion package
0000          .ident  /1-007/                                 ;Edit DSB1007
0000                                                          ;Edit DSB1006
0000                                                          ;Edit WHM1005
0000                                                          ;Edit WHM1004
0000                                                          ;Edit WHM1003
0000     ;
0000     ;****************************************************************************
0000     ;*                                                                          *
0000     ;*   COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                *
0000     ;*   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
0000     ;*   ALL RIGHTS RESERVED.                                                   *
0000     ;*                                                                          *
0000     ;*   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
0000     ;*   ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
0000     ;*   INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
0000     ;*   COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
0000     ;*   OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
0000     ;*   TRANSFERRED.                                                           *
0000     ;*                                                                          *
0000     ;*   THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
0000     ;*   AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
0000     ;*   CORPORATION.                                                           *
0000     ;*                                                                          *
0000     ;*   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
0000     ;*   SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
0000     ;*                                                                          *
0000     ;*                                                                          *
0000     ;****************************************************************************
0000     ;
0000     ;
0000     ; ++
0000     ; facility:
0000     ;
0000     ;      VAX-11 PL1 runtime library.
0000     ;
0000     ; abstract:
0000     ;
0000     ; This routine converts any pl1 computational data type to any other.
0000     ;
0000     ; author: R. Heinen 2-jan-1979
0000     ;
0000     ; Modifications:
0000     ;
0000     ;
0000     ;      1-002   Bill Matthews   1982
0000     ;
0000     ;              Added conversions from non-zero scaled fixed binary to and
0000     ;              from all other data types.
0000     ;
0000     ;      1-003   Bill Matthews   29-September-1982
0000     ;
0000     ;              Invoke macros $defdat and rtshare instead of $defopr and share.
0000     ;
0000     ;      1-004   Bill Matthews   09-March-1983
0000     ;
0000     ;              Add convert from d_float to g_float and convert g_float to d_float
```

```
0000    ;               previously error was signalled.
0000    ;
0000    ;       1-005   Bill Matthews   16-June-1983
0000    ;
0000    ;               Fix bug in fixed binary to fixed decimal conversion
0000    ;
0000    ;       1-006   Dave Blickstein 13-April-1984
0000    ;
0000    ;               Changed float binary to fixed decimal to truncate instead
0000    ;               of round.   SPR #11-66476
0000    ;
0000    ;       1-007   Dave Blickstein 18-April-1984
0000    ;
0000    ;               Fixed bug in float binary to fixed decimal that caused a
0000    ;               decimal overflow at run-time.   The ASHP instruction was
0000    ;               interpreting a large negative shift factor as a positive
0000    ;               shift.  This was because the ASHP interprets the shift factor
0000    ;               as a byte.   Bugs note #8.  Test program: BUGS8.
0000    ; --
0000
0000    ;
0000    ; external definitions
0000    ;
0000            $defdat                         ; define data types
0000            $psldef                         ; define psl bits
0000            $defpic                         ; define picture node offsets
0000            $chfdef                         ; condition handler offsets
0000            $defstk                         ; stack offsets
0000    ;
0000    ; local macros
0000    ;
0000            .macro casetab a
0000            .word   a-casebase
0000            .endm
0000
0000            .macro  eo$insert,char
0000            .byte   ^x44,char
0000            .endm   eo$insert
0000
0000            .macro  eo$store_sign
0000            .byte   4
0000            .endm   eo$store_sign
0000
0000            .macro  eo$fill,rept
0000            .byte   <^x80+rept>
0000            .endm   eo$fill
0000
0000            .macro  eo$move,rept
0000            .byte   <^x90+rept>
0000            .endm   eo$move
0000
0000            .macro  eo$float,rept
0000            .byte   <^xa0+rept>
0000            .endm   eo$float
0000
0000            .macro  eo$end_float
0000            .byte   1
```

```
                            0000             .endm   eo$end_float
                            0000
                            0000             .macro  eo$blank_zero,len
                            0000             .byte   45,len
                            0000             .endm   eo$blank_zero
                            0000
                            0000             .macro  eo$replace_sign,len
                            0000             .byte   46,len
                            0000             .endm   eo$replace_sign
                            0000
                            0000             .macro  eo$load_fill,char
                            0000             .byte   40,char
                            0000             .endm   eo$load_fill
                            0000
                            0000             .macro  eo$load_sign,char
                            0000             .byte   41,char
                            0000             .endm   eo$load_sign
                            0000
                            0000             .macro  ec$load_plus,char
                            0000             .byte   42,char
                            0000             .endm   eo$load_plus
                            0000
                            0000             .macro  eo$load_minus,char
                            0000             .byte   43,char
                            0000             .endm   eo$load_minus
                            0000
                            0000             .macro  eo$clear_signif
                            0000             .byte   2
                            0000             .endm   eo$clear_signif
                            0000
                            0000             .macro  eo$set_signif
                            0000             .byte   3
                            0000             .endm   eo$set_signif
                            0000
                            0000             .macro  eo$adjust_input,len
                            0000             .byte   47,len
                            0000             .endm   eo$adjust_input
                            0000
                            0000             .macro  eo$end
                            0000             .byte   0
                            0000             .endm   eo$end
                            0000
                            0000       ;
                            0000       ; local data
                            0000       ;
                            0000
                            0000             rtshare
                            0000
                            0000       d_power_of_10:
  00000000 00004080         0000             .double 1e0
  CCCDCCCC CCCC3ECC         0008             .double 1e-1
  A3D73D70 D70A3D23         0010             .double 1e-2
  4FDF978D 126E3B83         0018             .double 1e-3
  196558E2 B71739D1         0020             .double 1e-4
  4784471B C5AC3827         0028             .double 1e-5
  6C6A05AF 37BD3686         0030             .double 1e-6
  7A43D5E5 BF9434D6         0038             .double 1e-7
```

```
61CF1184 CC77332B  0040          .double 1e-8
B4A64136 705F3189  0048          .double 1e-9
EDD6CEBD E6FE2FDB  0050          .double 1e-10
24AB0BCB EBFF2E2F  0058          .double 1e-11
5089096F BCCC2C8C  0060          .double 1e-12
B40E424B 2E132AE1  0068          .double 1e-13
5CD83509 24DC2934  0070          .double 1e-14
B0ADF73A 1D7C2790  0078          .double 1e-15
4DE1BEC4 959425E6  0080          .double 1e-16
A4B43236 77AA2438  0088          .double 1e-17
1D5D8E92 92EE2293  0090          .double 1e-18
95627DB6 1E4A20EC  0098          .double 1e-19
111B6492 E5081F3C  00A0          .double 1e-20
DA7C5074 1DA01D97  00A8          .double 1e-21
F72D80BA C9001BF1  00B0          .double 1e-22
928A0095 6D9A1A41  00B8          .double 1e-23
753BCD44 BE14189A  00C0          .double 1e-24
EEC5AED3 968716F7  00C8          .double 1e-25
589E2576 12061546  00D0          .double 1e-26
E07EB791 74D1139E  00D8          .double 1e-27
00CAF283 87B511FD  00E0          .double 1e-28
9A3BF535 D2F7104A  00E8          .double 1e-29
14FCF75E 425F0EA2  00F0          .double 1e-30
43FD2C4B CEB30D01  00F8          .double 1e-31
                   0100    ;
                   0100    h_power_of_10:
                   0100    ;
00000000 00004001  0100          .quad   ^x0000000000004001
00000000 00000000  0108          .quad   ^X0000000000000000
                   0110    ;
99999999 99993FFD  0110          .quad   ^x9999999999993FFD
999A9999 99999999  0118          .quad   ^X999A999999999999
                   0120    ;
E147147A 47AE3FFA  0120          .quad   ^xE147147A47AE3FFA
147B47AE 7AE1AE14  0128          .quad   ^X147B47AE7AE1AE14
                   0130    ;
1A9FDD2F 06243FF7  0130          .quad   ^x1A9FDD2F06243FF7
10623958 C8B4BE76  0138          .quad   ^X10623958C8B4BE76
                   0140    ;
C4322EB1 A36E3FF3  0140          .quad   ^xC4322EB1A36E3FF3
809DC226 A786CA57  0148          .quad   ^X809DC226A786CA57
                   0150    ;
368F588E 4F8B3FF0  0150          .quad   ^x368F588E4F8B3FF0
66E401B8 1F9F0846  0158          .quad   ^X66E401B81F9F0846
                   0160    ;
5ED87A0B 0C6F3FED  0160          .quad   ^x5ED87A0B0C6F3FED
85833493 4C7FD36B  0168          .quad   ^X858334934C7FD36B
                   0170    ;
CAF429AB AD7F3FE9  0170          .quad   ^xCAF429ABAD7F3FE9
08D220EC 7A658578  0178          .quad   ^X08D220EC7A658578
                   0180    ;
08C3EE23 57983FE6  0180          .quad   ^x08C3EE2357983FE6
6D751A56 FB849DF9  0188          .quad   ^X6D751A56FB849DF9
                   0190    ;
6D69BE82 12E03FE3  0190          .quad   ^x6D69BE8212E03FE3
F12A1511 62D04B2E  0198          .quad   ^XF12A151162D04B2E
                   01A0    ;
```

```
7BDBFD9D B7CD3FDF  01A0    229          .quad    ^x7BDBFD9DB7CD3FDF
B511881C 6AE6AB7D  01A8    230          .quad    ^XB511881C6AE6AB7D
                   01B0    231 ;
9649FE17 5FD73FDC  01B0    232          .quad    ^x9649FE175FD73FDC
2A74D34A EF1E55FD  01B8    233          .quad    ^X2A74D34AEF1E55FD
                   01C0    234 ;
DEA19812 19793FD9  01C0    235          .quad    ^xDEA1981219793FD9
885D0F6E F27F1197  01C8    236          .quad    ^X885D0F6EF27F1197
                   01D0    237 ;
97682684 C25C3FD5  01D0    238          .quad    ^x97682684C25C3FD5
0D614BE4 50CB1C26  01D8    239          .quad    ^X0D614BE450CB1C26
                   01E0    240 ;
12B9B86A 68493FD2  01E0    241          .quad    ^x12B9B86A68493FD2
3DE70983 A709B01E  01E8    242          .quad    ^X3DE70983A709B01E
                   01F0    243 ;
7561F9EE 203A3FCF  01F0    244          .quad    ^x7561F9EE203A3FCF
97EC6E02 1F3A59B2  01F8    245          .quad    ^X97EC6E021F3A59B2
                   0200    246 ;
889B297D CD2B3FCB  0200    247          .quad    ^x889B297DCD2B3FCB
F3137CD0 985DC2B6  0208    248          .quad    ^XF3137CD0985DC2B6
                   0210    249 ;
6D495464 70EF3FC8  0210    250          .quad    ^x6D49546470EF3FC8
F5A9FD73 137D6892  0218    251          .quad    ^XF5A9FD73137D6892
                   0220    252 ;
243ADD1D 27253FC5  0220    253          .quad    ^x243ADD1D27253FC5
C487645C 75FEBA0E  0228    254          .quad    ^XC487645C75FEBA0E
                   0230    255 ;
6D2A94FB D83C3FC1  0230    256          .quad    ^x6D2A94FBD83C3FC1
A0D8D3C7 5663C34A  0238    257          .quad    ^XA0D8D3C75663C34A
                   0240    258 ;
242210C9 79CA3FBE  0240    259          .quad    ^x242210C979CA3FBE
4D7A7639 11E935D5  0248    260          .quad    ^X4D7A763911E935D5
                   0250    261 ;
E9B440A0 2E3B3FBB  0250    262          .quad    ^xE9B440A02E3B3FBB
D795F82D A7EDF7DD  0258    263          .quad    ^XD795F82DA7EDF7DD
                   0260    264 ;
75EE0101 E3923FB7  0260    265          .quad    ^x75EE0101E3923FB7
25BB8D16 A6495962  0268    266          .quad    ^X25BB8D16A6495962
                   0270    267 ;
2B253401 82DB3FB4  0270    268          .quad    ^x2B25340182DB3FB4
1E2F0A78 EB6E144E  0278    269          .quad    ^X1E2F0A78EB6E144E
                   0280    270 ;
88EA299A 357C3FB1  0280    271          .quad    ^x88EA299A357C3FB1
E4F2D52C 892476A5  0288    272          .quad    ^XE4F2D52C892476A5
                   0290    273 ;
A7DD0F5D EF2D3FAD  0290    274          .quad    ^xA7DD0F5DEF2D3FAD
07EABB7B 75078AA2  0298    275          .quad    ^X07EABB7B75078AA2
                   02A0    276 ;
ECB10C4A 8C243FAA  02A0    277          .quad    ^xECB10C4A8C243FAA
065595FC 2A6C3BB5  02A8    278          .quad    ^X065595FC2A6C3BB5
                   02B0    279 ;
23C0A36F 3CE93FA7  02B0    280          .quad    ^x23C0A36F3CE93FA7
9EAA44C9 EEBDFC90  02B8    281          .quad    ^X9EAA44C9EEBDFC90
                   02C0    282 ;
06016BE5 FB0F3FA3  02C0    283          .quad    ^x06016BE5FB0F3FA3
31113ADC 1795941B  02C8    284          .quad    ^X31113ADC1795941B
                   02D0    285 ;
```

```
6B34EFEA 95A53FA0  02D0   286          .quad   ^x6B34EFEA95A53FA0
2741C8B0 12DD767C  02D8   287          .quad   ^X2741C8B012DD767C
                   02E0   288 ;
BC29BFEE 44843F9D  02E0   289          .quad   ^xBC29BFEE44843F9D
529A06F3 424BF863  02E8   290          .quad   ^X529A06F3424BF863
                   02F0   291 ;
96876658 039D3F9A  02F0   292          .quad   ^x96876658039D3F9A
0EE29F29 01D5F9E9  02F8   293          .quad   ^X0EE29F2901D5F9E9
```

```
      0300   295 reverse_bit_tbl:
00    0300   296            .byte   ^b00000000      ;00000000
80    0301   297            .byte   ^b10000000      ;00000001
40    0302   298            .byte   ^b01000000      ;00000010
C0    0303   299            .byte   ^b11000000      ;00000011
20    0304   300            .byte   ^b00100000      ;00000100
A0    0305   301            .byte   ^b10100000      ;00000101
60    0306   302            .byte   ^b01100000      ;00000110
E0    0307   303            .byte   ^b11100000      ;00000111
10    0308   304            .byte   ^b00010000      ;00001000
90    0309   305            .byte   ^b10010000      ;00001001
50    030A   306            .byte   ^b01010000      ;00001010
D0    030B   307            .byte   ^b11010000      ;00001011
30    030C   308            .byte   ^b00110000      ;00001100
B0    030D   309            .byte   ^b10110000      ;00001101
70    030E   310            .byte   ^b01110000      ;00001110
F0    030F   311            .byte   ^b11110000      ;00001111
08    0310   312            .byte   ^b00001000      ;00010000
88    0311   313            .byte   ^b10001000      ;00010001
48    0312   314            .byte   ^b01001000      ;00010010
C8    0313   315            .byte   ^b11001000      ;00010011
28    0314   316            .byte   ^b00101000      ;00010100
A8    0315   317            .byte   ^b10101000      ;00010101
68    0316   318            .byte   ^b01101000      ;00010110
E8    0317   319            .byte   ^b11101000      ;00010111
18    0318   320            .byte   ^b00011000      ;00011000
98    0319   321            .byte   ^b10011000      ;00011001
58    031A   322            .byte   ^b01011000      ;00011010
D8    031B   323            .byte   ^b11011000      ;00011011
38    031C   324            .byte   ^b00111000      ;00011100
B8    031D   325            .byte   ^b10111000      ;00011101
78    031E   326            .byte   ^b01111000      ;00011110
F8    031F   327            .byte   ^b11111000      ;00011111
04    0320   328            .byte   ^b00000100      ;00100000
84    0321   329            .byte   ^b10000100      ;00100001
44    0322   330            .byte   ^b01000100      ;00100010
C4    0323   331            .byte   ^b11000100      ;00100011
24    0324   332            .byte   ^b00100100      ;00100100
A4    0325   333            .byte   ^b10100100      ;00100101
64    0326   334            .byte   ^b01100100      ;00100110
E4    0327   335            .byte   ^b11100100      ;00100111
14    0328   336            .byte   ^b00010100      ;00101000
94    0329   337            .byte   ^b10010100      ;00101001
54    032A   338            .byte   ^b01010100      ;00101010
D4    032B   339            .byte   ^b11010100      ;00101011
34    032C   340            .byte   ^b00110100      ;00101100
B4    032D   341            .byte   ^b10110100      ;00101101
74    032E   342            .byte   ^b01110100      ;00101110
F4    032F   343            .byte   ^b11110100      ;00101111
0C    0330   344            .byte   ^b00001100      ;00110000
8C    0331   345            .byte   ^b10001100      ;00110001
4C    0332   346            .byte   ^b01001100      ;00110010
CC    0333   347            .byte   ^b11001100      ;00110011
2C    0334   348            .byte   ^b00101100      ;00110100
AC    0335   349            .byte   ^b10101100      ;00110101
6C    0336   350            .byte   ^b01101100      ;00110110
EC    0337   351            .byte   ^b11101100      ;00110111
```

```
1C  0338   352          .byte   ^b00011100        ;00111000
9C  0339   353          .byte   ^b10011100        ;00111001
5C  033A   354          .byte   ^b01011100        ;00111010
DC  033B   355          .byte   ^b11011100        ;00111011
3C  033C   356          .byte   ^b00111100        ;00111100
BC  033D   357          .byte   ^b10111100        ;00111101
7C  033E   358          .byte   ^b01111100        ;00111110
FC  033F   359          .byte   ^b11111100        ;00111111
02  0340   360          .byte   ^b00000010        ;01000000
82  0341   361          .byte   ^b10000010        ;01000001
42  0342   362          .byte   ^b01000010        ;01000010
C2  0343   363          .byte   ^b11000010        ;01000011
22  0344   364          .byte   ^b00100010        ;01000100
A2  0345   365          .byte   ^b10100010        ;01000101
62  0346   366          .byte   ^b01100010        ;01000110
E2  0347   367          .byte   ^b11100010        ;01000111
12  0348   368          .byte   ^b00010010        ;01001000
92  0349   369          .byte   ^b10010010        ;01001001
52  034A   370          .byte   ^b01010010        ;01001010
D2  034B   371          .byte   ^b11010010        ;01001011
32  034C   372          .byte   ^b00110010        ;01001100
B2  034D   373          .byte   ^b10110010        ;01001101
72  034E   374          .byte   ^b01110010        ;01001110
F2  034F   375          .byte   ^b11110010        ;01001111
0A  0350   376          .byte   ^b00001010        ;01010000
8A  0351   377          .byte   ^b10001010        ;01010001
4A  0352   378          .byte   ^b01001010        ;01010010
CA  0353   379          .byte   ^b11001010        ;01010011
2A  0354   380          .byte   ^b00101010        ;01010100
AA  035    381          .byte   ^b10101010        ;01010101
6A  0356   382          .byte   ^b01101010        ;01010110
EA  0357   383          .byte   ^b11101010        ;01010111
1A  0358   384          .byte   ^b00011010        ;01011000
9A  0359   385          .byte   ^b10011010        ;01011001
5A  035A   386          .byte   ^b01011010        ;01011010
DA  035B   387          .byte   ^b11011010        ;01011011
3A  035C   388          .byte   ^b00111010        ;01011100
BA  035D   389          .byte   ^b10111010        ;01011101
7A  035E   390          .byte   ^b01111010        ;01011110
FA  035F   391          .byte   ^b11111010        ;01011111
06  0360   392          .byte   ^b00000110        ;01100000
86  0361   393          .byte   ^b10000110        ;01100001
46  0362   394          .byte   ^b01000110        ;01100010
C6  0363   395          .byte   ^b11000110        ;01100011
26  0364   396          .byte   ^b00100110        ;01100100
A6  0365   397          .byte   ^b10100110        ;01100101
66  0366   398          .byte   ^b01100110        ;01100110
E6  0367   399          .byte   ^b11100110        ;01100111
16  0368   400          .byte   ^b00010110        ;01101000
96  0369   401          .byte   ^b10010110        ;01101001
56  036A   402          .byte   ^b01010110        ;01101010
D6  036B   403          .byte   ^b11010110        ;01101011
36  036C   404          .byte   ^b00110110        ;01101100
B6  036D   405          .byte   ^b10110110        ;01101101
76  036E   406          .byte   ^b01110110        ;01101110
F6  036F   407          .byte   ^b11110110        ;01101111
0E  0370   408          .byte   ^b00001110        ;01110000
```

```
8E  0371   409         .byte   ^b10001110      ;01110001
4E  0372   410         .byte   ^b01001110      ;01110010
CE  0373   411         .byte   ^b11001110      ;01110011
2E  0374   412         .byte   ^b00101110      ;01110100
AE  0375   413         .byte   ^b10101110      ;01110101
6E  0376   414         .byte   ^b01101110      ;01110110
EE  0377   415         .byte   ^b11101110      ;01110111
1E  0378   416         .byte   ^b00011110      ;01111000
9E  0379   417         .byte   ^b10011110      ;01111001
5E  037A   418         .byte   ^b01011110      ;01111010
DE  037B   419         .byte   ^b11011110      ;01111011
3E  037C   420         .byte   ^b00111110      ;01111100
BE  037D   421         .byte   ^b10111110      ;01111101
7E  037E   422         .byte   ^b01111110      ;01111110
FE  037F   423         .byte   ^b11111110      ;01111111
01  0380   424         .byte   ^b00000001      ;10000000
81  0381   425         .byte   ^b10000001      ;10000001
41  0382   426         .byte   ^b01000001      ;10000010
C1  0383   427         .byte   ^b11000001      ;10000011
21  0384   428         .byte   ^b00100001      ;10000100
A1  0385   429         .byte   ^b10100001      ;10000101
61  0386   430         .byte   ^b01100001      ;10000110
E1  0387   431         .byte   ^b11100001      ;10000111
11  0388   432         .byte   ^b00010001      ;10001000
91  0389   433         .byte   ^b10010001      ;10001001
51  038A   434         .byte   ^b01010001      ;10001010
D1  038B   435         .byte   ^b11010001      ;10001011
31  038C   436         .byte   ^b00110001      ;10001100
B1  038D   437         .byte   ^b10110001      ;10001101
71  038E   438         .byte   ^b01110001      ;10001110
F1  038F   439         .byte   ^b11110001      ;10001111
09  0390   440         .byte   ^b00001001      ;10010000
89  0391   441         .byte   ^b10001001      ;10010001
49  0392   442         .byte   ^b01001001      ;10010010
C9  0393   443         .byte   ^b11001001      ;10010011
29  0394   444         .byte   ^b00101001      ;10010100
A9  0395   445         .byte   ^b10101001      ;10010101
69  0396   446         .byte   ^b01101001      ;10010110
E9  0397   447         .byte   ^b11101001      ;10010111
19  0398   448         .byte   ^b00011001      ;10011000
99  0399   449         .byte   ^b10011001      ;10011001
59  039A   450         .byte   ^b01011001      ;10011010
D9  039B   451         .byte   ^b11011001      ;10011011
39  039C   452         .byte   ^b00111001      ;10011100
B9  039D   453         .byte   ^b10111001      ;10011101
79  039E   454         .byte   ^b01111001      ;10011110
F9  039F   455         .byte   ^b11111001      ;10011111
05  03A0   456         .byte   ^b00000101      ;10100000
85  03A1   457         .byte   ^b10000101      ;10100001
45  03A2   458         .byte   ^b01000101      ;10100010
C5  03A3   459         .byte   ^b11000101      ;10100011
25  03A4   460         .byte   ^b00100101      ;10100100
A5  03A5   461         .byte   ^b10100101      ;10100101
65  03A6   462         .byte   ^b01100101      ;10100110
E5  03A7   463         .byte   ^b11100101      ;10100111
15  03A8   464         .byte   ^b00010101      ;10101000
95  03A9   465         .byte   ^b10010101      ;10101001
```

```
55  03AA   466        .byte   ^b01010101       ;10101010
D5  03AB   467        .byte   ^b11010101       ;10101011
35  03AC   468        .byte   ^b00110101       ;10101100
B5  03AD   469        .byte   ^b10110101       ;10101101
75  03AE   470        .byte   ^b01110101       ;10101110
F5  03AF   471        .byte   ^b11110101       ;10101111
0D  03B0   472        .byte   ^b00001101       ;10110000
8D  03B1   473        .byte   ^b10001101       ;10110001
4D  03B2   474        .byte   ^b01001101       ;10110010
CD  03B3   475        .byte   ^b11001101       ;10110011
2D  03B4   476        .byte   ^b00101101       ;10110100
AD  03B5   477        .byte   ^b10101101       ;10110101
6D  03B6   478        .byte   ^b01101101       ;10110110
ED  03B7   479        .byte   ^b11101101       ;10110111
1D  03B8   480        .byte   ^b00011101       ;10111000
9D  03B9   481        .byte   ^b10011101       ;10111001
5D  03BA   482        .byte   ^b01011101       ;10111010
DD  03BB   483        .byte   ^b11011101       ;10111011
3D  03BC   484        .byte   ^b00111101       ;10111100
BD  03BD   485        .byte   ^b10111101       ;10111101
7D  03BE   486        .byte   ^b01111101       ;10111110
FD  03BF   487        .byte   ^b11111101       ;10111111
03  03C0   488        .byte   ^b00000011       ;11000000
83  03C1   489        .byte   ^b10000011       ;11000001
43  03C2   490        .byte   ^b01000011       ;11000010
C3  03C3   491        .byte   ^b11000011       ;11000011
23  03C4   492        .byte   ^b00100011       ;11000100
A3  03C5   493        .byte   ^b10100011       ;11000101
63  03C6   494        .byte   ^b01100011       ;11000110
E3  03C7   495        .byte   ^b11100011       ;11000111
13  03C8   496        .byte   ^b00010011       ;11001000
93  03C9   497        .byte   ^b10010011       ;11001001
53  03CA   498        .byte   ^b01010011       ;11001010
D3  03CB   499        .byte   ^b11010011       ;11001011
33  03CC   500        .byte   ^b00110011       ;11001100
B3  03CD   501        .byte   ^b10110011       ;11001101
73  03CE   502        .byte   ^b01110011       ;11001110
F3  03CF   503        .byte   ^b11110011       ;11001111
0B  03D0   504        .byte   ^b00001011       ;11010000
8B  03D1   505        .byte   ^b10001011       ;11010001
4B  03D2   506        .byte   ^b01001011       ;11010010
CB  03D3   507        .byte   ^b11001011       ;11010011
2B  03D4   508        .byte   ^b00101011       ;11010100
AB  03D5   509        .byte   ^b10101011       ;11010101
6B  03D6   510        .byte   ^b01101011       ;11010110
EB  03D7   511        .byte   ^b11101011       ;11010111
1B  03D8   512        .byte   ^b00011011       ;11011000
9B  03D9   513        .byte   ^b10011011       ;11011001
5B  03DA   514        .byte   ^b01011011       ;11011010
DB  03DB   515        .byte   ^b11011011       ;11011011
3B  03DC   516        .byte   ^b00111011       ;11011100
BB  03DD   517        .byte   ^b10111011       ;11011101
7B  03DE   518        .byte   ^b01111011       ;11011110
FB  03DF   519        .byte   ^b11111011       ;11011111
07  03E0   520        .byte   ^b00000111       ;11100000
87  03E1   521        .byte   ^b10000111       ;11100001
47  03E2   522        .byte   ^b01000111       ;11100010
```

```
C7  03E3   523         .byte   ^b11000111      ;11100011
27  03E4   524         .byte   ^b00100111      ;11100100
A7  03E5   525         .byte   ^b10100111      ;11100101
67  03E6   526         .byte   ^b01100111      ;11100110
E7  03E7   527         .byte   ^b11100111      ;11100111
17  03E8   528         .byte   ^b00010111      ;11101000
97  03E9   529         .byte   ^b10010111      ;11101001
57  03EA   530         .byte   ^b01010111      ;11101010
D7  03EB   531         .byte   ^b11010111      ;11101011
37  03EC   532         .byte   ^b00110111      ;11101100
B7  03ED   533         .byte   ^b10110111      ;11101101
77  03EE   534         .byte   ^b01110111      ;11101110
F7  03EF   535         .byte   ^b11110111      ;11101111
0F  03F0   536         .byte   ^b00001111      ;11110000
8F  03F1   537         .byte   ^b10001111      ;11110001
4F  03F2   538         .byte   ^b01001111      ;11110010
CF  03F3   539         .byte   ^b11001111      ;11110011
2F  03F4   540         .byte   ^b00101111      ;11110100
AF  03F5   541         .byte   ^b10101111      ;11110101
6F  03F6   542         .byte   ^b01101111      ;11110110
EF  03F7   543         .byte   ^b11101111      ;11110111
1F  03F8   544         .byte   ^b00011111      ;11111000
9F  03F9   545         .byte   ^b10011111      ;11111001
5F  03FA   546         .byte   ^b01011111      ;11111010
DF  03FB   547         .byte   ^b11011111      ;11111011
3F  03FC   548         .byte   ^b00111111      ;11111100
BF  03FD   549         .byte   ^b10111111      ;11111101
7F  03FE   550         .byte   ^b01111111      ;11111110
FF  03FF   551         .byte   ^b11111111      ;11111111
    0400   552 ;
```

```
                        0400            .sbttl pli$cvrt_any - convert any data type
                        0400     ; ++
                        0400     ; pli$cvrt_any - convert any data type
                        0400     ;
                        0400     ; functional descritpion:
                        0400     ;
                        0400     ; This dispatch routine and the individual conversion routines represent
                        0400     ; an any to any conversion package. The philosophy is to convert wherever
                        0400     ; possible. If the arguments describe an undefined data type or out of range size
                        0400     ; of a known data type then the caller is in error and the general ERROR
                        0400     ; condition is signalled. Otherwise the conversion is done with expansion
                        0400     ; or truncation where necessary.
                        0400     ;
                        0400     ; This routine sets up the arguments and dispatches to the proper routine
                        0400     ; based on the data types of the source and destination.
                        0400     ;
                        0400     ; inputs: ( arguments are immediate )
                        0400     ;
                        0400     ;       (ap) = 8
                        0400     ;       4(ap) = address of the address of the source
                        0400     ;       8(ap) = data type of source
                        0400     ;       12(ap) = size (p,q) of source
                        0400     ;       16(ap) = bit offset of source, if necessary
                        0400     ;       20(ap) = address of the address of the target
                        0400     ;       24(ap) = data type of target
                        0400     ;       28(ap) = size (p,q) of target
                        0400     ;       32(ap) = bit offset of target, if necessary
                        0400     ;
                        0400     ; outputs:
                        0400     ;
                        0400     ;        The source is converted to the destination.
                        0400     ; --
                  CFFC  0400            .entry pli$cvrt_any,^m<iv,dv,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
      0040 8F       B8  0402            bispsw  #psl$m_fu               ; enable underflow
                        0406     ;
                        0406     ; merge data types and check for invalid types
                        0406     ;
 54     08 AC       9A  0406            movzbl  8(ap),r4                ; get source data type
 56     18 AC       9A  040A            movzbl  24(ap),r6               ; get the target data type
     0E     56      91  040E            cmpb    r6,#dat_k_bit_align     ; in range?
            42      1A  0411            bgtru   error                   ; if gtru then error
            02      12  0413            bneq    5$                      ; if neq then continue
            56      97  0415            decb    r6                      ; squeeze out bit varying
     0E     54      91  0417     5$:    cmpb    r4,#dat_k_bit_align     ; in range?
            39      1A  041A            bgtru   error                   ; if gtru then error
            02      12  041C            bneq    10$                     ; if neq then continue
            54      97  041E            decb    r4                      ; squeeze out bit varying
     05     54      91  0420     10$:   cmpb    r4,#dat_k_flt_dec       ; simplify range, by making
                        0423                                            ; making codes contiguous
            03      1B  0423            blequ   15$                     ;
     54     04      82  0425            subb    #4,r4;
     05     56      91  0428     15$:   cmpb    r6,#dat_k_flt_dec       ;
            03      1B  042B            blequ   20$                     ;
     56     04      82  042D            subb    #4,r6;
                        0430                                            ; find table entry
            54      97  0430     20$:   decb    r4                      ; adjust to zero
            56      97  0432            decb    r6                      ;
```

```
                54  09  84 0434   611           mulb    #9,r4                          ;
                54  56  80 0437   612           addb    r6,r4                          ;
                           043A   613 ;
                           043A   614 ; set up remainder of arguments
                           043A   615 ;
             50   04 BC  D0 043A   616           movl    @4(ap),r0                      ; address source
             51   0C AC  D0 043E   617           movl    12(ap),r1                      ; get source size
             55   10 AC  D0 0442   618           movl    16(ap),r5                      ; get source bit offset
             52   14 BC  D0 0446   619           movl    @20(ap),r2                     ; get target address
             53   1C AC  D0 044A   620           movl    28(ap),r3                      ; get target size
             56   20 AC  D0 044E   621           movl    32(ap),r6                      ; get target bit offset
                    0020  31 0452   622           brw     case_on_type                   ; continue
                           0455   623 ;
                           0455   624 ; fatal - undefined conversion
                           0455   625 ;
                           0455   626 error:
          00000000'8F  DD 0455   627           pushl   #pli$_cnverr                   ; actual error code
                   7E  D4 045B   628           clrl    -(sp)
          00000000'8F  DD 045D   629           pushl   #pli$_error
                   50  7C 0463   630           clrq    r0                             ; set no value - also no fcb
    00000000'GF    03  FB 0465   631           calls   #3,g^lib$signal                ; signal the error
                   50  7C 046C   632           clrq    r0                             ; set no value
                       04 046E   633           ret
```

```
                          046F        .sbttl  pli$cvrt_cg - perform out of line conversion
                          046F  ; ++
                          046F  ; pli$cvrt_cg - perform a conversion
                          046F  ;
                          046F  ; functional description:
                          046F  ;
                          046F  ; This is the entry to the conversion logic for the codegenerator.
                          046F  ;
                          046F  ; This routine is called to preserve trace back data, but the arguments are passed
                          046F  ; in registers.
                          046F  ;
                          046F
                          046F  ; inputs:
                          046F  ;
                          046F  ;       r0 = address of the source
                          046F  ;       r1 = size of the source
                          0'6F  ;       r2 = address of the destination
                          046F  ;       r3 = size of the destination
                          046F  ;       r4 = case table index
                          046F  ;       r5 = bit offset of source if any
                          046F  ;       r6 = bit offset of destination if any
                          046F  ;
                          046F  ; outputs:
                          046F  ;
                          046F  ;       The operation is done.
                          046F  ; ***********************************************************
                          046F  ;
                          046F  ; WARNING
                          046F  ;
                          046F  ; Do not change this interface without the proper changes to the codegenerator.
                          046F  ;
                          046F  ; ***********************************************************
                          046F  ; --
                    CFF0  046F        .entry  pli$cvrt_cg_r3,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                          0471  ;
                          0471  ; enable arithmetic traps
                          0471  ;
          0040 8F    B8   0471        bispsw  #psl$m_fu
                          0475  case_on_type:
                          0475  ;
                          0475  ; NOTE WELL:    DO NOT CHANGE THIS CASE TABLE WITHOUT CHANGING THE CODE
                          0475  ;               GENERATOR, THE FORMAT CONVERSION ROUTINES, AND THE DEFINITION
                          0475  ;               OF $DEFCVTIND IN PL1MAC.MLB. IF YOU ADD ENTRIES, YOU WILL
                          0475  ;               ALSO WANT TO CHANGE THE GET AND PUT ITEM ROUTINES.
   50 8F    00    54   8F 0475        caseb   r4,#0,#80               ;
                0000047A  047A  casebase=.
                          047A        casetab picpic
                          047C        casetab picfixb
                          047E        casetab picfltb
                          0480        casetab picfixd
                          0482        casetab picfltd
                          0484        casetab picchar
                          0486        casetab picvcha
                          0488        casetab picbit
                          048A        casetab picabit
                          048C        casetab fixbpic
                          048E        casetab fixbfixb
```

```
     0490   692         casetab fixbfltb
     0492   693         casetab fixbfixd
     0494   694         casetab fixbfltd
     0496   695         casetab fixbchar
     0498   696         casetab fixbvcha
     049A   697         casetab fixbbit
     049C   698         casetab fixbabit
     049E   699         casetab fltbpic
     04A0   700         casetab fltbfixb
     04A2   701         casetab fltbfltb
     04A4   702         casetab fltbfixd
     04A6   703         casetab fltbfltd
     04A8   704         casetab fltbchar
     04AA   705         casetab fltbvcha
     04AC   706         casetab fltbbit
     04AE   707         casetab fltbabit
     04B0   708         casetab fixdpic
     04B2   709         casetab fixdfixb
     04B4   710         casetab fixdfltb
     04B6   711         casetab fixdfixd
     04B8   712         casetab fixdfltd
     04BA   713         casetab fixdchar
     04BC   714         casetab fixdvcha
     04BE   715         casetab fixdbit
     04C0   716         casetab fixdabit
     04C2   717         casetab fltdpic
     04C4   718         casetab fltdfixb
     04C6   719         casetab fltdfltb
     04C8   720         casetab fltdfixd
     04CA   721         casetab fltdfltd
     04CC   722         casetab fltdchar
     04CE   723         casetab fltdvcha
     04D0   724         casetab fltdbit
     04D2   725         casetab fltdabit
     04D4   726         casetab charpic
     04D6   727         casetab charfixb
     04D8   728         casetab charfltb
     04DA   729         casetab charfixd
     04DC   730         casetab charfltd
     04DE   731         casetab charchar
     04E0   732         casetab charvcha
     04E2   733         casetab charbit
     04E4   734         casetab charabit
     04E6   735         casetab vchapic
     04E8   736         casetab vchafixb
     04EA   737         casetab vchafltb
     04EC   738         casetab vchafixd
     04EE   739         casetab vchafltd
     04F0   740         casetab vchachar
     04F2   741         casetab vchavcha
     04F4   742         casetab vchabit
     04F6   743         casetab vchaabit
     04F8   744         casetab bitpic
     04FA   745         casetab bitfixb
     04FC   746         casetab bitfltb
     04FE   747         casetab bitfixd
     0500   748         casetab bitfltd
```

```
               0502   749         casetab bitchar
               0504   750         casetab bitvcha
               0506   751         casetab bitbit
               0508   752         casetab bitabit
               050A   753         casetab abitpic
               050C   754         casetab abitfixb
               050E   755         casetab abitfltb
               0510   756         casetab abitfixd
               0512   757         casetab abitfltd
               0514   758         casetab abitchar
               0516   759         casetab abitvcha
               0518   760         casetab abitbit
               051A   761         casetab abitabit
   FF36   31   051C   762         brw     error
```

```
                                   051F   ;++
                                   051F   ;
                                   051F   ; pli$cnvrt_hnd
                                   051F   ;
                                   051F   ; this handler is used by pli$convert routines that may generate
                                   051F   ; reserved operand exceptions; that is, all the char and vchar
                                   051F   ; to arithmetic or bit conversions.  it handles only reserved
                                   051F   ; operand, by signalling a pl/i error with a conversion error
                                   051F   ; subcode.  all other conditions are resignalled.
                                   051F   ;
                                   051F   ; input:
                                   051F   ;       condition argument list
                                   051F   ;
                                   051F   ; output:
                                   051F   ;       if roprand, error is signalled
                                   051F   ;       else condition is resignalled
                                   051F   ;--
                                   051F           .sbttl  pli$cnvrt_hnd   conversion condition handler
                                   051F   ;
                             0000  051F           .entry  pli$cnvrt_hnd,0
                                   0521   ;
   50   00000000'8F     D0         0521           movl    #ss$_resignal,r0          ; assume resignal
           51   04 AC   D0         0528           movl    chf$l_sigarglst(ap),r1    ; address arg list
 00000000'8F    04 A1   D1         052C           cmpl    chf$l_sig_name(r1),#ss$_roprand ; check for roprand
                   3B   12         0534           bneq    30$                       ; if neq, resignal
             51    5D   D0         0536           movl    fp,r1                     ; addr the frame
10 A1   00000571'EF     9E         0539   10$:    movab   30$,stk_l_pc(r1)          ; force pc to be a return statement
           51   0C A1   D0         0541           movl    stk_l_fp(r1),r1           ; get the next frame
   50   0000051F'GF     DE         0545           moval   g^pli$cnvrt_hnd,r0        ; get address
             50    61   D1         054C           cmpl    stk_l_cnd_hnd(r1),r0      ; see if it's our establisher
                   E8   12         054F           bneq    10$                       ; if not, keep looking
                   61   D4         0551           clrl    stk_l_cnd_hnd(r1)         ; else, take away this cond. hndlr
        00000000'8F     DD         0553           pushl   #pli$_cnverr              ; set conversion error subcode
                   7E   D4         0559           clrl    -(sp)
        00000000'8F     DD         055B           pushl   #pli$_error               ; set error condition code
                   50   7C         0561           clrq    r0
    00000000'GF    03   FB         0563           calls   #3,g^lib$signal           ; and signal pli error
   50   00000000'8F     D0         056A           movl    #ss$_continue,r0
                        04         0571   30$:    ret
                        0572             .sbttl  input checking subroutines
                        0572     ;
                        0572     ; chk_fixb_string - check fixed binary for overflow
                        0572     ;
                        0572     chk_fixb_string:                                ; check for overflow condition
                        0572                                                     ; on fixb to string conversion
           55    51  9A 0572             movzbl  r1,r5                           ; get prec
           54    07  D0 0575             movl    #7,r4                           ; assume less than 7 bits
           55    54  D1 0578             cmpl    r4,r5                           ; this range?
                 2B  13 057B             beql    20$                             ; if eql then always ok
                 14  14 057D             bgtr    10$                             ; if gtr then check
           54    0F  D0 057F             movl    #15,r4                          ; assume less than 15 bits
           55    54  D1 0582             cmpl    r4,r5                           ; this range?
                 21  13 0585             beql    20$                             ; if eql then always ok
                 0A  14 0587             bgtr    10$                             ; if gtr then check
           54    1F  D0 0589             movl    #31,r4                          ; assume less than 31 bits
           55    54  D1 058C             cmpl    r4,r5                           ; this range?
                 17  13 058F             beql    20$                             ; if eql then always ok
                 26  19 0591             blss    30$                             ; if less then illegal
           54    55  C2 0593     10$:    subl    r5,r4                           ; get size of sign areai
                 00  DD 0596             pushl   #0                              ; assume positive number
        03 60    55  E1 0598             bbc     r5,(r0),15$                     ; if clear then positive
           6E    01  CE 059C             mnegl   #1,(sp)                         ; set negitive
                 55  D6 059F     15$:    incl    r5                              ; point to bit past the sign bit
  8E    60 54    55  EC 05A1             cmpv    r5,r4,(r0),(sp)+                ; check sign
                 01  12 05A6             bneq    25$                             ; if neq then overflow
                     05 05A8     20$:    rsb                                     ; else done
        00000000'8F  DD 05A9     25$:    pushl   #ss$_intovf                     ; signal error
                 50  7C 05AF             clrq    r0                              ; no value
   00000000'GF   01  FB 05B1             calls   #1,g^lib$signal                 ; signal error
                     04 05B8             ret                                     ; exit
         FE99        31 05B9     30$:    brw     error                           ; signal error condition
                        05BC
                        05BC     ;
                        05BC     ; chk_bit_arith - check for bit to arithmetic overflow
                        05BC     ;
                        05BC             .enabl  lsb
                        05BC     chk_bit_arith:
                 00  11 05BC             brb     10$                             ; continue to verify bits
                        05BE     chk_abit_arith:
           1F    51  D1 05BE     10$:    cmpl    r1,#31                          ; less than 31 bits?
                 27  15 05C1             bleq    30$                             ; then always ok
           51    1F  C2 05C3             subl    #31,r1
     7E    55    51  C1 05C6             addl3   r1,r5,-(sp)                     ; point at last 31 bits, and save
                 50  DD 05CA             pushl   r0                              ; save src base
                 51  D5 05CC     15$:    tstl    r1                              ; more bits to verify?
                 15  13 05CE             beql    20$                             ; if eql then no and string is checked
               11F6  30 05D0             bsbw    get_next_32bits                 ; get the next 32 bits of the string
                 54  D5 05D3             tstl    r4                              ; all 0's?
                 F5  13 05D5             beql    15$                             ; if eql then continue check
        00000000'8F  DD 05D7             pushl   #ss$_intovf                     ;
   00000000'GF   01  FB 05DD             calls   #1,g^lib$signal                 ; signal error
                     04 05E4             ret                                     ;
                 21  BA 05E5     20$:    popr    #^m<r0,r5>                      ; restore desc with adjusted offset
           51    1F  D0 05E7             movl    #31,r1                          ; set max size
                     05 05EA     30$:    rsb
                        05EB
                        05EB          .dsabl  lsb
                        05EB   ;
                        05EB   ; src_fltb_prec - calc floating source context
                        05EB   ;
                        05EB   src_fltb_prec:
        0040 8F     B8  05EB          bispsw  #psl$m_fu                ; enable underflow
     04 51   07     E5  05EF          bbcc    #7,r1,10$                ; test for grand and clear it
        54   02     D0  05F3          movl    #2,r4                    ; set grand context
             05         05F6          rsb
                        05F7   ;
        18   51     D1  05F7   10$:   cmpl    r1,#24                   ; float?
             03     14  05FA          bgtr    20$                      ; if not, br
             54     D4  05FC          clrl    r4                       ; set F float context
             05         05FE          rsb
                        05FF   ;
        35   51     D1  05FF   20$:   cmpl    r1,#53                   ; double?
             04     14  0602          bgtr    30$                      ; if not, br
        54   01     D0  0604          movl    #1,r4                    ; set double context
             05         0607          rsb
                        0608   ;
        54   03     D0  0608   30$:   movl    #3,r4                    ; must be huge
             05         060B          rsb
                        060C   ;
                        060C   ; dest_fltb_prec - calc floating destination context
                        060C   ;
                        060C   dest_fltb_prec:
        0040 8F     B8  060C          bispsw  #psl$m_fu                ; enable underflow
     04 53   07     E5  0610          bbcc    #7,r3,10$                ; test for grand
        57   02     D0  0614          movl    #2,r7                    ; set grand context
             05         0617          rsb
                        0618   ;
        18   53     D1  0618   10$:   cmpl    r3,#24                   ; float?
             03     14  061B          bgtr    20$                      ; if not, br
             57     D4  061D          clrl    r7                       ; set F float context
             05         061F          rsb
                        0620   ;
        35   53     D1  0620   20$:   cmpl    r3,#53                   ; double?
             04     14  0623          bgtr    30$                      ; if not, br
        57   01     D0  0625          movl    #1,r7                    ; set double context
             05         0628          rsb
                        0629   ;
        57   03     D0  0629   30$:   movl    #3,r7                    ; must be huge
             05         062C          rsb
                        062D   ;
                        062D   ; src_fltd_prec - calc floating decimal source context
                        062D   ;
                        062D   src_fltd_prec:
        0040 8F     B8  062D          bispsw  #psl$m_fu                ; enable underflow
     04 51   07     E5  0631          bbcc    #7,r1,10$                ; test for grand
        54   02     D0  0635          movl    #2,r4                    ; set grand context
             05         0638          rsb
                        0639   ;
        07   51     D1  0639   10$:   cmpl    r1,#7                    ; float?
             03     14  063C          bgtr    20$                      ; if not, br
             54     D4  063E          clrl    r4                       ; set F float context
             05         0640          rsb
                        0641   ;
        0F   51   D1  0641    918 20$:    cmpl    r1,#15                  ; double?
             04   14  0644    919         bgtr    30$                     ; if not, br
        54   01   D0  0646    920         movl    #1,r4                   ; set double context
                  05  0649    921         rsb
                      064A    922 ;
        54   03   D0  064A    923 30$:    movl    #3,r4                   ; must be huge
                  05  064D    924         rsb
                      064E    925 ;
                      064E    926 ; dest_fltd_prec - calc floating decimal source context
                      064E    927 ;
                      064E    928 dest_fltd_prec:
      0040 8F     B8  064E    929         bispsw  #psl$m_fu               ; enable underflow
     04 53   07   E5  0652    930         bbcc    #7,r3,10$               ; test for grand
        57   02   D0  0656    931         movl    #2,r7                   ; set grand context
                  05  0659    932         rsb
                      065A    933 ;
        07   53   D1  065A    934 10$:    cmpl    r3,#7                   ; float?
             03   14  065D    935         bgtr    20$                     ; if not, br
             57   D4  065F    936         clrl    r7                      ; set F float context
                  05  0661    937         rsb
                      0662    938 ;
        0F   53   D1  0662    939 20$:    cmpl    r3,#15                  ; double?
             04   14  0665    940         bgtr    30$                     ; if not, br
        57   01   D0  0667    941         movl    #1,r7                   ; set double context
                  05  066A    942         rsb
                      066B    943 ;
        57   03   D0  066B    944 30$:    movl    #3,r7                   ; must be huge
                  05  066E    945         rsb
                      066F    946
```

```
                              066F          .sbttl  picpic - picture to picture conversion
                              066F    ; ++
                              066F    ; picpic - picture to picture conversion
                              066F    ;
                              066F    ; functional description:
                              066F    ;
                              066F    ; This routine converts a picture value to a picture value.
                              066F    ;
                              066F    ; inputs:
                              066F    ;
                              066F    ;       r0 = address of the source
                              066F    ;       r1 = size or precision of source
                              066F    ;       r2 = address of the destination
                              066F    ;       r3 = size or the precision of the destination
                              066F    ;
                              066F    ; outputs:
                              066F    ;
                              066F    ;       The destination is filled in
                              066F    ; --
                        C010  066F          .entry  pli$picpic_r6,^m<iv,dv,r4>
                              0671  picpic:
           5E     10      C2  0671          subl    #16,sp                  ; alloc packed temp
                  5E      DD  0674          pushl   sp                      ; addr of target temp
           7E     61      3C  0676          movzwl  pic$w_pq(r1),-(sp)      ; prec & scale of target
                  50      DD  0679          pushl   r0                      ; addr of source
       7E     04  A1      9A  067B          movzbl  pic$b_byte_size(r1),-(sp); prec & scale of src
                  51      DD  067F          pushl   r1                      ; addr of pic node
           54     51      D0  0681          movl    r1,r4                   ; save pic node addr
  00000000'GF     05      FB  0684          calls   #5,g^pli$cvt_fr_pic     ; conv from pic to fix dec
                  52      DD  068B          pushl   r2                      ; final target addr
       7E     04  A3      9A  068D          movzbl  pic$b_byte_size(r3),-(sp); target prec & scale
              08  AE      9F  0691          pushab  8(sp)                   ; addr of fix dec src
           7E     64      3C  0694          movzwl  pic$w_pq(r4),-(sp)      ; src prec & scale
                  53      DD  0697          pushl   r3                      ; pic node addr
  00000000'GF     05      FB  0699          calls   #5,g^pli$cvt_to_pic     ; cvrt fix dec temp to pic
                          04  06A0          ret
```

```
                                  06A1         .sbttl  picfixb - picture to fixed binary conversion
                                  06A1  ; ++
                                  06A1  ; picfixb - picture to fixed binary conversion
                                  06A1  ;
                                  06A1  ; functional description:
                                  06A1  ;
                                  06A1  ; This routine converts a picture vsiue to a fixed binary value.
                                  06A1  ;
                                  06A1  ; inputs:
                                  06A1  ;
                                  06A1  ;      r0 = address of the source
                                  06A1  ;      r1 = size or precision of source
                                  06A1  ;      r2 = address of the destination
                                  06A1  ;      r3 = size or the precision of the destination
                                  06A1  ;
                                  06A1  ; outputs:
                                  06A1  ;
                                  06A1  ;      The destination is filled in
                                  06A1  ; --
                           C030   06A1         .entry  pli$picfixb_r6,^m<iv,dv,r4,r5>
                                  06A3  picfixb:
             5E    10      C2     06A3         subl    #16,sp                   ; alloc packed temp
                   5E      DD     06A6         pushl   sp                       ; addr of target temp
                   1F      DD     06A8         pushl   #31                      ; max precision, 0 scale
                   50      DD     06AA         pushl   r0                       ; src addr
        7E      04 A1      9A     06AC         movzbl  pic$b_byte_size(r1),-(sp); src prec & scale
                   51      DD     06B0         pushl   r1                       ; pic node
     00000000'GF   05      FB     06B2         calls   #5,g^pli$cvt_fr_pic      ; cvrt from pic to fix dec
             50    5E      D0     06B9         movl    sp,r0                    ; reset src to fix dec temp
             51    1F      D0     06BC         movl    #31,r1                   ; reset src size
                 092C      30     06BF         bsbw    cvrt_fixd_fixb           ; go cvrt to fix bin
                           04     06C2         ret
```

```
                        06C3              .sbttl  picfltb - picture to floating binary conversion
                        06C3      ; ++
                        06C3      ; picfltb - picture to floating binary conversion
                        06C3      ;
                        06C3      ; functional description:
                        06C3      ;
                        06C3      ; This routine converts a picture value to a floating binary value.
                        06C3      ;
                        06C3      ; inputs:
                        06C3      ;
                        06C3      ;       r0 = address of the source
                        06C3      ;       r1 = size or precision of source
                        06C3      ;       r2 = address of the destination
                        06C3      ;       r3 = size or the precision of the destination
                        06C3      ;
                        06C3      ; outputs:
                        06C3      ;
                        06C3      ;       The destination is filled in
                        06C3      ; --
                 C0F0   06C3              .entry  pli$picfltb_r6,^m<iv,dv,r4,r5,r6,r7>
                        06C5      oicfltb:
          FF44     30   06C5              bsbw    dest_fltb_prec          ; get dest context
            01     10   06C8              bsbb    cvrt_pic_flt
                   04   06CA              ret
                        06CB      cvrt_pic_flt:
       5E   10     C2   06CB              subl    #16,sp                  ; alloc packed temp
            5E     DD   06CE              pushl   sp                      ; addr of target temp
       54   61     3C   06D0              movzwl  pic$w_pq(r1),r4         ; save src prec & scale
            54     DD   06D3              pushl   r4                      ; use for target prec & scale
            50     DD   06D5              pushl   r0                      ; addr of src
    7E   04 A1     9A   06D7              movzbl  pic$b_byte_size(r1),-(sp); src prec
            51     DD   06DB              pushl   r1                      ; pic node
  00000000'GF 05   FB   06DD              calls   #5,g^pli$cvt_fr_pic     ; conv to fixed dec
       50   5E     D0   06E4              movl    sp,r0                   ; reset src to fix dec temp
       51   54     D0   06E7              movl    r4,r1                   ; reset src prec & scale
          09A9     30   06EA              bsbw    cvrt_fixd_flt           ; go conv to float bin
       5E   10     C0   06ED              addl    #16,sp                  ; clean stack
                   05   06F0              rsb
```

```
                             06F1             .sbttl  picfixd - picture to fixed decimal conversion
                             06F1        ; ++
                             06F1        ; picfixd - picture to fixed decimal conversion
                             06F1        ;
                             06F1        ; functional description:
                             06F1        ;
                             06F1        ; This routine converts a picture value to a fixed decimal value.
                             06F1        ;
                             06F1        ; inputs:
                             06F1        ;
                             06F1        ;       r0 = address of the source
                             06F1        ;       r1 = size or precision of source
                             06F1        ;       r2 = address of the destination
                             06F1        ;       r3 = size or the precision of the destination
                             06F1        ;
                             06F1        ; outputs:
                             06F1        ;
                             06F1        ;       The destination is filled in
                             06F1        ; --
                       C010  06F1             .entry  pli$picfixd_r6,^m<iv,dv,r4>
                             06F3       picfixd:
                    52   DD  06F3             pushl   r2                          ; target addr
                    53   DD  06F5             pushl   r3                          ; target size
                    50   DD  06F7             pushl   r0                          ; src addr
        7E    04 A1   9A  06F9             movzbl  pic$b_byte_size(r1),-(sp); src prec & scale
                    51   DD  06FD             pushl   r1                          ; pic node
   00000000'GF    05   FB  06FF             calls   #5,g^pli$cvt_fr_pic         ; convert pic to fix dec
                         04  0706             ret
```

```
                  0707          .sbttl  picfltd - picture to float decimal conversion
                  0707   ; ++
                  0707   ; picfltd - picture to float decimal conversion
                  0707   ;
                  0707   ; functional description:
                  0707   ;
                  0707   ; This routine converts a picture value to a float decimal value.
                  0707   ;
                  0707   ; inputs:
                  0707   ;
                  0707   ;       r0 = address of the source
                  0707   ;       r1 = size or precision of source
                  0707   ;       r2 = address of the destination
                  0707   ;       r3 = size or the precision of the destination
                  0707   ;
                  0707   ; outputs:
                  0707   ;
                  0707   ;       The destination is filled in
                  0707   ; --
           C0F0   0707          .entry  pli$picfltd_r6,^m<iv,dv,r4,r5,r6,r7>
                  0709 picfltd:
     FF42    30   0709          bsbw    dest_fltd_prec          ; get float context
       BD    10   070C          bsbb    cvrt_pic_flt            ; convert value
             04   070E          ret                             ; done
```

```
                                     070F        .sbttl picchar - picture to character conversion
                                     070F  ; ++
                                     070F  ; picchar - picture to character ccnversion
                                     070F  ;
                                     070F  ; functional description:
                                     070F  ;
                                     070F  ; This routine converts a picture value to a character string.
                                     070F  ;
                                     070F  ; inputs:
                                     070F  ;
                                     070F  ;       r0 = address of the source
                                     070F  ;       r1 = size or precision of source
                                     070F  ;       r2 = address of the destination
                                     070F  ;       r3 = size or the precision of the destination
                                     070F  ;
                                     070F  ; outputs:
                                     070F  ;
                                     070F  ;       The destination is filled in
                                     070F  ; --
                                C030 070F        .entry  pli$picchar_r6,^m<iv,dv,r4,r5>
                                     0711  picchar:
               51    04 A1      9A   0711        mcvzbl  pic$b_byte_size(r1),r1  ; get pic str size
 62    53    20    60    51     2C   0715        movc5   r1,(r0),#32,r3,(r2)     ; copy to char str
                                04   071B        ret
```

```
                              071C          .sbttl  picvcha - picture to character varying conversion
                              071C  ; ++
                              071C  ; picvcha - picture to character varying conversion
                              071C  ;
                              071C  ; functional description:
                              071C  ;
                              071C  ; This routine converts a picture value to a character varying string.
                              071C  ;
                              071C  ; inputs:
                              071C  ;
                              071C  ;           r0 = address of the source
                              071C  ;           r1 = size or precision of source
                              071C  ;           r2 = address of the destination
                              071C  ;           r3 = size or the precision of the destination
                              071C  ;
                              071C  ; outputs:
                              071C  ;
                              071C  ;           The destination is filled in
                              071C  ; --
                        C030  071C          .entry  pli$picvcha_r6,^m<iv,dv,r4,r5>
                              071E  picvcha:
          51    04 A1     9A  071E          movzbl  pic$b_byte_size(r1),r1  ; get pic str size
                62    51  B0  0722          movw    r1,(r2)                 ; put in dest str size
                53    51  B1  0725          cmpw    r1,r3                   ; see if it fits
                      03  1B  0728          blequ   10$                     ; if so, br
                62    53  B0  072A          movw    r3,(r2)                 ; else, use smaller size
                      82  B5  072D  10$:    tstw    (r2)+                   ; point to char str
 62    53    20    60    51   2C  072F          movc5   r1,(r0),#32,r3,(r2)     ; do the move
                          04  0735          ret
                          0736  1166          .sbttl  picbit - picture to bit string conversion
                          0736  1167  ; ++
                          0736  1168  ; picbit - picture to bit string conversion
                          0736  1169  ;
                          0736  1170  ; functional description:
                          0736  1171  ;
                          0736  1172  ; This routine converts a picture value to a bit string.
                          0736  1173  ;
                          0736  1174  ; inputs:
                          0736  1175  ;
                          0736  1176  ;       r0 = address of the source
                          0736  1177  ;       r1 = size or precision of source
                          0736  1178  ;       r2 = address of the destination
                          0736  1179  ;       r3 = size or the precision of the destination
                          0736  1180  ;
                          0736  1181  ; outputs:
                          0736  1182  ;
                          0736  1183  ;       The destination is filled in
                          0736  1184  ; --
                    C030  0736  1185          .entry  pli$picbit_r6,^m<iv,dv,r4,r5>
                          0738  1186  picbit:
          5E   10     C2  0738  1187          subl    #16,sp                  ; alloc packed temp
          7E   61     3C  073B  1188          movzwl  pic$w_pq(r1),-(sp)      ; save the src prec,scale
                          073E  1189                                          ; arg.list
             04 AE    DF  073E  1190          pushal  4(sp)                   ; addr of fix dec target temp
             04 AE    DD  0741  1191          pushl   4(sp)                   ; use same prec,scale for target temp
                 50   DD  0744  1192          pushl   r0                      ; src addr
       7E    04 A1    9A  0746  1193          movzbl  pic$b_byte_size(r1),-(sp); src prec
                 51   DD  074A  1194          pushl   r1                      ; pic node addr
 00000000'GF     05   FB  074C  1195          calls   #5,g^pli$cvt_fr_pic     ; conv to fix dec
                 51 8ED0  0753  1196          popl    r1                      ; get back src prec,scale
          50     5E   D0  0756  1197          movl    sp,r0                   ; set src to fix dec temp
            0B00     31   0759  1198          brw     fixdbit                 ; go conv to bit
```

```
                        075C          .sbttl  picabit - picture to bit aligned conversion
                        075C     ; ++
                        075C     ; picabit - picture to bit aligned conversion
                        075C     ;
                        075C     ; functional description:
                        075C     ;
                        075C     ; This routine converts a picture value to a bit aligned string.
                        075C     ;
                        075C     ; inputs:
                        075C     ;
                        075C     ;        r0 = address of the source
                        075C     ;        r1 = size or precision of source
                        075C     ;        r2 = address of the destination
                        075C     ;        r3 = size or the precision of the destination
                        075C     ;
                        075C     ; outputs:
                        075C     ;
                        075C     ;        The destination is filled in
                        075C     ; --
                  C070  075C          .entry  pli$picabit_r6,^m<iv,dv,r4,r5,r6>
                        075E     picabit:
        5E    10    C2  075E          subl    #16,sp                     ; alloc packed temp
        7E    61    3C  0761          movzwl  pic$w_pq(r1),-(sp)         ; save the src prec,scale
                        0764                                             ; arg.list
           04 AE    DF  0764          pushal  4(sp)                      ; addr of fix dec target temp
           04 AE    DD  0767          pushl   4(sp)                      ; use same prec,scale for target temp
              50    DD  076A          pushl   r0                         ; src addr
     7E    04 A1    9A  076C          movzbl  pic$b_byte_size(r1),-(sp); src prec
              51    DD  0770          pushl   r1                         ; pic node addr
00000000'GF   05    FB  0772          calls   #5,g^pli$cvt_fr_pic        ; conv to fix dec
              51  8ED0  0779          popl    r1                         ; get back src prec,scale
        50    5E    D0  077C          movl    sp,r0                      ; set src to fix dec temp
            07F2    30  077F          bsbw    clr_abit_trailer           ; clear abit last byte
            0AD7    31  0782          brw     fixdbit                    ; go conv to bit
```

```
                          0785  1235          .sbttl  fltbpic - floating to picture conversion
                          0785  1236  ; ++
                          0785  1237  ; fltbpic - floating to picture conversion
                          0785  1238  ;
                          0785  1239  ; functional description:
                          0785  1240  ;
                          0785  1241  ; This routine converts a floating binary value to a picture value.
                          0785  1242  ;
                          0785  1243  ; inputs:
                          0785  1244  ;
                          0785  1245  ;       r0 = address of the source
                          0785  1246  ;       r1 = size or precision of source
                          0785  1247  ;       r2 = address of the destination
                          0785  1248  ;       r3 = size or the precision of the destination
                          0785  1249  ;
                          0785  1250  ; outputs:
                          0785  1251  ;
                          0785  1252  ;       The destination is filled in
                          0785  1253  ; --
                     C1F0 0785  1254          .entry  pli$fltbpic_r6,^m<iv,dv,r4,r5,r6,r7,r8>
                          0787  1255  fltbpic:
          FE61     30     0787  1256          bsbw    src_fltb_prec           ; get src context
            01     10     078A  1257          bsbb    cvrt_flt_pic
                   04     078C  1258          ret
                          078D  1259  cvrt_flt_pic:
       5E   10     C2     078D  1260          subl    #16,sp                  ; alloc packed temp
            52     DD     0790  1261          pushl   r2                      ; make frame for pic cvrt before regs go awa
    7E   04 A3     9A     0792  1262          movzbl  pic$b_byte_size(r3),-(sp); frame target size
    52   08 AE     9E     0796  1263          movab   8(sp),r2                ; reset dest to temp
            52     DD     079A  1264          pushl   r2                      ; push it as pic cvrt src
       7E   63     3C     079C  1265          movzwl  pic$w_pq(r3),-(sp)      ; push target p,q as src p,q
            53     DD     079F  1266          pushl   r3                      ; pic node addr
       53   63     3C     07A1  1267          movzwl  pic$w_pq(r3),r3         ; reset dest size as pic p,q
          0193     30     07A4  1268          bsbw    cvrt_flt_fixd           ; convert flt bin src to fix dec
00000000'GF 05     FB     07A7  1269          calls   #5,g^pli$cvt_to_pic     ; frame all set, cvrt dec to pic
       5E   10     C0     07AE  1270          addl    #16,sp                  ; clean stack
                   05     07B1  1271          rsb
```

```
                          07B2          .sbttl  fltbfixb - floating to fixed binary conversion
                          07B2  ; ++
                          07B2  ; fltbfixb - floating to fixed binary conversion
                          07B2  ;
                          07B2  ; functional description:
                          07B2  ;
                          07B2  ; This routine converts a floating binary value to a fixed binary value.
                          07B2  ;
                          07B2  ; inputs:
                          07B2  ;
                          07B2  ;       r0 = address of the source
                          07B2  ;       r1 = size or precision of source
                          07B2  ;       r2 = address of the destination
                          07B2  ;       r3 = size or the precision of the destination
                          07B2  ;
                          07B2  ; outputs:
                          07B2  ;
                          07B2  ;       The destination is filled in
                          07B2  ; --
                    C010  07B2          .entry  pli$fltbfixb_r6,^m<iv,dv,r4>
                          07B4  fltbfixb:
              FE34    30  07B4          bsbw    src_fltb_prec             ; get src context
                01    10  07B7          bsbb    cvrt_flt_fixb             ; convert floating to fixed
                      04  07B9          ret                               ;
                          07BA  cvrt_flt_fixb:
53    53    02    03  EF  07BA          extzv   #3,#2,r3,r3               ; get dest size
  51    53    F8 8F   78  07BF          ashl    #-8,r3,r1                 ; get dest scale
            51    51  98  07C4          cvtbl   r1,r1                     ; sign extend dest scale
                  5C  12  07C7          bneq    120$                      ; if neq, scale ^= zero
            54    04  84  07C9  5$:     mulb    #4,r4                     ; get source size times 4
            53    54  80  07CC          addb    r4,r3                     ; get case index
                          07CF          case    type=b,r3,<10$,20$,30$,30$,40$,50$,60$,60$,70$,80$,90$,90$,100$,110$
            62    60 6AFD 07EF          cvthl   (r0),(r2)
                      05  07F3          rsb
            62    60  48  07F4  10$:    cvtfb   (r0),(r2)
                      05  07F7          rsb
            62    60  49  07F8  20$:    cvtfw   (r0),(r2)
                      05  07FB          rsb
            62    60  4A  07FC  30$:    cvtfl   (r0),(r2)
                      05  07FF          rsb
            62    60  68  0800  40$:    cvtdb   (r0),(r2)
                      05  0803          rsb
            62    60  69  0804  50$:    cvtdw   (r0),(r2)
                      05  0807          rsb
            62    60  6A  0808  60$:    cvtdl   (r0),(r2)
                      05  080B          rsb
            62    60 48FD 080C  70$:    cvtgb   (r0),(r2)
                      05  0810          rsb
            62    60 49FD 0811  80$:    cvtgw   (r0),(r2)
                      05  0815          rsb
            62    60 4AFD 0816  90$:    cvtgl   (r0),(r2)
                      05  081A          rsb
            62    60 68FD 081B  100$:   cvthb   (r0),(r2)
                      05  081F          rsb
            62    60 69FD 0820  110$:   cvthw   (r0),(r2)
                      05  0824          rsb
                          0825
```

```
                           7E    D4  0825  120$:   clrl    -(sp)                          ; temp for the longword result
                                     0827          case    type=b,r4,<130$,140$,150$>
                           7E    7C  0831          clrq    -(sp)                          ; convert to huge temp
                           7E    D4  0833          clrl    -(sp)                          ;
                      7E   51    D0  0835          movl    r1,-(sp)                       ; set the power of 2 in the exponent
          6E    00004001   8F    C0  0838          addl2   #^x4001,(sp)                   ; add in the constant h_floating part
 8E    10 AE    6E    00   60  74FD  083F          emodh   (r0),#0,(sp),16(sp),(sp)+; adjust to dest scale and convert to integ
                           4B    11  0847          brb     160$                           ; join common code
                           7E    7C  0849  130$:   clrq    -(sp)                          ;
         6E     19    07   51    F0  084B          insv    r1,#7,#25,(sp)                 ; set the power of 2 in the exponent
          6E    00004080   8F    C0  0850          addl2   #^x4080,(sp)                   ; add in the constant d_floating part
                      7E   60    56  0857          cvtfd   (r0),-(sp)                     ; convert to double temp
 8E    08 AE    6E    00   8E    74  085A          emodd   (sp)+,#0,(sp),8(sp),(sp)+; adjust to dest scale and convert to integ
                           31    11  0861          brb     160$                           ; join common code
                           7E    7C  0863  140$:   clrq    -(sp)                          ;
         6E     19    07   51    F0  0865          insv    r1,#7,#25,(sp)                 ; set the power of 2 in the exponent
          6E    00004080   8F    C0  086A          addl2   #^x4080,(sp)                   ; add in the constant d_floating part
 8E    08 AE    6E    00   60    74  0871          emodd   (r0),#0,(sp),8(sp),(sp)+; adjust to dest scale and convert to intege
                           1A    11  0878          brb     160$                           ; join common code
                           7E    7C  087A  150$:   clrq    -(sp)                          ; convert to huge temp
                           7E    D4  087C          clrl    -(sp)                          ;
                      7E   51    D0  087E          movl    r1,-(sp)                       ; set the power of 2 in the exponent
          6E    00004001   8F    C0  0881          addl2   #^x4001,(sp)                   ; add in the constant h_floating part
                      7E   60  56FD  0888          cvtgh   (r0),-(sp)                     ; convert to huge temp
 8E    10 AE    6E    00   8E  74FD  088C          emodh   (sp)+,#0,(sp),16(sp),(sp)+;adjust to dest scale and convert to integ
                                     0894  160$:   case    type=b,r3,<190$,180$,170$>; convert to target context
                      62   8E    D0  089E  170$:   movl    (sp)+,(r2)                     ;
                                 05  08A1          rsb
                      62   8E    F7  08A2  180$:   cvtlw   (sp)+,(r2)                     ;
                                 05  08A5          rsb
                      62   8E    F6  08A6  190$:   cvtlb   (sp)+,(r2)                     ;
                                 05  08A9          rsb
```

```
                    08AA  1363          .sbttl  fltbfltb - floating to floating binary conversion
                    08AA  1364  ; ++
                    08AA  1365  ; fltbfltb - floating to floating binary conversion
                    08AA  1366  ;
                    08AA  1367  ; functional description:
                    08AA  1368  ;
                    08AA  1369  ; This routine converts a floating binary value to a floating binary value.
                    08AA  1370  ;
                    08AA  1371  ; inputs:
                    08AA  1372  ;
                    08AA  1373  ;       r0 = address of the source
                    08AA  1374  ;       r1 = size or precision of source
                    08AA  1375  ;       r2 = address of the destination
                    08AA  1376  ;       r3 = size or the precision of the destination
                    08AA  1377  ;
                    08AA  1378  ; outputs:
                    08AA  1379  ;
                    08AA  1380  ;       The destination is filled in
                    08AA  1381  ; --
              C090  08AA  1382          .entry  pli$fltbfltb_r6,^m<iv,dv,r4,r7>
                    08AC  1383  fltbfltb:
        FD3C    30  08AC  1384          bsbw    src_fltb_prec           ; calc floating context for source
        FD5A    30  08AF  1385          bsbw    dest_fltb_prec          ; calc destination context
          01    10  08B2  1386          bsbb    cvrt_flt_flt
                04  08B4  1387          ret
                    08B5  1388  cvrt_flt_flt:
   57     04    84  08B5  1389          mulb2   #4,r7                   ; calculate index for case
   54     57    80  08B8  1390          addb    r7,r4
                    08BB  1391          case    type=b,r4,<10$,20$,30$,40$,50$,60$,70$,80$,90$,100$, -
                    08BB  1392                          110$,120$,130$,140$,150$>
   62     60  70FD  08DD  1393          movh    (r0),(r2)
                05  08E1  1394          rsb
   62     60    50  08E2  1395  10$:    movf    (r0),(r2)
                05  08E5  1396          rsb
   62     60    76  08E6  1397  20$:    cvtdf   (r0),(r2)
                05  08E9  1398          rsb
   62     60  33FD  08EA  1399  30$:    cvtgf   (r0),(r2)
                05  08EE  1400          rsb
   62     60  F6FD  08EF  1401  40$:    cvthf   (r0),(r2)
                05  08F3  1402          rsb
   62     60    56  08F4  1403  50$:    cvtfd   (r0),(r2)
                05  08F7  1404          rsb
   62     60    70  08F8  1405  60$:    movd    (r0),(r2)
                05  08FB  1406          rsb
   7E     60  56FD  08FC  1407  70$:    cvtgh   (r0),-(sp)
   62     8E  F7FD  0900  1408          cvthd   (sp)+,(r2)
                05  0904  1409          rsb
   62     60  F7FD  0905  1410  80$:    cvthd   (r0),(r2)
                05  0909  1411          rsb
   62     60  99FD  090A  1412  90$:    cvtfg   (r0),(r2)
                05  090E  1413          rsb
   7E     60  32FD  090F  1414  100$:   cvtdh   (r0),-(sp)
   62     8E  76FD  0913  1415          cvthg   (sp)+,(r2)
                05  0917  1416          rsb
   62     60  50FD  0918  1417  110$:   movg    (r0),(r2)
                05  091C  1418          rsb
   62     60  76FD  091D  1419  120$:   cvthg   (r0),(r2)
```

```
          05  0921  1420          rsb
62   60 98FD  0922  1421 130$:    cvtfh   (r0),(r2)
          05  0926  1422          rsb
62   60 32FD  0927  1423 140$:    cvtdh   (r0),(r2)
          05  092B  1424          rsb
62   60 56FD  092C  1425 150$:    cvtgh   (r0),(r2)
          05  0930  1426          rsb
```

```
                               0931  1428         .sbttl  fltbfixd - floating to fixed decimal conversion
                               0931  1429  ; ++
                               0931  1430  ; fltbfixd - floating to fixed decimal conversion
                               0931  1431  ;
                               0931  1432  ; functional description:
                               0931  1433  ;
                               0931  1434  ; This routine converts a floating binary value to a fixed decimal value.
                               0931  1435  ;
                               0931  1436  ; inputs:
                               0931  1437  ;
                               0931  1438  ;       r0 = address of the source
                               0931  1439  ;       r1 = size or precision of source
                               0931  1440  ;       r2 = address of the destination
                               0931  1441  ;       r3 = size or the precision of the destination
                               0931  1442  ;
                               0931  1443  ; outputs:
                               0931  1444  ;
                               0931  1445  ;       The destination is filled in
                               0931  1446  ; --
                         C1F0  0931  1447         .entry  pli$fltbfixd_r6,^m<iv,dv,r4,r5,r6,r7,r8>
                               0933  1448 fltbfixd:
                  FCB5    30   0933  1449         bsbw    src_fltb_prec           ;get src context
                  0001    30   0936  1450         bsbw    cvrt_flt_fixd
                          04   0939  1451         ret
                               093A  1452 cvrt_flt_fixd:
     5E   00000070 8F     C2   093A  1453         subl    #112,sp                 ; alloc local storage
                    54    D5   0941  1454         tstl    r4                      ; see if src is single floating
                    08    12   0943  1455         bneq    10$                     ; if not F, br
             68 AE  60    56   0945  1456         cvtfd   (r0),104(sp)            ; cvrt float to double
             50  68 AE    DE   0949  1457         moval   104(sp),r0              ; reset source
             51  34 AE    9E   094D  1458 10$:    movab   52(sp),r1               ; setup math call frame
          EC A1  38 AE    9E   0951  1459         movab   56(sp),-20(r1)          ; string_addr
             F0 A1  53    9A   0956  1460         movzbl  r3,-16(r1)              ; sig_digits
  F4 A1   01    18  01    F0   095A  1461         insv    #1,#24,#1,-12(r1);      ; flags (truncate)
                 E0 A1    D4   0960  1462         clrl    -32(r1)                 ; clr rt_round (no right rounding)
              7E    52    7D   0963  1463         movq    r2,-(sp)                ; save dest. regs
                               0966  1464         case    type=b,r4,<20$,20$,40$>
                    20    B9   0970  1465         bicpsw  #psl$m_iv               ; turn off iv
            00000000'GF   16   0972  1466         jsb     g^ots$$cvt_h_t_r8
                    20    B8   0978  1467         bispsw  #psl$m_iv
                    12    11   097A  1468         brb     50$
            00000000'GF   16   097C  1469 20$:    jsb     g^ots$$cvt_d_t_r8
                    0A    11   0982  1470         brb     50$
                    20    B9   0984  1471 40$:    bicpsw  #psl$m_iv
            00000000'GF   16   0986  1472         jsb     g^ots$$cvt_g_t_r8
                    20    B8   098C  1473         bispsw  #psl$m_iv
                               098E  1474 ;
              56    8E    7D   098E  1475 50$:    movq    (sp)+,r6                ; rest dest to r6,r7
    58    EC A1  E0 A1    C1   0991  1476         addl3   -32(r1),-20(r1),r8      ; add offset to get start of digit str
                 E8 A1    D5   0997  1477         tstl    -24(r1)                 ; test sign returned by cvt
                    0C    14   099A  1478         bgtr    70$                     ;  and put appropriate sign char
                    05    19   099C  1479         blss    60$                     ;  in front of digit string to make
              78    20    90   099E  1480         movb    #^a/ /,-(r8)            ;  a proper leading separate string
                    08    11   09A1  1481         brb     80$
              78    2D    90   09A3  1482 60$:    movb    #^a/-/,-(r8)
                    03    11   09A6  1483         brb     80$
              78    2B    90   09A8  1484 70$:    movb    #^a/+/,-(r8)
```

```
                55    57    9A   09AB   1485 80$:      movzbl  r7,r5                        ; get prec of dest
          55    E4    A1    D1   09AE   1486           cmpl    -28(r1),r5                   ; see if gtr exponent
                      11    15   09B2   1487           bleq    90$                          ; [note that ashp with count > destin.
                                 09B4   1488                                                ; length will not overflow on 11/780 ]
          00000000'8F       DD   09B4   1489           pushl   #ss$_decovf                  ; signal decimal overflow
                      50    7C   09BA   1490           clrq    r0
      00000000'GF     01    FB   09BC   1491           calls   #1,g^lib$signal
                      30    11   09C3   1492           brb     100$
     57    57    F8   8F    78   09C5   1493 90$:      ashl    #-8,r7,r7                    ; get scale
                57    55    C2   09CA   1494           subl2   r5,r7                        ; scale-prec
          57    E4    A1    C0   09CD   1495           addl2   -28(r1),r7                   ; (scale-prec)+exponent
                      0E    18   09D1   1496           bgeq    93$                          ; if positive scale factor
      FFFFFFE1 8F     57    D1   09D3   1497           cmpl    r7,#-31                      ; shift factor >max size of packed?
                      0D    18   09DA   1498           bgeq    97$                          ; if so,
                57    1F    8E   09DC   1499           mnegb   #31,r7                       ; use max
                      08    11   09DF   1500           brb     97$
                1F    57    D1   09E1   1501 93$:      cmpl    r7,#31                       ; shift factor >max size of packed?
                      03    15   09E4   1502           bleq    97$                          ; if so,
                57    1F    90   09E6   1503           movb    #31,r7                       ; use max
          6E    55    68    55   09    09E9   1504 97$:      cvtsp   r5,(r8),r5,(sp)              ; convert lead sep to packed temp
66    55    00    6E    55    57    F8   09EE   1505           ashp    r7,r5,(sp),#0,r5,(r6)        ; adjust result to scale
          5E    00000070 8F     C0   09F5   1506 100$:     addl    #112,sp                      ; clean stack
                      05         09FC   1507           rsb                                  ;return
```

```
                  09FD          .sbttl  fltbfltd - float binary to float decimal conversion
                  09FD    ; ++
                  09FD    ; fltbfltd - float binary to float decimal conversion
                  09FD    ;
                  09FD    ; functional description:
                  09FD    ;
                  09FD    ; This routine converts a float binary value to a float decimal value.
                  09FD    ;
                  09FD    ; inputs:
                  09FD    ;
                  09FD    ;       r0 = address of the source
                  09FD    ;       r1 = size or precision of source
                  09FD    ;       r2 = address of the destination
                  09FD    ;       r3 = size or the precision of the destination
                  09FD    ;
                  09FD    ; outputs:
                  09FD    ;
                  09FD    ;       The destination is filled in
                  09FD    ; --
            C090  09FD          .entry  pli$fltbfltd_r6,^m<iv,dv,r4,r7>
                  09FF  fltbfltd:
       FBE9   30  09FF          bsbw    src_fltb_prec           ; get src context
       FC49   30  0A02          bsbw    dest_fltd_prec          ; get dest context
       FEAD   30  0A05          bsbw    cvrt_flt_flt            ; continue in common
              04  0A08          ret
```

```
                              0A09        .sbttl fltbchar - floating to character conversion
                              0A09  ; ++
                              0A09  ; fltbchar - floating to character conversion
                              0A09  ;
                              0A09  ; functional description:
                              0A09  ;
                              0A09  ; This routine converts a floating binary value to a character string.
                              0A09  ;
                              0A09  ; inputs:
                              0A09  ;
                              0A09  ;       r0 = address of the source
                              0A09  ;       r1 = size or precision of source
                              0A09  ;       r2 = address of the destination
                              0A09  ;       r3 = size or the precision of the destination
                              0A09  ;
                              0A09  ; outputs:
                              0A09  ;
                              0A09  ;       r0 = size of actual data string
                              0A09  ;
                              0A09  ;       The destination is filled in
                              0A09  ; --
                        CFF0  0A09        .entry  pli$fltbchar_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                              0A0B fltbchar:
             FBDD    30       0A0B        bsbw    src_fltb_prec           ; get src context
51    00000064 8F    C4       0A0E        mull2   #100,r1                 ; conv to decimal digit prec
51    0000014B 8F    C0       0A15        addl    #331,r1                 ; this will assure the ceil
51    0000014C 8F    C6       0A1C        divl    #332,r1
          22   51    D1       0A23        cmpl    r1,#34                  ; can't be greater than max dec prec
               03    15       0A26        bleq    6$
       51      22    D0       0A28        movl    #34,r1                  ; set max
               01    10       0A2B 6$:    bsbb    cvrt_flt_char           ; convert number
                     04       0A2D        ret
                              0A2E cvrt_flt_char:
5E    00000084 8F    C2       0A2E        subl2   #132,sp                 ; alloc local storage
               54    D5       0A35        tstl    r4                      ; check for single floating
               0A    12       0A37        bneq    10$                     ; br if not F
    80 AE      60    56       0A39        cvtfd   (r0),-128(sp)           ; conv to D
    50   80 AE       DE       0A3D        moval   -128(sp),r0             ; reset src addr
               54    D6       0A41        incl    r4                      ; reset context to D
                              0A43  ;
               51    DD       0A43 10$:   pushl   r1                      ; save src prec
    51   50 AE       DE       0A45        moval   80(sp),r1               ; setup math call frame (size=40 bytes)
    EC A1      51    D0       0A49        movl    r1,-20(r1)              ; string_addr
          F0 A1    8ED0       0A4D        popl    -16(r1)                 ; sig_digits
          F4 A1      D4       0A51        clrl    -12(r1)                 ; caller flags (default round)
          DC A1      D4       0A54        clrl    -36(r1)                 ; clr rt_round
       59      52    7D       0A57        movq    r2,r9                   ; save r2,r3,r4
       5B      54    D0       0A5A        movl    r4,r11
                              0A5D        case    type=b,r4,<21$,22$,23$>
               20    B9       0A67        bicpsw  #psl$m_iv               ; turn off iv
      00000000'GF    16       0A69        jsb     g^ots$$cvt_h_t_r8
               20    B8       0A6F        bispsw  #psl$m_iv
               1A    11       0A71        brb     25$
      00000000'GF    16       0A73 21$:   jsb     g^ots$$cvt_d_t_r8
               12    11       0A79        brb     25$
      00000000'GF    16       0A7B 22$:   jsb     g^ots$$cvt_d_t_r8
               0A    11       0A81        brb     25$
```

```
                                    0A83   1592 23$:
                      20     B9     0A83   1593          bicpsw  #psl$m_iv
              00000000'GF    16     0A85   1594          jsb     g^ots$$cvt_g_t_r8
                      20     B8     0A8B   1595          bispsw  #psl$m_iv
                                    0A8D   1596 ;
                 58   5E     D0     0A8D   1597 25$:     movl    sp,r8                      ; addr of temp for char str
                   E8 A1     D5     0A90   1598          tstl    -24(r1)                    ; tst sign
                      05     18     0A93   1599          bgeq    30$                        ; br if plus or zero
                 88   2D     90     0A95   1600          movb    #^a/-/,(r8)+               ; else put - sign in chr str
                      03     11     0A98   1601          brb     40$
                 88   20     90     0A9A   1602 30$:     movb    #^a/ /,(r8)+               ; put in blank for pos or 0
      52    EC A1    E0 A1   C1     0A9D   1603 40$:     addl3   -32(r1),-20(r1),r2         ; add offset to str. addr to get 1st digit
                 88   82     90     0AA3   1604          movb    (r2)+,(r8)+                ; copy most sig digit
                 88   2E     90     0AA6   1605          movb    #^a/./,(r8)+               ; put in dec. pt.
          56    F0 A1 01     C3     0AA9   1606          subl3   #1,-16(r1),r6              ; get length of remaining digits
                 57   51     D0     0AAE   1607          movl    r1,r7                      ; copy call frame ptr
            68   62   56     28     0AB1   1608          movc3   r6,(r2),(r8)               ; copy remaining frac digits
                 58   53     D0     0AB5   1609          movl    r3,r8                      ; point r8 past end of dest string
            88     45 8F     90     0AB8   1610          movb    #^a/E/,(r8)+               ; put in the "E"
                      5B     D6     0ABC   1611          incl    r11                        ; get correct exponent digit size
                   E8 A7     D5     0ABE   1612          tstl    -24(r7)                    ; test sign for zero
                      03     13     0AC1   1613          beql    45$                        ; if 0, do not decr exponent
                   E4 A7     D7     0AC3   1614          decl    -28(r7)                    ; adjust exponent for dec pt.
      80 AE    5B    E4 A7   F9     0AC6   1615 45$:     cvtlp   -28(r7),r11,-128(sp)       ; cvt exp to packed
   68    5B    80 AE     5B  08     0ACC   1616          cvtps   r11,-128(sp),r11,(r8)      ; cvt packed to leading sep
                 56   05     C0     0AD2   1617          addl2   #5,r6                      ; get length of frac+extra chars
                 56   5B     C0     0AD5   1618          addl2   r11,r6                     ; add in exp size
   69    5A    20   6E   56  2C     0AD8   1619          movc5   r6,(sp),#32,r10,(r9)       ; copy temp char str to dest
                 50   56     D0     0ADE   1620          movl    r6,r0                      ; return dest length
        5E    00000084 8F    C0     0AE1   1621          addl2   #132,sp                    ; clean stack
                             05     0AE8   1622          rsb
                                    0AE9   1623 ;
```

```
                              0AE9             .sbttl  fltbvcha - floating to character varying conversion
                              0AE9      ; ++
                              0AE9      ; fltbvcha - floating to character varying conversion
                              0AE9      ;
                              0AE9      ; functional description:
                              0AE9      ;
                              0AE9      ; This routine converts a floating binary value to a character varying string.
                              0AE9      ;
                              0AE9      ; inputs:
                              0AE9      ;
                              0AE9      ;       r0 = address of the source
                              0AE9      ;       r1 = size or precision of source
                              0AE9      ;       r2 = address of the destination
                              0AE9      ;       r3 = size or the precision of the destination
                              0AE9      ;
                              0AE9      ; outputs:
                              0AE9      ;
                              0AE9      ;       The destination is filled in
                              0AE9      ; --
                         CFF0 0AE9             .entry  pli$fltbvcha_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                              0AEB      fltbvcha:
                   82     3F  0AEB             pushaw  (r2)+                   ; skip current length
                 FAFB     30  0AED             bsbw    src_fltb_prec           ; get src context
     51  00000064 8F     C4  0AF0             mull2   #100,r1                 ; conv to decimal digit prec
     51  0000014B 8F     C0  0AF7             addl2   #331,r1
     51  0000014C 8F     C6  0AFE             divl    #332,r1
              22     51  D1  0B05             cmpl    r1,#34                  ; can't be gtr than max dec prec
                     03  15  0B08             bleq    6$
              51     22  D0  0B0A             movl    #34,r1                  ;
                 FF1E     30  0B0D      6$:    bsbw    cvrt_flt_char           ; convert to char
              9E     50  B0  0B10             movw    r0,@(sp)+               ; store length
                         04  0B13             ret                             ; return
```

```
                      0B14          .sbttl  floating to bit conversion
                      0B14     ; ++
                      0B14     ; fltbbit - floating to bit string conversion
                      0B14     ; fltbabit - floating to bit aligned conversion
                      0B14     ;
                      0B14     ; functional description:
                      0B14     ;
                      0B14     ; This routine converts a floating binary value to a bit aligned string.
                      0B14     ;
                      0B14     ; inputs:
                      0B14     ;
                      0B14     ;       r0 = address of the source
                      0B14     ;       r1 = size or precision of source
                      0B14     ;       r2 = address of the destination
                      0B14     ;       r3 = size or the precision of the destination
                      0B14     ;       r6 = bit offset to destination
                      0B14     ;
                      0B14     ; outputs:
                      0B14     ;
                      0B14     ;       The destination is filled in
                      0B14     ; --
                C070  0B14          .entry  pli$fltbabit_r6,^m<iv,dv,r4,r5,r6>
                      0B16     fltbabit:
          045B    30  0B16          bsbw    clr_abit_trailer        ; clear abit last byte
            02    11  0B19          brb     fltbbit                 ;
                C030  0B1B          .entry  pli$fltbbit_r6,^m<iv,dv,r4,r5>
                      0B1D     fltbbit:
          FACB    30  0B1D          bsbw    src_fltb_prec           ; get src context
            7E    D4  0B20          clrl    -(sp)                   ; get temp space
      1F    51    D1  0B22          cmpl    r1,#31                  ; see if gtr max binary prec
            03    15  0B25          bleq    10$                     ; if lss 31, ok
      51    1F    D0  0B27          movl    #31,r1                  ; else plug in max
                      0B2A     10$:
       004E 8F    BB  0B2A          pushr   #^m<r1,r2,r3,r6>        ; save destination
    52   10 AE    DE  0B2E          moval   16(sp),r2               ; plug address of temp for dest
      53    51    D0  0B32          movl    r1,r3                   ; plug precision
          FC82    30  0B35          bsbw    cvrt_flt_fixb           ; convert source to fixb temp
       004E 8F    BA  0B38          popr    #^m<r1,r2,r3,r6>        ; restore destination
      50    5E    D0  0B3C          movl    sp,r0                   ; plug address of temp for source
          03D2    30  0B3F          bsbw    cvrt_fixb_bit           ; convert temp to bit
                  04  0B42          ret
```

```
                                  0B43          .sbttl  fixbpic - fixed binary to picture conversion
                                  0B43   ; ++
                                  0B43   ; fixbpic - fixed binary to picture conversion
                                  0B43   ;
                                  0B43   ; functional description:
                                  0B43   ;
                                  0B43   ; This routine converts a fixed binary value to a picture value.
                                  0B43   ;
                                  0B43   ; inputs:
                                  0B43   ;
                                  0B43   ;      r0 = address of the source
                                  0B43   ;      r1 = size or precision of source
                                  0B43   ;      r2 = address of the destination
                                  0B43   ;      r3 = size or the precision of the destination
                                  0B43   ;
                                  0B43   ; outputs:
                                  0B43   ;
                                  0B43   ;      The destination is filled in
                                  0B43   ; --
                            C030  0B43          .entry  pli$fixbpic_r6,^m<iv,dv,r4,r5>
                                  0B45   fixbpic:
               5E     10     C2   0B45          subl    #16,sp                        ; alloc packed temp
                      52     DD   0B48          pushl   r2                            ; make frame for pic cvrt before regs go awa
          7E    04    A3     9A   0B4A          movzbl  pic$b_byte_size(r3),-(sp); frame target size
          52    08    AE     9E   0B4E          movab   8(sp),r2                      ; reset dest to temp
                      52     DD   0B52          pushl   r2                            ; push it as pic cvrt src
               7E     63     3C   0B54          movzwl  pic$w_pq(r3),-(sp)            ; push target p,q as src p,q
                      53     DD   0B57          pushl   r3                            ; pic node addr
               53     63     3C   0B59          movzwl  pic$w_pq(r3),r3               ; reset dest size as pic p,q
                    01B8     30   0B5C          bsbw    cvrt_fixb_fixd                ; conv fixb src to fix dec
     00000000'GF      05     FB   0B5F          calls   #5,g^pli$cvt_to_pic           ; frame all set, cvrt dec to pic
                             04   0B66          ret
```

```
                               0B67          .sbttl  fixbfixb - fixed binary to fixed binary conversion
                               0B67  ; ++
                               0B67  ; fixbfixb - fixed binary to fixed binary conversion
                               0B67  ;
                               0B67  ; functional description:
                               0B67  ;
                               0B67  ; This routine converts fixed binary values to fixed binary values of a different
                               0B67  ; precision.
                               0B67  ;
                               0B67  ; inputs:
                               0B67  ;
                               CB67  ;       r0 = address of the source
                               0B67  ;       r1 = size or precision of source
                               0B67  ;       r2 = address of the destination
                               0B67  ;       r3 = size or the precision of the destination
                               0B67  ;
                               0B67  ; outputs:
                               0B67  ;
                               0B67  ;       The destination is filled in
                               0B67  ; --
                         C030  0B67          .entry  pli$fixbfixb_r6,^m<iv,dv,r4,r5>
                               0B69  fixbfixb:
                   01      10  0B69          bsbb    cvrt_fixb_fixb          ; store the result
                           04  0B6B          ret                             ;
                               0B6C  ;
                               0B6C  ; subroutine to store a fixed binary value
                               0B6C  ;
                               0B6C  cvrt_fixb_fixb:                         ; store fixed binary result
    54    51    F8 8F      78  0B6C          ashl    #-8,r1,r4               ; get source scale
 51    51    02      03    EF  0B71          extzv   #3,#2,r1,r1             ; get valid contexts
    55    53    F8 8F      78  0B76          ashl    #-8,r3,r5               ; get dest scale
 53    53    02      03    EF  0B7B          extzv   #3,#2,r3,r3             ; get valid contexts
             55      54    82  0B80          subb2   r4,r5                   ; calc dest scale - source scale
                     4A    12  0B83          bneq    90$                     ; if neq different scales
             51      04    84  0B85          mulb    #4,r1                   ;
             53      51    80  0B88          addb    r1,r3
                               0B8B          case    type=b,r3,<10$,20$,30$,30$,40$,50$,60$,60$,70$,80$,5$,5$,70$,80$>
                               0BAB  5$:
             62      60    D0  0BAB          movl    (r0),(r2)               ;
                           05  0BAE          rsb
                               0BAF
             62      60    90  0BAF  10$:    movb    (r0),(r2)               ;
                           05  0BB2          rsb
             62      60    99  0BB3  20$:    cvtbw   (r0),(r2)               ;
                           05  0BB6          rsb
             62      60    98  0BB7  30$:    cvtbl   (r0),(r2)               ;
                           05  0BBA          rsb
             62      60    33  0BBB  40$:    cvtwb   (r0),(r2)               ;
                           05  0BBE          rsb
             62      60    B0  0BBF  50$:    movw    (r0),(r2)               ;
                           05  0BC2          rsb
             62      60    32  0BC3  60$:    cvtwl   (r0),(r2)               ;
                           05  0BC6          rsb
             62      60    F6  0BC7  70$:    cvtlb   (r0),(r2)               ;
                           05  0BCA          rsb
             62      60    F7  0BCB  80$:    cvtlw   (r0),(r2)               ;
                           05  0BCE          rsb
```

```
                          0BCF   1790
                          0BCF   1791 90$:      case      type=b,r1,<120$,110$,100$>
                          0BD9   1792
         50    60    D0   0BD9   1793 100$:     movl      (r0),r0                       ; get source in longword
               08    11   0BDC   1794           brb       130$                          ;
         50    60    32   0BDE   1795 110$:     cvtwl     (r0),r0                       ;
               03    11   0BE1   1796           brb       130$                          ;
         50    60    98   0BE3   1797 120$:     cvtbl     (r0),r0                       ;
         55    55    98   0BE6   1798 130$:     cvtbl     r5,r5                         ; sign extend dest scale - source scale
               06    19   0BE9   1799           blss      131$                          ; branch if shift right
   51    50    55    78   0BEB   1800           ashl      r5,r0,r1                      ; convert to dest scale
               14    11   0BEF   1801           brb       135$                          ; join common code
         55    55    CE   0BF1   1802 131$:     mnegl     r5,r5                         ; make positive
         1F    55    D1   0BF4   1803           cmpl      r5,#31                        ; trying to shift away all the bits?
               04    1F   0BF7   1804           blssu     132$                          ; if lssu then no
               51    D4   0BF9   1805           clrl      r1                            ; else result is zero
               08    11   0BFB   1806           brb       135$                          ; go move to dest
   55    01    55    78   0BFD   1807 132$:     ashl      r5,#1,r5                      ; calc 2**(abs(dest scale))
   51    50    55    C7   0C01   1808           divl3     r5,r0,r1                      ; convert to dest scale
                          0C05   1809 135$:     case      type=b,r3,<160$,150$,140$>
         62    51    D0   0C0F   1810 140$:     movl      r1,(r2)                       ; put back to dest context
               05         0C12   1811           rsb                                     ;
         62    51    F7   0C13   1812 150$:     cvtlw     r1,(r2)                       ;
               05         0C16   1813           rsb                                     ;
         62    51    F6   0C17   1814 160$:     cvtlb     r1,(r2)                       ;
               05         0C1A   1815           rsb                                     ;
```

```
                               0C1B          .sbttl  fixbfltb - fixed binary to floating binary conversion
                               0C1B   ; ++
                               0C1B   ; fixbfltb - fixed binary to floating binary conversion
                               0C1B   ;
                               0C1B   ; functional description:
                               0C1B   ;
                               0C1B   ; This routine converts fixed binary values to floating binary values of a different
                               0C1B   ; precision.
                               0C1B   ;
                               0C1B   ; inputs:
                               0C1B   ;
                               0C1B   ;      r0 = address of the source
                               0C1B   ;      r1 = size or precision of source
                               0C1B   ;      r2 = address of the destination
                               0C1B   ;      r3 = size or the precision of the destination
                               0C1B   ;
                               0C1B   ; outputs:
                               0C1B   ;
                               0C1B   ;      The destination is filled in
                               0C1B   ; --
                        C090   0C1B          .entry  pli$fixbfltb_r6,^m<iv,dv,r4,r7>
                               0C1D  fixbfltb:
                   F9EC   30   0C1D          bsbw    dest_fltb_prec          ; get destination floating context
                     01   10   0C20          bsbb    cvrt_fixb_flt
                          04   0C22          ret
                               0C23  cvrt_fixb_flt:
       54    51    F8 8F  78   0C23          ashl    #-8,r1,r4               ; get source scale
 51    51    02    03     EF   0C28          extzv   #3,#2,r1,r1             ; determine size of source
             54    54     98   0C2D          cvtbl   r4,r4                   ; non-zero scale?
                   5C     12   0C30          bneq    120$                    ; if neq, yes
             57    04     84   0C32          mulb    #4,r7                   ; calculate index for case
             57    51     80   0C35          addb    r1,r7                   ;
                               0C38          case    type=b,r7,<10$,20$,30$,30$,40$,50$,60$,60$,70$,80$,90$,90$,100$,110$
             62    60 6EFD     0C58          cvtlh   (r0),(r2)
                          05   0C5C          rsb
             62    60     4C   0C5D  10$:    cvtbf   (r0),(r2)
                          05   0C60          rsb
             62    60     4D   0C61  20$:    cvtwf   (r0),(r2)
                          05   0C64          rsb
             62    60     4E   0C65  30$:    cvtlf   (r0),(r2)
                          05   0C68          rsb
             62    60     6C   0C69  40$:    cvtbd   (r0),(r2)
                          05   0C6C          rsb
             62    60     6D   0C6D  50$:    cvtwd   (r0),(r2)
                          05   0C70          rsb
             62    60     6E   0C71  60$:    cvtld   (r0),(r2)
                          05   0C74          rsb
             62    60 4CFD     0C75  70$:    cvtbg   (r0),(r2)
                          05   0C79          rsb
             62    60 4DFD     0C7A  80$:    cvtwg   (r0),(r2)
                          05   0C7E          rsb
             62    60 4EFD     0C7F  90$:    cvtlg   (r0),(r2)
                          05   0C83          rsb
             62    60 6CFD     0C84  100$:   cvtbh   (r0),(r2)
                          05   0C88          rsb
             62    60 6DFD     0C89  110$:   cvtwh   (r0),(r2)
                          05   0C8D          rsb
```

```
                                0C8E   1874
                                0C8E   1875 120$:    case    type=b,r1,<150$,140$,130$>; convert source to long context
                                0C98   1876
             50    60      D0   0C98   1877 130$:    movl    (r0),r0                  ;
                   08      11   0C9B   1878          brb     160$                     ;
             50    60      32   0C9D   1879 140$:    cvtwl   (r0),r0                  ;
                   03      11   0CA0   1880          brb     160$                     ;
             50    60      98   0CA2   1881 150$:    cvtbl   (r0),r0                  ;
             54    54      CE   0CA5   1882 160$:    mnegl   r4,r4                    ;negate scale factor
                                0CA8   1883          case    type=b,r7,<170$,180$,190$>; case on dest type
             7E    50 6EFD      0CB2   1884          cvtlh   r0,-(sp)                 ; convert to huge temp
                   7E      7C   0CB6   1885          clrq    -(sp)                    ; convert to huge temp
                   7E      D4   0CB8   1886          clrl    -(sp)                    ;
             7E    54      D0   0CBA   1887          movl    r4,-(sp)                 ; set the power of 2 in the exponent
    6E     00004001 8F     C0   0CBD   1888          addl2   #^x4001,(sp)             ; add in the constant h_floating part
       62     8E   8E 65FD      0CC4   1889          mulh3   (sp)+,(sp)+,(r2)         ; adjust result for scale
                           05   0CC9   1890          rsb
             7E    50      6E   0CCA   1891 170$:    cvtld   r0,-(sp)                 ; convert to double temp
                   7E      7C   0CCD   1892          clrq    -(sp)                    ;
    6E     19      07  54  F0   0CCF   1893          insv    r4,#7,#25,(sp)           ; set the power of 2 in the exponent
    6E     00004080 8F     C0   0CD4   1894          addl2   #^x4080,(sp)             ; add in the constant h_floating part
             6E    8E      64   0CDB   1895          muld2   (sp)+,(sp)               ; adjust for scale
             62    8E      76   0CDE   1896          cvtdf   (sp)+,(r2)               ; convert to float result
                           05   0CE1   1897          rsb
             7E    50      6E   0CE2   1898 180$:    cvtld   r0,-(sp)                 ; convert to double temp
                   7E      7C   0CE5   1899          clrq    -(sp)                    ;
    6E     19      07  54  F0   0CE7   1900          insv    r4,#7,#25,(sp)           ; set the power of 2 in the exponent
    6E     00004080 8F     C0   0CEC   1901          addl2   #^x4080,(sp)             ; add in the constant h_floating part
       62     8E   8E      65   0CF3   1902          muld3   (sp)+,(sp)+,(r2)         ; adjust result for scale
                           05   0CF7   1903          rsb
             7E    50 6EFD      0CF8   1904 190$:    cvtlh   r0,-(sp)                 ; convert to huge temp
                   7E      7C   0CFC   1905          clrq    -(sp)                    ; convert to huge temp
                   7E      D4   0CFE   1906          clrl    -(sp)                    ;
             7E    54      D0   0D00   1907          movl    r4,-(sp)                 ; set the power of 2 in the exponent
    6E     00004001 8F     C0   0D03   1908          addl2   #^x4001,(sp)             ; add in the constant h_floating part
             6E    8E      64   0D0A   1909          muld2   (sp)+,(sp)               ; adjust for scale
             62    8E    76FD   0D0D   1910          cvthg   (sp)+,(r2)               ; convert to grand result
                           05   0D11   1911          rsb
```

```
                          0D12         .sbttl  fixbfixd - fixed binary to fixed decimal conversion
                          0D12  ; ++
                          0D12  ; fixbfixd - fixed binary to fixed decimal conversion
                          0D12  ;
                          0D12  ; functional description:
                          0D12  ;
                          0D12  ; This routine converts fixed binary values to fixed decimal values of a different
                          0D12  ; precision.
                          0D12  ;
                          0D12  ; inputs:
                          0D12  ;
                          0D12  ;      r0 = address of the source
                          0D12  ;      r1 = size or precision of source
                          0D12  ;      r2 = address of the destination
                          0D12  ;      r3 = size or the precision for the destination
                          0D12  ;
                          0D12  ; outputs:
                          0D12  ;
                          0D12  ;      The destination is filled in
                          0D12  ; --
                    C030  0D12         .entry  pli$fixbfixd_r6,^m<iv,dv,r4,r5>
                          0D14  fixbfixd:
               01     10  0D14         bsbb    cvrt_fixb_fixd
                      04  0D16         ret
                          0D17  ;
                          0D17  ; convert fixed binary to fixed decimal
                          0D17  ;
                          0D17  cvrt_fixb_fixd:
      55   51   F8 8F     78  0D17         ashl    #-8,r1,r5                 ; get the source scale factor
           55   55        98  0D1C         cvtbl   r5,r5                     ; sign extend byte value scale to long
           51   51        9A  0D1F         movzbl  r1,r1                     ; zero extend precision
              0080        30  0D22         bsbw    get_src_fixprec           ; calc number of target digits
      54   53   F8 8F     78  0D25         ashl    #-8,r3,r4                 ; get scale factor of dest
           53   53        9A  0D2A         movzbl  r3,r3                     ; zero extend precision
           54   54        98  0D2D         cvtbl   r4,r4                     ; sign extend scale
                19        12  0D30         bneq    10$                       ; if neq, no zero scale factor
                55        D5  0D32         tstl    r5                        ; source scale factor negative?
                15        19  0D34         blss    10$                       ; if lss, yes
           1F   55        D1  0D36         cmpl    r5,#31                    ; trying to shift away all the bits?
                04        1F  0D39         blssu   5$                        ; if lssu then no
                50        D4  0D3B         clrl    r0                        ; else the result is zero
                07        11  0D3D         brb     7$                        ; go convert to decimal
      55   01   55        78  0D3F  5$:    ashl    r5,#1,r5                  ; calc 2** source scale factor
           50   55        C6  0D43         divl2   r5,r0                     ; convert source to zero scale integer
      62   53   50        F9  0D46  7$:    cvtlp   r0,r3,(r2)                ; do the conversion to decimal
                05            0D4A         rsb                               ; return
                          0D4B  ;
                          0D4B  ; convert number to stack
                          0D4B  ;
           5E   10        C2  0D4B  10$:   subl    #16,sp                    ; allocate more than enough room
                0E        BB  0D4E         pushr   #^m<r1,r2,r3>
   0C AE   51   50        F9  0D50         cvtlp   r0,r1,12(sp)              ; convert value
                0E        BA  0D55         popr    #^m<r1,r2,r3>
                55        D5  0D57         tstl    r5                        ; source scale factor zero?
                0B        12  0D59         bneq    20$                       ; if neq, no.
62   53   00   6E   51    54  F8  0D5B     ashp    r4,r1,(sp),#0,r3,(r2)     ; move to result field
           5E   10        C0  0D62         addl    #16,sp                    ; clean stack
```

```
                                   05  0D65  1970           rsb
                         5E   10   C2  0D66  1971 20$:      subl     #16,sp                       ; allocate another decimal buffer
                              52   DD  0D69  1972           pushl    r2                           ; save dest address
                              53   DD  0D6B  1973           pushl    r3                           ; save dest prec
      1F    00    18 AE  51   54   F8  0D6D  1974           ashp     r4,r1,24(sp),#0,#31,8(sp);make decimal integer
                         08 AE         0D74
                              55   D5  0D76  1975           tstl     r5                           ; source scale factor negative?
                              16   14  0D78  1976           bgtr     30$                          ; if gtr, no
                         55   55   CE  0D7A  1977           mnegl    r5,r5                        ; get abs(source scale)
                         55   06   C4  0D7D  1978           mull2    #6,r5                        ; use scale as an index to a power 2 table
        1F    00000000'GF45   0A   25  0D80  1979           mulp     #10,g^pli$b_pac_2_power_00[r5],-;mul by 2**(abs(source scale))
              04 BE    6E    08 AE     0D89
                                       0D8E  1980                    #31,8(sp),(sp),a4(sp)   ; and move to result
                              11   11  0D8E  1981           brb      40$                          ; join common return
                         55   06   C4  0D90  1982 30$:      mull2    #6,r5                        ; use scale as an index to a power 2 table
        1F    00000000'GF45   0A   27  CD93  1983           divp     #10,g^pli$b_pac_2_power_00[r5],-;div by 2**(source scale) and
              04 BE    6E    08 AE     0D9C
                                       0DA1  1984                    #31,8(sp),(sp),a4(sp)   ; move to result
                         5E   28   C0  0DA1  1985 40$:      addl     #40,sp                       ; clean up stack
                                   05  0DA4  1986           rsb                                   ; return
                                       0DA5  1987
                                       0DA5  1988 ;
                                       0DA5  1989 ; get_s:c_fixprec
                                       0DA5  1990 ;
                                       0DA5  1991 ; calc the number of digits based on a fixed bin precision
                                       0DA5  1992 ;
                                       0DA5  1993 get_src_fixprec:
                              54   DD  0DA5  1994           pushl    r4
                    54   51   01   C1  0DA7  1995           addl3    #1,r1,r4                     ; get fixed field size
               50   60   54   00   EE  0DAB  1996           extv     #0,r4,(r0),r0                ; get the value
                51   00000064 8F   C4  0DB0  1997           mull2    #100,r1                      ; get precision of result by rule
                51   00000297 8F   C0  0DB7  1998           addl     #663,r1
                51   0000014C 8F   C6  0DBE  1999           divl     #332,r1
                              54 8ED0  0DC5  2000           popl     r4
                                   05  0DC8  2001           rsb
                    0DC9         .sbttl  fixbfltd - fixed binary to float decimal conversion
                    0DC9   ; ++
                    0DC9   ; fixbfltd - fixed binary to float decimal conversion
                    0DC9   ;
                    0DC9   ; functional description:
                    0DC9   ;
                    0DC9   ; This routine converts a fixed binary value to a float decimal value.
                    0DC9   ;
                    0DC9   ; inputs:
                    0DC9   ;
                    0DC9   ;       r0 = address of the source
                    0DC9   ;       r1 = size or precision of source
                    0DC9   ;       r2 = address of the destination
                    0DC9   ;       r3 = size or the precision of the destination
                    0DC9   ;
                    0DC9   ; outputs:
                    0DC9   ;
                    0DC9   ;       The destination is filled in
                    0DC9   ; --
              C090  0DC9         .entry  pli$fixbfltd_r6,^m<iv,dv,r4,r7>
                    0DCB   fixbfltd:
     F880       30  0DCB         bsbw    dest_fltd_prec          ; get dest context
     FE52       30  0DCE         bsbw    cvrt_fixb_flt           ; continue in common
                04  0DD1         ret
```

```
                              0DD2  2028          .sbttl  fixbchar - convert fixed binary to character
                              0DD2  2029 ; ++
                              0DD2  2030 ; fixbchar - convert fixed binary to character
                              0DD2  2031 ;
                              0DD2  2032 ; functional description:
                              0DD2  2033 ;
                              0DD2  2034 ; This routine converts fixed binary numbers to character
                              0DD2  2035 ;
                              0DD2  2036 ; inputs:
                              0DD2  2037 ;
                              0DD2  2038 ;        r0 = source value
                              0DD2  2039 ;        r1 = precision of source
                              0DD2  2040 ;        r2 = address of the target
                              0DD2  2041 ;        r3 = size of the target
                              0DD2  2042 ;
                              0DD2  2043 ; outputs:
                              0DD2  2044 ;
                              0DD2  2045 ;        The output field is filled.
                              0DD2  2046 ; --
                        C070  0DD2  2047          .entry  pli$fixbchar_r6,^m<iv,dv,r4,r5,r6>
                              0DD4  2048 fixbchar:
                  F79B   30   0DD4  2049          bsbw    chk_fixb_string          ; check for possible overflow
           51  00FF 8F   B1   0DD7  2050          cmpw    #^xff,r1                 ; non-zero source scale?
                    16   1F   0DDC  2051          blssu   10$                      ; if lssu yes.
                  FFC4   30   0DDE  2052          bsbw    get_src_fixprec          ; convert precision of source
                51  03   C0   0DE1  2053          addl    #3,r1                    ; include for sign
                5E  51   C2   0DE4  2054          subl    r1,sp                    ; allocate the space
                56  5E   D0   0DE7  2055          movl    sp,r6                    ; save address
                  000E   30   0DEA  2056          bsbw    cvrt_fixb_char           ; do conversion
  62   53   20  66  51   2C   0DED  2057          movc5   r1,(r6),#32,r3,(r2)      ; move to target
                         04   0DF3  2058          ret                              ; return to caller
                              0DF4  2059
                  0033   30   0DF4  2060 10$:     bsbw    fixbfixdtemp             ; first convert to a fixed decimal temp
                  03BA   30   0DF7  2061          bsbw    fixdchar                 ; convert to char
                         04   0DFA  2062          ret                              ; never used fixdchar does ret.
                              0DFB  2063 ;
                              0DFB  2064 ; cvrt_fixb_char
                              0DFB  2065 ;
                              0DFB  2066 ; convert fixed bin to a character string
                              0DFB  2067 ;
                              0DFB  2068 ;
                              0DFB  2069 cvrt_fixb_char:
                    0F   BB   0DFB  2070          pushr   #^m<r0,r1,r2,r3>         ; save regs
  66   51   20  66  00   2C   0DFD  2071          movc5   #0,(r6),#32,r1,(r6)      ; fill with spaces
                    50 8ED0   0E03  2072          popl    r0                       ; get value
       50   00  50  01   7A   0E06  2073 10$:     emul    #1,r0,#0,r0              ; sign extend value
       52   50  50  0A   7B   0E0B  2074          ediv    #10,r0,r0,r2             ; get remainder
                51  52   D0   0E10  2075          movl    r2,r1                    ; get remainder
                    03   18   0E13  2076          bgeq    15$                      ; if geq then no
                51  51   CE   0E15  2077          mnegl   r1,r1                    ;
           73   51  30   81   0E18  2078 15$:     addb3   #^a/0/,r1,-(r3)          ; insert character
                    50   D5   0E1C  2079          tstl    r0                       ; quo = 0?
                    E6   12   0E1E  2080          bneq    10$                      ; if neq then no
                    52   D5   0E20  2081          tstl    r2                       ; last remainder negitive?
                    03   18   0E22  2082          bgeq    20$                      ; if geq then no
                73  2D   90   0E24  2083          movb    #^a/-/,-(r3)             ; insert minus sign
                    0E   BA   0E27  2084 20$:     popr    #^m<r1,r2,r3>            ;
```

```
                                05  0E29               rsb
                                    0E2A      ;
                                    0E2A      ; fixbfixdtemp
                                    0E2A      ;
                                    0E2A      ; convert fixed bin to a fixed decimal temporary
                                    0E2A      ;
                                    0E2A      ;        inputs:
                                    0E2A      ;               r0 - fixed bin value
                                    0E2A      ;               r1 - (p,q) of source
                                    0E2A      ;        outputs:
                                    0E2A      ;               r0 - address of converted value
                                    0E2A      ;               r1 - (p,q) of converted value
                                    0E2A      ;                       r4,r5 destroyed
                                    0E2A      ;
                                    0E2A     fixbfixdtemp:
                          56 8ED0   0E2A               popl    r6                      ; save return address
                     7E   52   7D   0E2D               movq    r2,-(sp)                ; save some regs
          55    51   F8 8F   78   0E30               ashl    #-8,r1,r5               ; get the source scale
                     54   51   9A   0E35               movzbl  r1,r4                   ; get source prec in longword
               51    54   01   C1   0E38               addl3   #1,r4,r1                ; get fixed field size
          50   60    51   00   EE   0E3C               extv    #0,r1,(r0),r0           ; get source value in a longword
                     5E   10   C2   0E41               subl    #16,sp                  ; allocate space for a decimal temp
               6E    1F   50   F9   0E44               cvtlp   r0,#31,(sp)             ; convert to decimal
                     55   55   98   0E48               cvtbl   r5,r5                   ; source scale negative?
                          06   14   0E4B               bgtr    15$                     ; if geq no
                     51   55   CE   0E4D               mnegl   r5,r1                   ; get abs(scale)
                     54   51   C0   0E50               addl    r1,r4                   ; calc number of decimal digits
          54   00000064 8F   C4   0E53     15$:      mull2   #100,r4                 ;
          54   00000297 8F   C0   0E5A               addl2   #663,r4                 ;
          54   0000014C 8F   C6   0E61               divl2   #332,r4                 ;
                          54   DD   0E68               pushl   r4                      ; save number of decimal digits
                          55   D5   0E6A               tstl    r5                      ; source scale negative?
                          20   18   0E6C               bgeq    16$                     ; if geq no
                     51   06   C4   0E6E               mull2   #6,r1                   ; use scale as index into a power 2 table
                     5E   10   C2   0E71               subl    #16,sp                  ; allocate more space
          0A    14 AE   1F   25   0E74               mulp    #31,20(sp),#10,-        ;scale up for implied zero bits
     6E    54    00000000'GF41        0E79                       g^pli$b_pac_2_power_00[r1],r4,(sp);
                     50   5E   D0   0E81               movl    sp,r0                   ; set up for convert fixd to char
               51    10 AE   D0   0E84               movl    16(sp),r1               ;
               52    24 AE   7D   0E88               movq    36(sp),r2               ;
                          66   17   0E8C               jmp     (r6)                    ; return
     54   55   00000064 8F   C5   0E8E     16$:      mull3   #100,r5,r4              ; convert scale to decimal precision
          54   0000014B 8F   C0   0E96               addl    #331,r4                 ;
          54   0000014C 8F   C6   0E9D               divl    #332,r4                 ;
                     5E   10   C2   0EA4               subl    #16,sp                  ; allocate a second decimal temp
6E   1F   00    14 AE   1F   54   F8   0EA7               ashp    r4,#31,20(sp),#0,#31,(sp);make a decimal integer
               11 AE    54   90   0EAF               movb    r4,17(sp)               ; set decimal scale
                     55   06   C4   0EB3               mull2   #6,r5                   ; use scale as index into power 2 table
               51    10 AE   9A   0EB6               movzbl  16(sp),r1               ; get decimal prec
  6E   1F   00000000'GF45   0A   27   0EBA               divp    #10,g^pli$b_pac_2_power_00[r5],-;div by 2**(source scale)
               14 AE    51        0EC4
                                    0EC7                       #31,(sp),r1,20(sp)      ;
               50    14 AE   DE   0EC7               moval   20(sp),r0               ; set up for convert fixd to char
               51    10 AE   D0   0ECB               movl    16(sp),r1               ;
               52    24 AE   7D   0ECF               movq    36(sp),r2               ;
                          66   17   0ED3               jmp     (r6)                    ; return
                                    0ED5
                          0ED5         .sbttl  fixbvcha - convert fixed binary to character varying
                          0ED5  ; ++
                          0ED5  ; fixbvcha - convert fixed binary to character varying
                          0ED5  ;
                          0ED5  ; functional description:
                          0ED5  ;
                          0ED5  ; This routine converts fixed binary numbers to character varying
                          0ED5  ;
                          0ED5  ; inputs:
                          0ED5  ;
                          0ED5  ;        r0 = source value
                          0ED5  ;        r1 = precision of source
                          0ED5  ;        r2 = address of the target
                          0ED5  ;        r3 = size of the target
                          0ED5  ;
                          0ED5  ; outputs:
                          0ED5  ;
                          0ED5  ;        The output field is filled.
                          0ED5  ; --
                    C070  0ED5         .entry  pli$fixbvcha_r6,^m<iv,dv,r4,r5,r6>
                          0ED7  fixbvcha:
           F698     30    0ED7         bsbw    chk_fixb_string          ; check for possible overflow
   51   00FF 8F     B1    0EDA         cmpw    #^xff,r1                 ; non-zero source scale?
             1F     1F    0EDF         blssu   20$                      ; if lssu yes.
           FEC1     30    0EE1         bsbw    get_src_fixprec          ; convert precision of source
        51   03     C0    0EE4         addl    #3,r1                    ; include for sign
        5E   51     C2    0EE7         subl    r1,sp                    ; allocate the space
        56   5E     D0    0EEA         movl    sp,r6                    ; save address
           FF0B     30    0EED         bsbw    cvrt_fixb_char           ; do conversion
        53   51     B1    0EF0         cmpw    r1,r3                    ; room enough?
             03     1B    0EF3         blequ   10$                      ; if lequ then yes
        51   53     D0    0EF5         movl    r3,r1                    ; use smaller size
        82   51     B0    0EF8  10$:   movw    r1,(r2)+                 ; insert size
   62   6E   51     28    0EFB         movc3   r1,(sp),(r2)             ; move to target
                    04    0EFF         ret
                          0F00
           FF27     30    0F00  20$:   bsbw    fixbfixdtemp             ; convert to fixed decimal temp
           0337     30    0F03         bsbw    fixdvcha                 ; convert to char
                    04    0F06         ret                              ; never used fixdvcha does ret.
                          0F07
```

```
                         0F07             .sbttl  fixbbit - fixed binary to bit string conversion
                         0F07             .sbttl  fixbabit - fixed binary to bit aligned conversion
                         0F07      ; ++
                         0F07      ; fixbabit - fixed binary to bit aligned conversion
                         0F07      ; fixbbit - fixed binary to bit string conversion
                         0F07      ;
                         0F07      ; functional description:
                         0F07      ;
                         0F07      ; This routine converts a fixed binary value to a bit aligned string.
                         0F07      ;
                         0F07      ; inputs:
                         0F07      ;
                         0F07      ;      r0 = address of the source
                         0F07      ;      r1 = size or precision of source
                         0F07      ;      r2 = address of the destination
                         0F07      ;      r3 = size or the precision of the destination
                         0F07      ;      r6 = bit offset to destination
                         0F07      ;
                         0F07      ; outputs:
                         0F07      ;
                         0F07      ;      The destination is filled in
                         0F07      ; --
                  C070   0F07             .entry  pli$fixbabit_r6,^m<iv,dv,r4,r5,r6>
                         0F09     fixbabit:
            0068    30   0F09             bsbw    clr_abit_trailer        ; clear abit last byte
              02    11   0F0C             brb     fixbbit
                  C030   0F0E             .entry  pli$fixbbit_r6,^m<iv,dv,r4,r5>
                         0F10     fixbbit:
            0001    30   0F10             bsbw    cvrt_fixb_bit
                    04   0F13             ret
                         0F14     cvrt_fixb_bit:
            F65B    30   0F14             bsbw    chk_fixb_string         ; check values
   55    51   01    81   0F17             addb3   #1,r1,r5
         55   55    9A   0F1B             movzbl  r5,r5                   ; zero extend field size
50   60  55   00    EE   0F1E             extv    #0,r5,(r0),r0           ; get sign extended value
              03    14   0F23             bgtr    5$                      ; branch if positive
         50   50    CE   0F25             mnegl   r0,r0                   ; make positive
  55   51  F8 8F    78   0F28     5$:     ashl    #-8,r1,r5               ; get source scale
         51   51    9A   0F2D             movzbl  r1,r1                   ; zero extend the source prec
         55   55    98   0F30             cvtbl   r5,r5                   ; sign extend source scale
         51   55    C2   0F33             subl2   r5,r1                   ; get prec minus scale
         55   55    CE   0F36             mnegl   r5,r5                   ; set up to convert to zero scale
   50    50   55    78   0F39             ashl    r5,r0,r0                ; convert to zero scale
         55   5E    D0   0F3D             movl    sp,r5                   ; address a temp
              7E    D4   0F40             clrl    -(sp)                   ;
                         0F42     ;
         54   50    9A   0F42     10$:    movzbl  r0,r4                   ; get low order byte of src
  75   F3B6 CF44    90   0F45             movb    reverse_bit_tbl[r4],-(r5) ; get reversed byte
 50    50  F8 8F    78   0F48             ashl    #-8,r0,r0               ; shift src down a byte
              F0    14   0F50             bgtr    10$                     ; if more, continue
                         0F52     ;
   55    20   51    C3   0F52             subl3   r1,#32,r5               ; adjust to converted bit prec
              06    14   0F56             bgtr    15$                     ; if 32-(prec-scale)>0 get value
         51   1F    D0   0F58             movl    #31,r1                  ; set max prec
         55   01    D0   0F5B             movl    #1,r5                   ; get full 31 bit field
6E   6E  51   55    EF   0F5E     15$:    extzv   r5,r1,(sp),(sp)         ;
         20   53    D1   0F63             cmpl    r3,#32                  ; see if dest. gtr longword
```

```
                   06   15  0F66  2240          bleq    20$                    ; if not, ok
                 001F   30  0F68  2241          bsbw    clr_bit_dest           ; else, clr bit dest.
              53   20   D0  0F6B  2242          movl    #32,r3                 ; set max src. prec.
    62   53   56   8E   F0  0F6E  2243 20$:     insv    (sp)+,r6,r3,(r2)       ; insert dest.
                        05  0F73  2244          rsb
                        0F74
                        0F74  ;
                        0F74  ; clr_abit_trailer
                        0F74  ;
                        0F74  ; inputs:
                        0F74  ;
                        0F74  ;         r2 = base address of the destination field
                        0F74  ;         r3 = size of the destination field
                        0F74  ;
                        0F74  ; outputs:
                        0F74  ;
                        0F74  ;         r6 = 0
                        0F74  ;         the last byte of the destination is cleared
                        0F74  ;
                        0F74  clr_abit_trailer:
          56   53   3C  0F74          movzwl  r3,r6                     ;
          56   07   C0  0F77          addl    #7,r6
          56   07   CA  0F7A          bicl    #7,r6
          56   53   C2  0F7D          subl    r3,r6                     ; any trailer?
               07   13  0F80          beql    10$                       ; if eql then n
 62   56   53   00   F0  0F82          insv    #0,r3,r6,(r2)             ; insert zero trailer
               56   D4  0F87          clrl    r6                        ;
                    05  0F89  10$:    rsb                               ; done
```

```
                                  0F8A  ;
                                  0F8A  ; clr_bit_dest
                                  0F8A  ;
                                  0F8A  ; inputs:
                                  0F8A  ;
                                  0F8A  ;        r2 = base address of the destination field
                                  0F8A  ;        r3 = size of the destination field
                                  0F8A  ;        r6 = offset to the destination field
                                  0F8A  ;
                                  0F8A  ; outputs:
                                  0F8A  ;
                                  0F8A  ;        destination field is zeroed
                                  0F8A  ;
                                  0F8A  clr_bit_dest:
                  20    53   D1   0F8A          cmpl    r3,#32                  ; short operation?
                        06   1A   0F8D          bgtru   10$                     ; if gtru then no
        62    53    56  00   F0   0F8F          insv    #0,r6,r3,(r2)           ; zero short field
                        05        0F94          rsb
                  007F  8F   BB   0F95  10$:    pushr   #^m<r0,r1,r2,r3,r4,r5,r6>; save registers
        54    56    03  00   EF   0F99          extzv   #0,#3,r6,r4             ; get offset byte bias
                        0F   13   0F9E          beql    20$                     ; if eql then byte aligned
              54    08  54   83   0FA0          subb3   r4,#8,r4                ; get remainder in byte
        62    54    56  00   F0   0FA4          insv    #0,r6,r4,(r2)           ; zero initial unaligned bits
                    56  54   C0   0FA9          addl    r4,r6                   ; byte aligned now
                    53  54   C2   0FAC          subl    r4,r3                   ; remove zeroed bits from count
              50    53  08   C7   0FAF  20$:    divl3   #8,r3,r0                ; calc number of bytes in field
                    56  08   C6   0FB3          divl    #8,r6                   ; calc number of bytes to field from base
        54    53    03  00   EF   0FB6          extzv   #0,#3,r3,r4             ; get end byte bias
                        09   13   0FBB          beql    30$                     ; if eql then byte sized
                    50  56   C0   0FBD          addl    r6,r0                   ; point to last byte
           6240     54  00   F0   0FC0          insv    #0,#0,r4,(r2)[r0]       ; zero end field
  6246   50    00 6246  00   2C   0FC6  30$:    movc5   #0,(r2)[r6],#0,r0,(r2)[r6]; clear middle
                  007F  8F   BA   0FCE          popr    #^m<r0,r1,r2,r3,r4,r5,r6>;
                        05        0FD2          rsb
```

```
                          0FD3          .sbttl  fixdpic - fixed decimal to picture conversion
                          0FD3   ; ++
                          0FD3   ; fixdpic - fixed decimal to picture conversion
                          0FD3   ;
                          0FD3   ; functional description:
                          0FD3   ;
                          0FD3   ; This routine converts a fixed decimal value to a picture value.
                          0FD3   ;
                          0FD3   ; inputs:
                          0FD3   ;
                          0FD3   ;       r0 = address of the source
                          0FD3   ;       r1 = size or precision of source
                          0FD3   ;       r2 = address of the destination
                          0FD3   ;       r3 = size or the precision of the destination
                          0FD3   ;
                          0FD3   ; outputs:
                          0FD3   ;
                          0FD3   ;       The destination is filled in
                          0FD3   ; --
                    C010  0FD3          .entry  pli$fixdpic_r6,^m<iv,dv,r4>
                          0FD5   fixdpic:
              52     DD   0FD5          pushl   r2                      ; target addr
     7E    04 A3     9A   0FD7          movzbl  pic$b_byte_size(r3),-(sp); target p,q
              50     DD   0FDB          pushl   r0                      ; src addr
              51     DD   0FDD          pushl   r1                      ; src p,q
              53     DD   0FDF          pushl   r3                      ; pic cons node
00000000'GF   05     FB   0FE1          calls   #5,g^pli$cvt_to_pic     ; convert to picture
                     04   0FE8          ret
```

```
                              0FE9                .sbttl  fixdfixb - fixed decimal to fixed binary conversion
                              0FE9      ; ++
                              0FE9      ; fixdfixb - fixed decimal to fixed binary conversion
                              0FE9      ;
                              0FE9      ; functional description:
                              0FE9      ;
                              0FE9      ; This routine converts a fixed decimal value to a fixed binary value.
                              0FE9      ;
                              0FE9      ; inputs:
                              0FE9      ;
                              0FE9      ;         r0 = address of the source
                              0FE9      ;         r1 = size or precision of source
                              0FE9      ;         r2 = address of the destination
                              0FE9      ;         r3 = size or the precision of the destination
                              0FE9      ;
                              0FE9      ; outputs:
                              0FE9      ;
                              0FE9      ;         The destination is filled in
                              0FE9      ;
                              0FE9      ; --
                        C030  0FE9                .entry  pli$fixdfixb_r6,^m<iv,dv,r4,r5>
                              0FEB      fixdfixb:
                   01     10  0FEB                bsbb    cvrt_fixd_fixb          ; use common routine
                          04  0FED                ret                             ;
                              0FEE      cvrt_fixd_fixb:
              5E   10     C2  0FEE                subl    #16,sp                  ; make a buffer
              55   51     9A  0FF1                movzbl  r1,r5                   ; get prec
     51   51   F8  8F     78  0FF4                ashl    #-8,r1,r1               ; get scale
              51   51     98  0FF9                cvtbl   r1,r1                   ; sign extend scale
              51   51     CE  0FFC                mnegl   r1,r1                   ; negate for shift off fraction digits
     54   53   F8  8F     78  0FFF                ashl    #-8,r3,r4               ; get destination scale
              53   53     9A  1004                movzbl  r3,r3                   ; zero extend dest prec
              54   54     98  1007                cvtbl   r4,r4                   ; sign extend dest scale, zero scale?
                   60     13  100A                beql    60$                     ; if eql yes
                   53     DD  100C                pushl   r3                      ; save destination prec and scale
              7E   51     7D  100E                movq    r1,-(sp)                ; save source prec and scale and target addr
              5E   10     C2  1011                subl    #16,sp                  ; allocate a second buffer
                   54     D5  1014                tstl    r4                      ; dest scale negative?
                   14     14  1016                bgtr    10$                     ; if gtr, no
              54   54     CE  1018                mnegl   r4,r4                   ; calc abs(dest scale)
              54   06     C4  101B                mull2   #6,r4                   ; use scale as offset into a power 2 table
 55   00000000'GF44   0A  27  101E                divp    #10,g^pli$b_pac_2_power_00[r4],-;truncate implied zero bits for fixe
             6E   1F  60      1027                        r5,(r0),#31,(sp)        ;
                   0F     11  102A                brb     20$                     ; join common code for pos and neg scale
              54   06     C4  102C      10$:      mull2   #6,r4                   ; use scale as offset into a power 2 table
 ;5   00000000'GF44   0A  25  102F                mulp    #10,g^pli$b_pac_2_power_00[r4],-; calc 2**(dest scale) * source
             6E   1F  60      1038                        r5,(r0),#31,(sp)        ;
1F   00   6E   1F  10 AE  F8  103B      20$:      ashp    16(sp),#31,(sp),#0,#31,28(sp); shift to truncate decimal fraction
                   1C AE      1042
       55   1C AE  1F     36  1044                cvtpl   #31,28(sp),r5           ; do conversion to integer
    53   18 AE  02    03  EF  1049                extzv   #3,#2,24(sp),r3         ; get context
             52   14 AE   D0  104F                movl    20(sp),r2               ; restore address of destination
              5E   2C     C0  1053                addl    #44,sp                  ; clean up the stack
                              1056                case    type=b,r3,<50$,40$,30$> ; case on destination context
              62   55     D0  1060      30$:      movl    r5,(r2)                 ;
                          05  1063                rsb
              62   55     F7  1064      40$:      cvtlw   r5,(r2)                 ;
                          05  1067                rsb
                        62    55   F6  1068  50$:   cvtlb   r5,(r2)                   ;
                                   05  106B         rsb
                        7E    52   7D  106C  60$:   movq    r2,-(sp)                  ;
08 AE    1F    00    60 55    51   F8  106F         ashp    r1,r5,(r0),#0,#31,8(sp)   ; shift into integer
                 55  08 AE    1F   36  1077         cvtpl   #31,8(sp),r5              ; do conversion
                        52    8E   7D  107C         movq    (sp)+,r2                  ; restore
                              55   DD  107F         pushl   r5                        ; store in memory
                        50    5E   D0  1081         movl    sp,r0                     ; address it
                        51    1F   D0  1084         movl    #31,r1                    ; set size
                            FAE2   30  1087         bsbw    cvrt_fixb_fixb            ; store result
                        5E    14   C0  108A         addl    #20,sp                    ; clean stack
                                   05  108D         rsb
```

```
                               108E          .sbttl  fixdfltb - fixed decimal to floating binary conversion
                               108E  ; ++
                               108E  ; fixdfltb - fixed decimal to floating binary conversion
                               108E  ;
                               108E  ; functional description:
                               108E  ;
                               108E  ; This routine converts a fixed decimal value to a floating binary value.
                               108E  ;
                               108E  ; inputs:
                               108E  ;
                               108E  ;       r0 = address of the source
                               108E  ;       r1 = size or precision of source
                               108E  ;       r2 = address of the destination
                               108E  ;       r3 = size or the precision of the destination
                               108E  ;
                               108E  ; outputs:
                               108E  ;
                               108E  ;       The destination is filled in
                               108E  ; --
                         C0F0  108E          .entry  pli$fixdfltb_r6,^m<iv,dv,r4,r5,r6,r7>
                               1090  fixdfltb:
                   F579    30  1090          bsbw    dest_fltb_prec          ; get dest context
                     01    10  1093          bsbb    cvrt_fixd_flt
                           04  1095          ret
                               1096  cvrt_fixd_flt:
      54    51    F8 8F    78  1096          ashl    #-8,r1,r4               ; save scale
            55       51    9A  109B          movzbl  r1,r5                   ; get prec
                               109E  ;
                               109E  ; try  u k convert by going to longword
                               109E  ;
                     20    B9  109E          bicpsw  #psl$m_iv               ; turn off int overflow
                     0F    BB  10A0          pushr   #^m<r0,r1,r2,r3>        ; save regs
          56    60   55    36  10A2          cvtpl   r5,(r0),r6              ; cvt packed to long
                     0F    BA  10A6          popr    #^m<r0,r1,r2,r3>        ; restore regs
                     41    1D  10A8          bvs     9$                      ; if overflow, do it the long way
                     20    B8  10AA          bispsw  #psl$m_iv               ; re-enable int overflow
                               10AC          case    type=b,r7,<1$,2$,3$>    ; case on dest type
               7E    56  6EFD  10B6          cvtlh   r6,-(sp)                ; cvrt to huge temp
  62    8E    F040 CF44  65FD  10BA          mulh3   h_power_of_10[r4],(sp)+,(r2)    ; adjust result for scale
                           05  10C2          rsb
               56    56    6E  10C3  1$:     cvtld   r6,r6                   ; cvrt to double
        56    EF35 CF44    64  10C6          muld2   d_power_of_10[r4],r6    ; adjust for scale
                     62    56    76  10CC          cvtdf   r6,(r2)                 ; cvrt to float result
                           05  10CF          rsb
               56    56    6E  10D0  2$:     cvtld   r6,r6                   ; cvrt to double
  62    56    EF28 CF44    65  10D3          muld3   d_power_of_10[r4],r6,(r2)       ;adjust result to scale
                           05  10DA          rsb
               7E    56  6EFD  10DB  3$:     cvtlh   r6,-(sp)                ; cvrt src to huge
        6E    F01? CF44  64FD  10DF          mulh2   h_power_of_10[r4],(sp)  ; adjust for scale
               62    8E  76FD  10E6          cvthg   (sp)+,(r2)              ; cvrt to grand result
                           05  10EA          rsb
                               10EB  ;
                               10EB  ; the long way
                               10EB  ;
                     20    B8  10EB  9$:     bispsw  #psl$m_iv               ; reset int overflow
               5E    20    C2  10ED          subl    #32,sp                  ; allocate temp for leading sep string
                     5E    DD  10F0          pushl   sp                      ; make a descriptor for l.s. str
```

```
         7E    55   01    C1  10F2  2460          addl3   #1,r5,-(sp)              ; inc sign byte in desc str length
               7E   52    7D  10F6  2461          movq    r2,-(sp)                 ; save dest. regs
  10 AE   55   60   55    08  10F9  2462          cvtps   r5,(r0),r5,16(sp)        ; cvrt packed to leading sep
               52   6E    7D  10FF  2463          movq    (sp),r2                  ; restore dest,but leave space on stack
                    7E    7C  1102  2464          clrq    -(sp)                    ; and make more room for return value
                              1104  2465 ;                                         ; make frame for convert call
                    7E    D4  1104  2466          clrl    -(sp)                    ; caller flags (default round)
                    00    DD  1106  2467          pushl   #0                       ; scale
                    00    DD  1108  2468          pushl   #0                       ; frac
                 0C AE    DF  110A  2469          pushal  12(sp)                   ; return addr
                 20 AE    DF  110D  2470          pushal  32(sp)                   ; src descriptor addr
                              1110  2471 ;
                              1110  2472          case    type=b,r7,<10$,20$,30$> ; case on dest context
                              111A  2473 ;
    00000000'GF     05    FB  111A  2474          calls   #5,g^ots$cvt_t_h         ; cvrt to huge
                 51 50    E9  1121  2475          blbc    r0,50$                   ; br if error
  62    6E   EFD6 CF44  65FD  1124  2476          mulh3   h_power_of_10[r4],(sp),(r2)     ; mul return value by scale
               5E   38    C0  112C  2477          addl    #56,sp                   ; clean stack
                          05  112F  2478          rsb
    00000000'GF     05    FB  1130  2479 10$:     calls   #5,g^ots$cvt_t_d         ; cvrt to double
                 3B 50    E9  1137  2480          blbc    r0,50$                   ; br if error
        6E   EEC1 CF44    64  113A  2481          muld2   d_power_of_10[r4],(sp)   ; adjust for scale
               62   6E    76  1140  2482          cvtdf   (sp),(r2)                ; cvrt result to float
               5E   38    C0  1143  2483          addl    #56,sp                   ; clean stack
                          05  1146  2484          rsb
    00000000'GF     05    FB  1147  2485 20$:     calls   #5,g^ots$cvt_t_d         ; cvrt to double
                 24 50    E9  114E  2486          blbc    r0,50$                   ; br if error
  62    6E   EEAA CF44    65  1151  2487          muld3   d_power_of_10[r4],(sp),(r2)     ; mul return value by scale
               5E   38    C0  1158  2488          addl    #56,sp                   ; clean stack
                          05  115B  2489          rsb
    00000000'GF     05    FB  115C  2490 30$:     calls   #5,g^ots$cvt_t_h         ; cvrt to huge
                 0F 50    E9  1163  2491          blbc    r0,50$                   ; br if error
        6E   EF94 CF44  64FD  1166  2492          mulh2   h_power_of_10[r4],(sp)   ; adjust for scale
               62   6E  76FD  116D  2493          cvthg   (sp),(r2)                ; cvrt result to grand
               5E   38    C0  1171  2494          addl    #56,sp                   ; clean stack
                          05  1174  2495          rsb
                  F2DD    31  1175  2496 50$:     brw     error
```

```
                                        1178            .sbttl  fixdfixd - fixed decimal to fixed decimal conversion
                                        1178      ; ++
                                        1178      ; fixdfixd - fixed decimal to fixed decimal conversion
                                        1178      ;
                                        1178      ; functional description:
                                        1178      ;
                                        1178      ; This routine converts a fixed decimal value to a fixed decimal value.
                                        1178      ;
                                        1178      ; inputs:
                                        1178      ;
                                        1178      ;         r0 = address of the source
                                        1178      ;         r1 = size or precision of source
                                        1178      ;         r2 = address of the destination
                                        1178      ;         r3 = size or the precision of the destination
                                        1178      ;
                                        1178      ; outputs:
                                        1178      ;
                                        1178      ;         The destination is filled in
                                        1178      ; --
                                  C030  1178            .entry  pli$fixdfixd_r6,^m<iv,dv,r4,r5>
                                        117A    fixdfixd:
                         54   51    9A  117A            movzbl  r1,r4                      ; get prec and scale
               51   51     F8 8F    78  117D            ashl    #-8,r1,r1                  ;
                         55   53    9A  1182            movzbl  r3,r5                      ;
               53   53     F8 8F    78  1185            ashl    #-8,r3,r3                  ;
                         53   51    C2  118A            subl    r1,r3                      ; calc scale change
   62   55   00   60     54   53    F8  118D            ashp    r3,r4,(r0),#0,r5,(r2)      ; move data
                                    04  1194            ret
```

```
                   1195        .sbttl  fixdfltd - fixed decimal to float decimal conversion
                   1195 ; ++
                   1195 ; fixdfltd - fixed decimal to float decimal conversion
                   1195 ;
                   1195 ; functional description:
                   1195 ;
                   1195 ; This routine converts a fixed decimal value to a float decimal value.
                   1195 ;
                   1195 ; inputs:
                   1195 ;
                   1195 ;       r0 = address of the source
                   1195 ;       r1 = size or precision of source
                   1195 ;       r2 = address of the destination
                   1195 ;       r3 = size or the precision of the destination
                   1195 ;
                   1195 ; outputs:
                   1195 ;
                   1195 ;       The destination is filled in
                   1195 ; --
              C0F0 1195        .entry  pli$fixdfltd_r6,^m<iv,dv,r4,r5,r6,r7>
                   1197 fixdfltd:
        F4B4   30  1197        bsbw    dest_fltd_prec          ; get dest context
        FEF9   30  119A        bsbw    cvrt_fixd_flt           ; continue in common
               04  119D        ret
```

```
                                  119E  2552 ; ++
                                  119E  2553 ; fixdchar - fixed dccimal to character conversion
                                  119E  2554 ;
                                  119E  2555 ; functional description:
                                  119E  2556 ;
                                  119E  2557 ; This routine converts a fixed decimal value to a character string.
                                  119E  2558 ;
                                  119E  2559 ; inputs:
                                  119E  2560 ;
                                  119E  2561 ;       r0 = address of the source
                                  119E  2562 ;       r1 = size or precision of source
                                  119E  2563 ;       r2 = address of the destination
                                  119E  2564 ;       r3 = size or the precision of the destination
                                  119E  2565 ;
                                  119E  2566 ; outputs:
                                  119E  2567 ;
                                  119E  2568 ;       The destination is filled in
                                  119E  2569 ; --
                                  119E  2570
                                  119E  2571 edit_beg:
                                  119E  2572         eo$insert       <^x20>
                                  11A0  2573         eo$insert       <^x20>
                                  11A2  2574 edint:  eo$float        0
                                  11A3  2575         eo$float        15
                                  11A4  2576         eo$end_float
                                  11A5  2577         eo$set_signif
                                  11A6  2578         eo$move         1
                                  11A7  2579 edpt:   eo$insert       <^a/./>
                                  11A9  2580 edfrac: eo$move         0
                                  11AA  2581         eo$move         15
                                  11AB  2582         eo$move         1
                                  11AC  2583         eo$end
                                  11AD  2584         eo$end
                                  11AE  2585 edit_end:
                                  11AE  2586
                                  11AE  2587 no_int: eo$set_signif
                                  11AF  2588         eo$store_sign
                                  11B0  2589         eo$insert       <^a/0/>
                                  11B2  2590
                        00000010  11B2  2591 edit_len        =       edit_end-edit_beg
                        00000004  11B2  2592 edit_int        =       edint-edit_beg
                        00000009  11B2  2593 edit_pt         =       edpt-edit_beg
                        00000008  11B2  2594 edit_frac       =       edfrac-edit_beg
                                  11B2  2595
                            C070  11B2  2596         .entry  pli$fixdchar_r6,^m<iv,dv,r4,r5,r6>
                                  11B4  2597 fixdchar:
              54    51      9A    11B4  2598         movzbl  r1,r4                   ;
        56    54    03      C1    11B7  2599         addl3   #3,r4,r6                ; r6 is the precision based size
              5E    56      C2    11BB  2600         subl    r6,sp                   ; allocate the space on the stack
              7E    53      7D    11BE  2601         movq    r3,-(sp)                ; save regs
                    52      DD    11C1  2602         pushl   r2                      ; save r2
           7E    E0 AF      7D    11C3  2603         movq    edit_beg+8,-(sp)        ; copy end of edit table to stack
           7E    D4 AF      7D    11C7  2604         movq    edit_beg,-(sp)          ; copy beginning of table to stack
              52    5E      D0    11CB  2605         movl    sp,r2                   ; save address of beginning of table
  55    51   F8 8F          78    11CE  2606         ashl    #-8,r1,r5               ; get scale
                    05      12    11D3  2607         bneq    10$                     ; if neq, scale present
              09 AE         94    11D5  2608         clrb    edit_pt(sp)             ; no scale, don't do dec pt or frac
```

```
                    22   11  11D8        brb     20$                       ; continue in common
                 54 55   C2  11DA 10$:   subl    r5,r4                     ; get size of int part
                 55 10   C2  11DD        subl    #16,r5                    ; scale > 16
                    0E   14  11E0        bgtr    15$                       ; if gtr, yes
                    06   12  11E2        bneq    14$                       ; if neg, scale < 16
           0B AE    03   90  11E4        movb    #3,edit_frac(sp)          ; nop first move of frac
                    0C   11  11E8        brb     16$                       ; continue
                 0C AE   94  11EA 14$:   clrb    edit_frac+1(sp)           ; skip last move for fraction
                 55 10   C0  11ED        addl    #16,r5                    ; readjust scale
     0B AE    04    00 55   F0  11F0 15$:   insv    r5,#0,#4,edit_frac(sp)    ; set size of fraction
                    82   B5  11F6 16$:   tstw    (r2)+                     ; skip first insert in table
                    54   D5  11F8        tstl    r4                        ; check size of integer part
                    27   13  11FA        beql    40$                       ; if eql, no integer part
                    54   D7  11FC 20$:   decl    r4                        ; calculate size of float int part
                    2D   13  11FE        beql    50$                       ; if eql, only 1 digit integer
                 54 0F   C2  1200        subl    #15,r4                    ; int part > 15 digits?
                    07   14  1203        bgtr    25$                       ; if gtr, yes
           05 AE    01   90  1205        movb    #1,edit_int+1(sp)         ; don't do second float
                 54 0F   C0  1209        addl    #15,r4                    ; readjust size
     04 AE    04    00 54   F0  120C 25$:   insv    r4,#0,#4,edit_int(sp)     ; set size of float int part
 1C AE   62    60    18 AE   38  1212 30$:   editpc  24(sp),(r0),(r2),28(sp)   ; edit the string
 14 AE   20    1C AE    56   2C  1219        movc5   r6,28(sp),#^x20,20(sp),@16(sp) ; copy to destination
                 10 BE       1220
                         04  1222        ret                               ; and return
              52    05 AE    9E  1223 40$:   movab   edit_int+1(sp),r2         ; get address of new start of table
              62    84 AF    D0  1227        movl    no_int,(r2)               ; copy new start of table
                       E5    11  122B        brb     30$                       ; continue in common
        04 AE    FF7D CF     90  122D 50$:   movb    no_int,edit_int(sp)       ; nop first byte of float int part
        05 AE    FF77 CF     B0  1233        movw    no_int,edit_int+1(sp)     ; nop rest of float int part
                       D7    11  1239        brb     30$                       ; continue in common
                             123B
                    123B              .sbttl  fixdvcha - fixed decimal to character varying conversion
                    123B      ; ++
                    123B      ; fixdvcha - fixed decimal to character varying conversion
                    123B      ;
                    123B      ; functional description:
                    123B      ;
                    123B      ; This routine converts a fixed decimal value to a character varying string.
                    123B      ;
                    123B      ; inputs:
                    123B      ;
                    123B      ;       r0 = address of the source
                    123B      ;       r1 = size or precision of source
                    123B      ;       r2 = address of the destination
                    123B      ;       r3 = size or the precision of the destination
                    123B      ;
                    123B      ; outputs:
                    123B      ;
                    123B      ;       The destination is filled in
                    123B      ; --
              C070  123B              .entry  pli$fixdvcha_r6,^m<iv,dv,r4,r5,r6>
                    123D      fixdvcha:
  54    51    9A    123D              movzbl  r1,r4                           ; get precision of source
  54    03    C0    1240              addl    #3,r4                           ; get size of dest based on precision
  62    54    B0    1243              movw    r4,(r2)                         ; insert size
  53    54    B1    1246              cmpw    r4,r3                           ; destination large enough?
        03    1B    1249              blequ   10$                             ; if lequ then yes
  62    53    B0    124B              movw    r3,(r2)                         ; insert max size
        82    B5    124E      10$:    tstw    (r2)+                           ; address actual text target
      FF61    31    1250              brw     fixdchar                        ; continue in common
```

```
                              1253              .sbttl  fixdabit - fixed decimal to bit aligned conversion
                              1253      ; ++
                              1253      ; fixdabit - fixed decimal to bit aligned conversion
                              1253      ; fixdbit - fixed decimal to bit string conversion
                              1253      ;
                              1253      ; functional description:
                              1253      ;
                              1253      ; This routine converts a fixed decimal value to a bit aligned string.
                              1253      ;
                              1253      ; inputs:
                              1253      ;
                              1253      ;       r0 = address of the source
                              1253      ;       r1 = size or precision of source
                              1253      ;       r2 = address of the destination
                              1253      ;       r3 = size or the precision of the destination
                              1253      ;       r6 = bit offset to destination
                              1253      ;
                              1253      ; outputs:
                              1253      ;
                              1253      ;       The destination is filled in
                              1253      ;
                              1253      ; --
                       C070   1253              .entry  pli$fixdabit_r6,^m<iv,dv,r4,r5,r6>
                              1255      fixdabit:
               FD1C     30    1255              bsbw    clr_abit_trailer        ; clear abit last byte
                 02     11    1258              brb     fixdbit
                       C030   125A              .entry  pli$fixdbit_r6,^m<iv,dv,r4,r5>
                              125C      fixdbit:
           5E    04     C2    125C              subl    #4,sp                   ; get space for temp
           004C 8F      BB    125F              pushr   #^m<r2,r3,r6>           ; save destination
        52    0C AE     DE    1263              moval   12(sp),r2               ; plug address of temp for dest
           53    51     9A    1267              movzbl  r1,r3                   ; get src prec
   54    51    F8 8F    78    126A              ashl    #-8,r1,r4               ; get src scale
           53    54     C2    126F              subl2   r4,r3                   ; prec-scale
   53   0000014C 8F     C4    1272              mull    #332,r3                 ; conv prec from dec to binary digits
   53   00000063 8F     C0    1279              addl    #99,r3
   53   00000064 8F     C6    1280              divl    #100,r3
           1F    53     D1    1287              cmpl    r3,#31                  ; check for max prec
           03    15           128A              bleq    20$                     ; if leq, br
           53    1F     D0    128C              movl    #31,r3                  ; else set dest prec to max
                 53     DD    128F      20$:    pushl   r3                      ; save  binary prec
           53    1F     D0    1291              movl    #31,r3                  ; convert to fixed bin(31)
               FD57     30    1294              bsbw    cvrt_fixd_fixb          ; convert source to fixb temp
              51 8ED0         1297              popl    r1                      ; reset src prec to binary prec
           004C 8F      BA    129A              popr    #^m<r2,r3,r6>           ; restore destination
           50    5E     D0    129E              movl    sp,r0                   ; plug address of temp for source
               FC6C     31    12A1              brw     fixbbit                 ; done
```

```
                   12A4            .sbttl  fltdpic - float decimal to picture conversion
                   12A4      ; ++
                   12A4      ; fltdpic - float decimal to picture conversion
                   12A4      ;
                   12A4      ; functional description:
                   12A4      ;
                   12A4      ; This routine converts a float decimal value to a picture value.
                   12A4      ;
                   12A4      ; inputs:
                   12A4      ;
                   12A4      ;         r0 = address of the source
                   12A4      ;         r1 = size or precision of source
                   12A4      ;         r2 = address of the destination
                   12A4      ;         r3 = size or the precision of the destination
                   12A4      ;
                   12A4      ; outputs:
                   12A4      ;
                   12A4      ;         The destination is filled in
                   12A4      ; --
             C1F0  12A4            .entry  pli$fltdpic_r6,^m<iv,dv,r4,r5,r6,r7,r8>
                   12A6      fltdpic:
     F384     30   12A6            bsbw    src_fltd_prec           ; get src context
     F4E1     30   12A9            bsbw    cvrt_flt_pic            ; cont in common
              04   12AC            ret
```

```
                 12AD          .sbttl  fltdfixb - float decimal to fixed binary conversion
                 12AD  ; ++
                 12AD  ; fltdfixb - float decimal to fixed binary conversion
                 12AD  ;
                 12AD  ; functional description:
                 12AD  ;
                 12AD  ; This routine converts a float decimal value to a fixed binary value.
                 12AD  ;
                 12AD  ; inputs:
                 12AD  ;
                 12AD  ;       r0 = address of the source
                 12AD  ;       r1 = size or precision of source
                 12AD  ;       r2 = address of the destination
                 12AD  ;       r3 = size or the precision of the destination
                 12AD  ;
                 12AD  ; outputs:
                 12AD  ;
                 12AD  ;       The destination is filled in
                 12AD  ; --
           C010  12AD          .entry  pli$fltdfixb_r6,^m<iv,dv,r4>
                 12AF  fltdfixb:
      F37B   30  12AF          bsbw    src_fltd_prec           ; get src context
      F505   30  12B2          bsbw    cvrt_flt_fixb           ; do conversion
             04  12B5          ret
```

```
                 12B6         .sbttl  fltdfltb - float decimal to float binary conversion
                 12B6  ; ++
                 12B6  ; fltdfltb - float decimal to float binary conversion
                 12B6  ;
                 12B6  ; functional description:
                 12B6  ;
                 12B6  ; This routine converts a float decimal value to a float binary value.
                 12B6  ;
                 12B6  ; inputs:
                 12B6  ;
                 12B6  ;      r0 = address of the source
                 12B6  ;      r1 = size or precision of source
                 12B6  ;      r2 = address of the destination
                 12B6  ;      r3 = size or the precision of the destination
                 12B6  ;
                 12B6  ; outputs:
                 12B6  ;
                 12B6  ;      The destination is filled in
                 12B6  ; --
           C090  12B6         .entry  pli$fltdfltb_r6,^m<iv,dv,r4,r7>
                 12B8  fltdfltb:
F372    30       12B8         bsbw    src_fltd_prec          ; get src context
F34E    30       12BB         bsbw    dest_fltb_prec         ; get dest context
F5F4    30       12BE         bsbw    cvrt_flt_flt           ; cont in common
        04       12C1         ret
```

```
                12C2          .sbttl  fltdfixd - float decimal to fixed decimal conversion
                12C2   ; ++
                12C2   ; fltdfixd - float decimal to fixed decimal conversion
                12C2   ;
                12C2   ; functional description:
                12C2   ;
                12C2   ; This routine converts a float decimal value to a fixed decimal value.
                12C2   ;
                12C2   ; inputs:
                12C2   ;
                12C2   ;      r0 = address of the source
                12C2   ;      r1 = size or precision of source
                12C2   ;      r2 = address of the destination
                12C2   ;      r3 = size or the precision of the destination
                12C2   ;
                12C2   ; outputs:
                12C2   ;
                12C2   ;      The destination is filled in
                12C2   ; --
          C1F0  12C2          .entry  pli$fltdfixd_r6,^m<iv,dv,r4,r5,r6,r7,r8>
                12C4   fltdfixd:
   F366    30   12C4          bsbw    src_fltd_prec           ; get src context
   F670    30   12C7          bsbw    cvrt_flt_fixd           ; do conversion
           04   12CA          ret
```

```
                12CB          .sbttl  fltdfltd - float decimal to float decimal conversion
                12CB   ; ++
                12CB   ; fltdfltd - float decimal to float decimal conversion
                12CB   ;
                12CB   ; functional description:
                12CB   ;
                12CB   ; This routine converts a float decimal value to a float decimal value.
                12CB   ;
                12CB   ; inputs:
                12CB   ;
                12CB   ;      r0 = address of the source
                12CB   ;      r1 = size or precision of source
                12CB   ;      r2 = address of the destination
                12CB   ;      r3 = size or the precision of the destination
                12CB   ;
                12CB   ; outputs:
                12CB   ;
                12CB   ;      The destination is filled in
                12CB   ; --
          C090  12CB          .entry  pli$fltdfltd_r6,^m<iv,dv,r4,r7>
                12CD   fltdfltd:
   F35D   30    12CD          bsbw    src_fltd_prec           ; get src context
   F37B   30    12D0          bsbw    dest_fltd_prec          ; get dest context
   F5DF   30    12D3          bsbw    cvrt_flt_flt            ; cont in common
          04    12D6          ret
```

```
                 12D7        .sbttl  fltdchar - float decimal to character conversion
                 12D7 ; ++
                 12D7 ; fltdchar - float decimal to character conversion
                 12D7 ;
                 12D7 ; functional description:
                 12D7 ;
                 12D7 ; This routine converts a float decimal value to a character value.
                 12D7 ;
                 12D7 ; inputs:
                 12D7 ;
                 12D7 ;      r0 = address of the source
                 12D7 ;      r1 = size or precision of source
                 12D7 ;      r2 = address of the destination
                 12D7 ;      r3 = size or the precision of the destination
                 12D7 ;
                 12D7 ; outputs:
                 12D7 ;
                 12D7 ;      The destination is filled in
                 12D7 ; --
            CFF0 12D7        .entry  pli$fltdchar_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                 12D9 fltdchar:
    F351    30   12D9        bsbw    src_fltd_prec           ; get src context
    F74F    30   12DC        bsbw    cvrt_flt_char           ; do conversion
            04   12DF        ret
```

```
                   12E0           .sbttl  fltdvcha - float decimal to character varying conversion
                   12E0   ; ++
                   12E0   ; fltdvcha - float decimal to character varying conversion
                   12E0   ;
                   12E0   ; functional description:
                   12E0   ;
                   12E0   ; This routine converts a float decimal value to a character varying value.
                   12E0   ;
                   12E0   ; inputs:
                   12E0   ;
                   12E0   ;       r0 = address of the source
                   12E0   ;       r1 = size or precision of source
                   12E0   ;       r2 = address of the destination
                   12E0   ;       r3 = size or the precision of the destination
                   12E0   ;
                   12E0   ; outputs:
                   12E0   ;
                   12E0   ;       The destination is filled in
                   12E0   ; --
             CFF0  12E0           .entry  pli$fltdvcha_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                   12E2   fltdvcha:
        82    3F   12E2           pushaw  (r2)+                   ; save dest & point to string
      F346    30   12E4           bsbw     src_fltd_prec          ; get src context
      F744    30   12E7           bsbw    cvrt_flt_char           ; do conversion
   9E   50    B0   12EA           movw    r0,@(sp)+               ; plug in size
              04   12ED           ret
```

```
                                 12EE        .sbttl  fltdbit - float decimal to bit conversion
                                 12EE  ; ++
                                 12EE  ; fltdbit - float decimal to bit conversion
                                 12EE  ;
                                 12EE  ; functional description:
                                 12EE  ;
                                 12EE  ; This routine converts a float decimal value to a bit value.
                                 12EE  ;
                                 12EE  ; inputs:
                                 12EE  ;
                                 12EE  ;       r0 = address of the source
                                 12EE  ;       r1 = size or precision of source
                                 12EE  ;       r2 = address of the destination
                                 12EE  ;       r3 = size or the precision of the destination
                                 12EE  ;
                                 12EE  ; outputs:
                                 12EE  ;
                                 12EE  ;       The destination is filled in
                                 12EE  ; --
                          C030   12EE        .entry  pli$fltdbit_r6,^m<iv,dv,r4,r5>
                                 12F0  fltdbit:
51   0000014C 8F    C4   12F0        mull    #332,r1                   ; convert decimal to binary prec
51   00000063 8F    C0   12F7        addl    #99,r1
51   00000064 8F    C6   12FE        divl    #100,r1
              F815  31   1305        brw     fltbbit                   ; continue in common
```

```
                   1308          .sbttl  fltdabit - float decimal to bit aligned conversion
                   1308  ; ++
                   1308  ; fltdabit - float decimal to bit aligned conversion
                   1308  ;
                   1308  ; functional description:
                   1308  ;
                   1308  ; This routine converts a float decimal value to a bit aligned value.
                   1308  ;
                   1308  ; inputs:
                   1308  ;
                   1308  ;       r0 = address of the source
                   1308  ;       r1 = size or precision of source
                   1308  ;       r2 = address of the destination
                   1308  ;       r3 = size or the precision of the destination
                   1308  ;
                   1308  ; outputs:
                   1308  ;
                   1308  ;       The destination is filled in
                   1308  ; --
           C070    1308          .entry  pli$fltdabit_r6,^m<iv,dv,r4,r5,r6>
                   130A  fltdabit:
    FC67     30    130A          bsbw    clr_abit_trailer        ; clear abit last byte
    FFE0     31    130D          brw     fltdbit
```

```
                               1310          .sbttl  charpic - character string to picture conversion
                               1310   ; ++
                               1310   ; charpic - character string to picture conversion
                               1310   ;
                               1310   ; functional description:
                               1310   ;
                               1310   ; This routine converts a character string to a picture value.
                               1310   ;
                               1310   ; inputs:
                               1310   ;
                               1310   ;       r0 = address of the source
                               1310   ;       r1 = size or precision of source
                               1310   ;       r2 = address of the destination
                               1310   ;       r3 = size or the precision of the destination
                               1310   ;
                               1310   ; outputs:
                               1310   ;
                               1310   ;       The destination is filled in
                               1310   ; --
                        CFF0   1310          .entry  pli$charpic_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                               1312  charpic:
      6E   F209 CF        9E   1312          movab   pli$cnvrt_hnd,(sp)      ; conversion condition handler
        5A   24'AF        9E   1317          movab   b^10$,r10               ; address completion routine
            5B   53       D0   131B          movl    r3,r11                  ; save pic node addr
            53   63       32   131E          cvtwl   pic$w_pq(r3),r3         ; set to target p,q
              01C5        31   1321          brw     charfix                 ; convert to fixed decimal
                               1324   ;
                               1324   ; complete processing
                               1324   ;
                               1324  10$:
                 52       DD   1324          pushl   r2                      ; target addr
         7E   04 AB       9A   1326          movzbl  pic$b_byte_size(r11),-(sp); target size
                 50       DD   132A          pushl   r0                      ; src addr
                 51       DD   132C          pushl   r1                      ; src p,q
                 5B       DD   132E          pushl   r11                     ; pic node addr
  00000000'GF    05       FB   1330          calls   #5,g^pli$cvt_to_pic     ; convert to picture
                          04   1337          ret
```

```
                              1338          .sbttl  charfixb - character string to fixed binary conversion
                              1338    ; ++
                              1338    ; charfixb - character string to fixed binary conversion
                              1338    ;
                              1338    ; functional d-scription:
                              1338    ;
                              1338    ; This routine converts a character string to a fixed binary value.
                              1338    ;
                              1338    ; inputs:
                              1338    ;
                              1338    ;         r0 = address of the source
                              1338    ;         r1 = size or precision of source
                              1338    ;         r2 = address of the destination
                              1338    ;         r3 = size or the precision of the destination
                              1338    ;
                              1338    ; outputs:
                              1338    ;
                              1338    ;         The destination is filled in
                              1338    ; --
                        C7F0  1338            .entry  pli$charfixb_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10>
                              133A    charfixb:
  6D    F1E1 CF    9E         133A            movab   pli$cnvrt_hnd,(fp)       ; conversion condition handler
  5A    FCA8 CF    9E         133F            movab   w^fixdfixb,r10           ; pass address of completion routine
             01A2  31         1344            brw     charfix                  ; continue
```

```
                                 1347  3009          .sbttl  charfltb - character string to floating binary conversion
                                 1347  3010 ; ++
                                 1347  3011 ; charfltb - character string to floating binary conversion
                                 1347  3012 ;
                                 1347  3013 ; functional description:
                                 1347  3014 ;
                                 1347  3015 ; This routine converts a character string to a floating binary value.
                                 1347  3016 ;
                                 1347  3017 ; inputs:
                                 1347  3018 ;
                                 1347  3019 ;       r0 = address of the source
                                 1347  3020 ;       r1 = size or precision of source
                                 1347  3021 ;       r2 = address of the destination
                                 1347  3022 ;       r3 = size or the precision of the destination
                                 1347  3023 ;
                                 1347  3024 ; outputs:
                                 1347  3025 ;
                                 1347  3026 ;       The destination is filled in
                                 1347  3027 ; --
                           C090  1347  3028         .entry  pli$charfltb_r6,^m<iv,dv,r4,r7>
                                 1349  3029 charfltb:
         6D    F1D2 CF     9E    1349  3030         movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                  F2BB     30    134E  3031         bsbw    dest_fltb_prec          ; get dest context
                    01     10    1351  3032         bsbb    cvrt_char_flt
                           04    1353  3033         ret
                                 1354  3034 cvrt_char_flt:
                    54     D4    1354  3035         clrl    r4                      ;set no default fractional digits
                                 1356  3036 cvrt_fchr_flt:                           ;entry with fractional digits
              5E    10     C2    1356  3037         subl    #16,sp                  ; allocate a place for the return
                    50     DD    1359  3038         pushl   r0                      ; set up source desc
                    51     DD    135B  3039         pushl   r1                      ;
                    7E     D4    135D  3040         clrl    -(sp)                   ; caller flags: default round
                    00     DD    135F  3041         pushl   #0                      ; set scale
                    54     DD    1361  3042         pushl   r4                      ; set fraction size
                 14 AE     DF    1363  3043         pushal  20(sp)                  ; address return
                 10 AE     DF    1366  3044         pushal  16(sp)                  ; address source descr
     18 BE    14 AE  20    3B    1369  3045         skpc    #^x20,20(sp),@24(sp)    ; skip leading blanks
                    08     13    136F  3046         beql    5$                      ; all blanks, ok
         61    50    20    3A    1371  3047         locc    #^x20,r0,(r1)           ; find next blank
          14 AE    50      C2    1375  3048         subl    r0,20(sp)               ; treat as end of string
                                 1379  3049 5$:     case    type=b,r7,<6$,7$,8$>    ; case to appropriate conversion
   00000000'GF      05     FB    1383  3050         calls   #5,g^ots$cvt_t_h
                 40 50     E9    138A  3051         blbc    r0,20$
         62    08 AE     70FD    138D  3052         movh    8(sp),(r2)
              5E    18     C0    1392  3053         addl    #24,sp                  ; clean stack
                           05    1395  3054         rsb
   00000000'GF      05     FB    1396  3055 6$:     calls   #5,g^ots$cvt_t_d
                 2D 50     E9    139D  3056         blbc    r0,20$
         62    08 AE       76    13A0  3057         cvtdf   8(sp),(r2)
              5E    18     C0    13A4  3058         addl    #24,sp                  ; clean stack
                           05    13A7  3059         rsb
   00000000'GF      05     FB    13A8  3060 7$:     calls   #5,g^ots$cvt_t_d
                 1B 50     E9    13AF  3061         blbc    r0,20$
         62    08 AE       70    13B2  3062         movd    8(sp),(r2)
              5E    18     C0    13B6  3063         addl    #24,sp                  ; clean stack
                           05    13B9  3064         rsb
   00000000'GF      05     FB    13BA  3065 8$:     calls   #5,g^ots$cvt_t_g
```

```
          09 50    E9  13C1   3066         blbc    r0,20$
     62   08 AE  50FD  13C4   3067         movg    8(sp),(r2)
       5E    18    C0  13C9   3068         addl    #24,sp                   ; clean stack
                   05  13CC   3069         rsb
                       13CD   3070 ;
            F085   31  13CD   3071 20$:    brw     error                    ; continue - no stack cleanup needed
```

```
                13D0          .sbttl  charfixd - character string to fixed decimal conversion
                13D0      ; ++
                13D0      ; charfixd - character string to fixed decimal conversion
                13D0      ;
                13D0      ; functional description:
                13D0      ;
                13D0      ; This routine converts character strings of the form:
                13D0      ; [<blanks>][sign][integer][.[fraction]][e!E[sign]exponent][<blanks>]
                13D0      ; to a fixed decimal value.
                13D0      ;
                13D0      ; inputs:
                13D0      ;       r0 = address of source
                13D0      ;       r1 = length of source
                13D0      ;       r2 = address of destination
                13D0      ;       r3 = precision and scale of the destination
                13D0      ;
                13D0      ; outputs:
                13D0      ;       r0-r5 destroyed
                13D0      ;       r6-r14 preserved
                13D0      ;       the input operand is converted to fixed decimal.
                13D0      ;
                13D0      ; local register usage
                13D0      ;       r0-r5 clobbered by string instructions
                13D0      ;       r6 = address of next byte in source string
                13D0      ;       r7 = number of bytes remaining in source string
                13D0      ;       r8 = address of next byte in leading separate temp
                13D0      ;       r9 = mask value for scanc
                13D0      ;       r10 = address of routine to convert from fixd to final destination
                13D0      ;
                13D0      ; --
                13D0
                13D0      ;
                13D0      ; local symbols
                13D0      ;
                13D0
      00000001  13D0      blank=1
      00000002  13D0      pt=2
      00000004  13D0      exp=4
                13D0
                13D0      ;
                13D0      ; local data
                13D0      ;
                13D0
                13D0      scantbl:
      000014D0  13D0              .blkb   256
      000014D0  14D0      $$$t1=.
      000013F0  14D0      .=scantbl+^x20
            01  13F0              .byte   blank
      000013FE  13F1      .=scantbl+^x2e
            02  13FE              .byte   pt
      00001415  13FF      .=scantbl+^x45
            04  1415              .byte   exp
      00001435  1416      .=scantbl+^x65
            04  1435              .byte   exp
      000014D0  1436      .=$$$t1
                14D0
                14D0
```

```
                                  14D0  3130          .enabl    lsb
                                  14D0  3131
                            C7F0  14D0  3132          .entry    pli$charfixd_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10>
                                  14D2  3133 charfixd:
              6D   F049 CF    9E  14D2  3134          movab     pli$cnvrt_hnd,(fp)        ; conversion condition handler
              5A   FC9F CF    9E  14D7  3135          movab     w^fixdfixd,r10            ; set completion routine address
                      000A    31  14DC  3136          brw       charfix                   ; do the conversion
                                  14DF  3137
                                  14DF  3138
                  5E    10    C2  14DF  3139 5$:      subl2     #16,sp                    ; get space for packed temp
            6E    1F    00    F9  14E2  3140          cvtlp     #0,#31,(sp)               ; set result to zero
                      00A6    31  14E6  3141          brw       70$                       ; continue in common
                                  14E9  3142 charfix:
                        0C    BB  14E9  3143          pushr     #^m<r2,r3>                ; save registers
                        7E    D4  14EB  3144          clrl      -(sp)                     ; initialize scale factor
                  56    50    7D  14ED  3145          movq      r0,r6                     ; copy r0,r1 to r6,r7
                  5E    20    C2  14F0  3146          subl2     #32,sp                    ; get space for leading sep temp
                  58    5E    D0  14F3  3147          movl      sp,r8                     ; copy leading sep addr
            66    57    20    3B  14F6  3148          skpc      #^x20,r7,(r6)             ; skip leading blanks in source
                        E3    13  14FA  3149          beql      5$                        ; if eql, then all blanks, use 0
                  57    50    D0  14FC  3150          movl      r0,r7                     ; update source length
                  56    51    D0  14FF  3151          movl      r1,r6                     ; update source pointer
                  59    07    D0  1502  3152          movl      #<blank+exp+pt>,r9        ; set mask to terminate integer
                      009C    30  1505  3153          bsbw      gen_lead_sep              ; copy integer to lead sep temp
                     20 AE    D4  1508  3154          clrl      32(sp)                    ; set zero scale
                        57    D5  150B  3155          tstl      r7                        ; more characters?
                        24    13  150D  3156          beql      10$                       ; if eql then no
                  2E    66    91  150F  3157          cmpb      (r6),#^a/./               ; was integer finished by a decimal pt?
                        1F    12  1512  3158          bneq      10$                       ; if neq, no
                        56    D6  1514  3159          incl      r6                        ; advance source pointer past dec. pt.
                        57    D7  1516  3160          decl      r7                        ; update source length
  05   FEB2 CF    66    57    2A  1518  3161          scanc     r7,(r6),scantbl,#<blank+exp> ; find end of fraction
                  51    56    C2  151F  3162          subl2     r6,r1                     ; get number of digits in fraction
                  57    51    C2  1522  3163          subl2     r1,r7                     ; subtract from source length
               20 AE    51    D0  1525  3164          movl      r1,32(sp)                 ; save as scale
            68    66    51    28  1529  3165          movc3     r1,(r6),(r8)              ; copy frac to lead sep temp
                  56    51    D0  152D  3166          movl      r1,r6                     ; update source pointer
                  58    53    D0  1530  3167          movl      r3,r8                     ; update dest pointer
                  58    5E    C2  1533  3168 10$:     subl2     sp,r8                     ; get size of leading sep string
                        58    D7  1536  3169          decl      r8                        ;
                  5E    10    C2  1538  3170          subl2     #16,sp                    ; get space for packed temp
   6E    1F    10 AE    58    09  153B  3171          cvtsp     r8,16(sp),#31,(sp)        ; convert leading sep to packed
                        57    D5  1541  3172          tstl      r7                        ; done with source string?
                        4A    13  1543  3173          beql      70$                       ; if eql, yes
               45 8F    66    91  1545  3174          cmpb      (r6),#^a/E/               ; exponent specified?
                        06    13  1549  3175          beql      20$                       ; if eql, yes
               65 8F    66    91  154B  3176          cmpb      (r6),#^a/e/               ; exponent with a small e?
                        38    12  154F  3177          bneq      50$                       ; if neq, no, check rest of source
                        56    D6  1551  3178 20$:     incl      r6                        ; skip past e or E
                        57    D7  1553  3179          decl      r7                        ; update source length
               58    10 AE    9E  1555  3180          movab     16(sp),r8                 ; point to lead sep temp
                        58    DD  1559  3181          pushl     r8                        ; save address
                  59    01    D0  155B  3182          movl      #blank,r9                 ; set mask to terminate exponent
                        44    10  155E  3183          bsbb      gen_lead_sep              ; transfer sign and exponent to lead sep
                  58    6E    C2  1560  3184          subl      (sp),r8                   ; calculate size of lead sep
                        58    D7  1563  3185          decl      r8
   6E    04    14 AE    58    09  1565  3186          cvtsp     r8,20(sp),#4,(sp)         ; convert exponent to packed
```

```
            6E    6E    04    36  156B        cvtpl   #4,(sp),(sp)                ; convert exponent to long
               30 AE    8E    C2  156F        subl    (sp)+,48(sp)                ; subtract exponent from scale
                        14    18  1573        bgeq    50$                         ; if scale geq, we're set
         10 AE    6E    1F    34  1575        movp    #31,(sp),16(sp)             ; copy packed integer and fraction
               50    30 AE    CE  157A        mnegl   48(sp),r0                   ; get negative scale
6E   1F    00    10 AE    1F    50    F8  157E        ashp    r0,#31,16(sp),#0,#31,(sp) ; shift so we have positive scale
                     30 AE    D4  1586        clrl    48(sp)                      ; indicate this in scale factor
            66    57    20    3B  1589  50$:  skpc    #^x20,r7,(r6)               ; skip past blanks
                        12    12  158D        bneq    80$                         ; if blanks don't finish the source
               52    34 AE    7D  158F  70$:  movq    52(sp),r2                   ; get back original destination
                  50    5E    D0  1593        movl    sp,r0                       ; source is packed temp
                  51    1F    D0  1596        movl    #31,r1                      ; precision is max
      51    08    08    30 AE    F0  1599        insv    48(sp),#8,#8,r1             ; add in scale factor
                        6A    17  159F        jmp     (r10)                       ; return
                                  15A1
                      EEB1    31  15A1  80$:  brw     error                       ; continue - no stack cleanup needed
                                  15A4
                                  15A4          .dsabl  lsb
                                  15A4  ; + gen_lead_sep - copy sign and integer from source to destination
                                  15A4  ;
                                  15A4  ; this routine copies an integer from the source string to the destination
                                  15A4  ; string. the destination string will be in leading separate format because
                                  15A4  ; gen_lead_sep will put a + into the first byte of the destination if there is
                                  15A4  ; no explicit sign in the source string. the source string pointer
                                  15A4  ; will be updated to point past the integer. the source string length
                                  15A4  ; will be updated to not include the integer. the destination pointer
                                  15A4  ; will point to the byte after the integer in the destination string.
                                  15A4  ; no checking is done as the the validity of the integer. any leading
                                  15A4  ; blanks should be removed before calling int_sign.
                                  15A4  ;
                                  15A4  ; inputs:
                                  15A4  ;       r6 = address of the source string
                                  15A4  ;       r7 = length of the source string
                                  15A4  ;       r8 = address of the destination string
                                  15A4  ;       r9 = mask to use with scanc to determine end of integer
                                  15A4  ;
                                  15A4  ; outputs:
                                  15A4  ;       r0-r5 destroyed
                                  15A4  ;       r6 = address of the remaining source string
                                  15A4  ;       r7 = length of the remaining source string
                                  15A4  ;       r8 = address of the next free byte in the destination string
                                  15A4  ;       r9-r14 unchanged
                                  15A4  ; -
                                  15A4
                                  15A4  gen_lead_sep:
                  68    2B    90  15A4        movb    #^a/+/,(r8)                 ; plug a + into the destination
                  88    66    91  15A7        cmpb    (r6),(r8)+                  ; was there a + in the source?
                        05    13  15AA        beql    10$                         ; if eql, yes
                  2D    66    91  15AC        cmpb    (r6),#^a/-/                 ; was there a -?
                        06    12  15AF        bneq    20$                         ; if neq       no, default to +
               FF A8    86    90  15B1  10$:  movb    (r6)+,-1(r8)                ; plug the source sign into the dest
                        57    D7  15B5        decl    r7                          ; correct source length
    59    FE13 CF    66    57    2A  15B7  20$:  scanc   r7,(r6),scantbl,r9          ; look for terminator in source
                  51    56    C2  15BE        subl    r6,r1                       ; calculate length for movc
                  57    51    C2  15C1        subl    r1,r7                       ; correct source length
                  1F    51    D1  15C4        cmpl    r1,#31                      ;is this too big?
                        03    1B  15C7        blequ   30$                         ;if lssu , no, cont
```

```
                EE89   31  15C9  3244          brw     error                     ;signal error
 68      66       51   28  15CC  3245  30$:    movc3   r1,(r6),(r8)              ; move the integer
         56       51   D0  15D0  3246          movl    r1,r6                     ; update pointers
         58       53   D0  15D3  3247          movl    r3,r8                     ;
                       05  15D6  3248          rsb                               ; return
                           15D7  3249
                           15D7  3250
```

```
                          15D7          .sbttl  charfltd - character to float decimal conversion
                          15D7   ; ++
                          15D7   ; charfltd - character to float decimal conversion
                          15D7   ;
                          15D7   ; functional description:
                          15D7   ;
                          15D7   ; This routine converts a character value to a float decimal value.
                          15D7   ;
                          15D7   ; inputs:
                          15D7   ;
                          15D7   ;       r0 = address of the source
                          15D7   ;       r1 = size or precision of source
                          15D7   ;       r2 = address of the destination
                          15D7   ;       r3 = size or the precision of the destination
                          15D7   ;
                          15D7   ; outputs:
                          15D7   ;
                          15D7   ;       The destination is filled in
                          15D7   ; --
                    C090  15D7          .entry  pli$charfltd_r6,^m<iv,dv,r4,r7>
                          15D9   charfltd:
 6D    EF42 CF       9E   15D9          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
            F06D     30   15DE          bsbw    dest_fltd_prec          ; get dest context
            FD70     30   15E1          bsbw    cvrt_char_flt           ; continue in common
                     04   15E4          ret
```

```
                         15E5          .sbttl  fchrfltd - fractioned character to float decimal conversion
                         15E5     ; ++
                         15E5     ; fchrfltd - fractioned character to float decimal conversion
                         15E5     ;
                         15E5     ; functional description:
                         15E5     ;
                         15E5     ; This routine converts a character value to a float decimal value. It
                         15E5     ; accepts as input the default number of digits in the fraction, if no
                         15E5     ; decimal point is contained within the character string source. This is
                         15E5     ; currently used only by the e format input routine.
                         15E5     ;
                         15E5     ; inputs:
                         15E5     ;
                         15E5     ;        r0 = address of the source
                         15E5     ;        r1 = size or precision of source
                         15E5     ;        r2 = address of the destination
                         15E5     ;        r3 = size or the precision of the destination
                         15E5     ;        r4 = number of default fractional digits, if decimal point is missing
                         15E5     ;
                         15E5     ; outputs:
                         15E5     ;
                         15E5     ;        The destination is filled in
                         15E5     ; --
                   C090  15E5          .entry  pli$fchrfltd_r6,^m<iv,dv,r4,r7>
  6D    EF34 CF      9E  15E7          movab   pli$cnvrt_hnd,(fp)       ; conversion condition handler
             F05F    30  15EC          bsbw    dest_fltd_prec           ; get dest context
             FD64    30  15EF          bsbw    cvrt_fchr_flt            ; continue in common
                     04  15F2          ret
```

```
                                  15F3        .sbttl  charchar - convert character to character
                                  15F3  ; ++
                                  15F3  ; charchar - convert cha acter to character
                                  15F3  ;
                                  15F3  ; functional description:
                                  15F3  ;
                                  15F3  ; This routine converts character strings to character.
                                  15F3  ;
                                  15F3  ; inputs:
                                  15F3  ;
                                  15F3  ;       r0 = address of the source
                                  15F3  ;       r1 = size or precision of source
                                  15F3  ;       r2 = address of the destination
                                  15F3  ;       r3 = size or the precision of the destination
                                  15F3  ;
                                  15F3  ; outputs:
                                  15F3  ;
                                  15F3  ;       The destination is filled in
                                  15F3  ; --
                            C030  15F3        .entry  pli$charchar_r6,^m<iv,dv,r4,r5>
                                  15F5  charchar:
     62   53   20   60   51   2C  15F5        movc5   r1,(r0),#32,r3,(r2)     ; perform the operation
                              04  15FB        ret
```

```
                                  15FC        .sbttl  charvcha - convert character to character varying
                                  15FC  ; ++
                                  15FC  ; charvcha - character to character varying
                                  15FC  ;
                                  15FC  ; functional description:
                                  15FC  ;
                                  15FC  ; This routine converts character string to character varying.
                                  15FC  ;
                                  15FC  ; inputs:
                                  15FC  ;
                                  15FC  ;       r0 = address of the source
                                  15FC  ;       r1 = size or precision of source
                                  15FC  ;       r2 = address of the destination
                                  15FC  ;       r3 = size or the precision of the destination
                                  15FC  ;
                                  15FC  ; outputs:
                                  15FC  ;
                                  15FC  ;       The destination is filled in
                                  15FC  ; --
                            C030  15FC        .entry  pli$charvcha_r6,^m<iv,dv,r4,r5>
                                  15FE  charvcha:
                   62   51    B0  15FE        movw    r1,(r2)                 ; move size
                   53   51    B1  1601        cmpw    r1,r3                   ; that size fit?
                        03    1B  1604        blequ   10$                     ; if lequ then yes
                   62   53    B0  1606        movw    r3,(r2)                 ; use smaller size
                        82    B5  1609  10$:  tstw    (r2)+                   ; point to string
62    53    20     60   51    2C  160B        movc5   r1,(r0),#32,r3,(r2)     ; move it
                              04  1611        ret
```

```
                                 1612  3360          .sbttl  charbit - convert character to bit
                                 1612  3361  ; ++
                                 1612  3362  ; charabit - character to bit aligned
                                 1612  3363  ; charbit - character to bit conversion
                                 1612  3364  ;
                                 1612  3365  ; functional description:
                                 1612  3366  ;
                                 1612  3367  ; This routine converts character string to a bit string.
                                 1612  3368  ;
                                 1612  3369  ; inputs:
                                 1612  3370  ;
                                 1612  3371  ;       r0 = address of the source
                                 1612  3372  ;       r1 = size or precision of source
                                 1612  3373  ;       r2 = address of the destination
                                 1612  3374  ;       r3 = size or the precision of the destination
                                 1612  3375  ;       r6 = bit offset of the destination
                                 1612  3376  ;
                                 1612  3377  ; outputs:
                                 1612  3378  ;
                                 1612  3379  ;       The destination is filled in
                                 1612  3380  ; --
                           C070  1612  3381          .entry  pli$charabit_r6,^m<iv,dv,r4,r5,r6>
                                 1614  3382  charabit:
         6D   EF07 CF       9E   1614  3383          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                 F958       30   1619  3384          bsbw    clr_abit_trailer        ; clear abit last byte
                   02       11   161C  3385          brb     charbit                 ;
                           C030  161E  3386          .entry  pli$charbit_r6,^m<iv,dv,r4,r5>
                                 1620  3387  charbit:
         6D   EEFB CF       9E   1620  3388          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                                 1625  3389
                 F962       30   1625  3390          bsbw    clr_bit_dest            ; reset bit destination
                   53       D7   1628  3391  10$:    decl    r3                      ; get next bit
                   1A       19   162A  3392          blss    50$                     ; if lss then done
                   51       D7   162C  3393          decl    r1                      ; get next char
                   16       19   162E  3394          blss    50$                     ; if lss then done
              54   80       9A   1630  3395          movzbl  (r0)+,r4                ;
              54   30       82   1633  3396          subb    #^a/0/,r4               ; find bit equiv
                   33       19   1636  3397          blss    70$                     ; if lss then out of range
              01   54       91   1638  3398          cmpb    r4,#1                   ; in range
                   2E       1A   163B  3399          bgtru   70$                     ; if gtru then error
     62   01   56  54       F0   163D  3400          insv    r4,r6,#1,(r2)           ; insert in list
                   56       D6   1642  3401          incl    r6                      ; address next offset
                   E2       11   1644  3402          brb     10$                     ; continue until done
                                 1646  3403
              53   51       D1   1646  3404  50$:    cmpl    r1,r3                   ; see if there's more chars in src
                   1F       15   1649  3405          bleq    60$                     ; if not, br
                   51       D7   164B  3406  55$:    decl    r1                      ; get the remaining chars
                   1B       19   164D  3407          blss    60$                     ;
              54   80       90   164F  3408          movb    (r0)+,r4                ;
              54   20       91   1652  3409          cmpb    #^a/ /,r4               ; see if blank
                   08       12   1655  3410          bneq    56$                     ;
         60   51   20       3B   1657  3411          skpc    #^a/ /,r1,(r0)          ; if blank, then must be all blank
                   0D       13   165B  3412          beql    60$                     ; all done, if all blank
                   0C       11   165D  3413          brb     70$                     ; else, error
              54   30       C2   165F  3414  56$:    subl    #^a/0/,r4               ; see if valid bit char
                   E7       13   1662  3415          beql    55$                     ; if 0, ok
                   54       D7   1664  3416          decl    r4                      ; if 1, ok
```

```
             E3   13  1666  3417          beql    55$                           ;
             01   11  1668  3418          brb     70$                           ; otherwise, error
                  04  166A  3419 60$:     ret
                      166B  3420
           EDE7   31  166B  3421 70$:     brw     error
```

```
                         166E  3423          .sbttl  vchapic - character varying to picture conversion
                         166E  3424  ; ++
                         166E  3425  ; vchapic - character varying to picture conversion
                         166E  3426  ;
                         166E  3427  ; functional description:
                         166E  3428  ;
                         166E  3429  ; This routine converts a character varying string to a picture value.
                         166E  3430  ;
                         166E  3431  ; inputs:
                         166E  3432  ;
                         166E  3433  ;       r0 = address of the source
                         166E  3434  ;       r1 = size or precision of source
                         166E  3435  ;       r2 = address of the destination
                         166E  3436  ;       r3 = size or the precision of the destination
                         166E  3437  ;
                         166E  3438  ; outputs:
                         166E  3439  ;
                         166E  3440  ;       The destination is filled in
                         166E  3441  ; --
                    CFF0 166E  3442          .entry  pli$vchapic_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10,r11>
                         1670  3443  vchapic:
      6D   EEAB CF    9E 1670  3444          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                80    B5 1675  3445          tstw    (r0)+                   ; point to char string
              FC98    31 1677  3446          brw     charpic
```

```
                            167A          .sbttl  vchafixb - character varying to fixed binary conversion
                            167A    ; ++
                            167A    ; vchafixb - character varying to fixed binary conversion
                            167A    ;
                            167A    ; functional description:
                            167A    ;
                            167A    ; This routine converts a character varying string to a fixed binary value.
                            167A    ;
                            167A    ; inputs:
                            167A    ;
                            167A    ;         r0 = address of the source
                            167A    ;         r1 = size or precision of source
                            167A    ;         r2 = address of the destination
                            167A    ;         r3 = size or the precision of the destination
                            167A    ;
                            167A    ; outputs:
                            167A    ;
                            167A    ;         The destination is filled in
                            167A    ; --
                     C7F0   167A          .entry  pli$vchafixb_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10>
                            167C    vchafixb:
    6D    EE9F CF      9E   167C          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                80     B5   1681          tstw    (r0)+                   ; point to character string
              FCB4     31   1683          brw     charfixb                ;
```

```
                               1686  3473          .sbttl  vchafltb - character  arying to floating binary conversion
                               1686  3474 ; ++
                               1686  3475 ; vchafltb - character varying to floating binary conversion
                               1686  3476 ;
                               1686  3477 ; functional description:
                               1686  3478 ;
                               1686  3479 ; This routine converts a character varying string to a floating binary value.
                               1686  3480 ;
                               1686  3481 ; inputs:
                               1686  3482 ;
                               1686  3483 ;       r0 = address of the source
                               1686  3484 ;       r1 = size or precision of source
                               1686  3485 ;       r2 = address of the destination
                               1686  3486 ;       r3 = size or the precision of the destination
                               1686  3487 ;
                               1686  3488 ; outputs:
                               1686  3489 ;
                               1686  3490 ;       The destination is filled in
                               1686  3491 ; --
                         C090  1686  3492          .entry  pli$vchafltb_r6,^m<iv,dv,r4,r7>
                               1688  3493 vchafltb:
          6D   EE93 CF    9E   1688  3494          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
                    80    B5   168D  3495          tstw    (r0)+                   ; point to character string
                  FCB7    31   168F  3496          brw     charfltb                ; do conversion
```

```
                        1692  3498          .sbttl  vchafixd - character varying to fixed decimal conversion
                        1692  3499 ; ++
                        1692  3500 ; vchafixd - character varying to fixed decimal conversion
                        1692  3501 ;
                        1692  3502 ; functional description:
                        1692  3503 ;
                        1692  3504 ; This routine converts a character varying string to a fixed decimal value.
                        1692  3505 ;
                        1692  3506 ; inputs:
                        1692  3507 ;
                        1692  3508 ;       r0 = address of the source
                        1692  3509 ;       r1 = size or precision of source
                        1692  3510 ;       r2 = address of the destination
                        1692  3511 ;       r3 = size or the precision of the destination
                        1692  3512 ;
                        1692  3513 ; outputs:
                        1692  3514 ;
                        1692  3515 ;       The destination is filled in
                        1692  3516 ; --
                  C7F0  1692  3517          .entry  pli$vchafixd_r6,^m<iv,dv,r4,r5,r6,r7,r8,r9,r10>
                        1694  3518 vchafixd:
 6D    EE87 CF     9E   1694  3519          movab   pli$cnvrt_hnd,(fp)       ; conversion condition handler
             80    B5   1699  3520          tstw    (r0)+                    ; skip size of string
           FE34    31   169B  3521          brw     charfixd                 ; convert as character
```

```
                           169E          .sbttl  vchafltd - character varying to float decimal conversion
                           169E  ; ++
                           169E  ; vchafltd - character varying to float decimal conversion
                           169E  ;
                           169E  ; functional description:
                           169E  ;
                           169E  ; This routine converts a character varying value to a float decimal value.
                           169E  ;
                           169E  ; inputs:
                           169E  ;
                           169E  ;         r0 = address of the source
                           169E  ;         r1 = size or precision of source
                           169E  ;         r2 = address of the destination
                           169E  ;         r3 = size or the precision of the destination
                           169E  ;
                           169E  ; outputs:
                           169E  ;
                           169E  ;         The destination is filled in
                           169E  ; --
                     C090  169E          .entry  pli$vchafltd_r6,^m<iv,dv,r4,r7>
                           16A0  vchafltd:
6D    EE7B CF         9E   16A0          movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
            80        B5   16A5          tstw    (r0)+                   ; point to string
          EFA4        30   16A7          bsbw    dest_fltd_prec          ; get dest context
          FCA7        30   16AA          bsbw    cvrt_char_flt           ; continue in common
                      04   16AD          ret
```

```
                                16AE  3550         .sbttl  vchavcha - convert character varying to character varying
                                16AE  3551 ; ++
                                16AE  3552 ; vchavcha - convert character varying to character varying
                                16AE  3553 ;
                                16AE  3554 ; functional description:
                                16AE  3555 ;
                                16AE  3556 ; This routine converts character varying strings to character varying.
                                16AE  3557 ;
                                16AE  3558 ; inputs:
                                16AE  3559 ;
                                16AE  3560 ;       r0 = address of the source
                                16AE  3561 ;       r1 = size or precision of source
                                16AE  3562 ;       r2 = address of the destination
                                16AE  3563 ;       r3 = size or the precision of the destination
                                16AE  3564 ;
                                16AE  3565 ; outputs:
                                16AE  3566 ;
                                16AE  3567 ;       The destination is filled in
                                16AE  3568 ; --
                          C030  16AE  3569         .entry  pli$vchavcha_r6,^m<iv,dv,r4,r5>
                                16B0  3570 vchavcha:
               62    51     B0  16B0  3571         movw    r1,(r2)                 ; insert size
               53    51     B1  16B3  3572         cmpw    r1,r3                   ; room for source
                     03     1B  16B6  3573         blequ   10$                     ; if lequ then yes
               62    53     B0  16B8  3574         movw    r3,(r2)                 ;
               82    80     B1  16BB  3575 10$:    cmpw    (r0)+,(r2)+             ; point to strings
   62    53    20    60    51     2C  16BE  3576         movc5   r1,(r0),#32,r3,(r2)     ; move it
                            04  16C4  3577         ret
```

```
                                 16C5  3579          .sbttl  vchachar - convert character varying to character
                                 16C5  3580 ; ++
                                 16C5  3581 ; vchachar - character varying to character
                                 16C5  3582 ;
                                 16C5  3583 ; functional description:
                                 16C5  3584 ;
                                 16C5  3585 ; This routine converts character varying strings to character.
                                 16C5  3586 ;
                                 16C5  3587 ; inputs:
                                 16C5  3588 ;
                                 16C5  3589 ;       r0 = address of the source
                                 16C5  3590 ;       r1 = size or precision of source
                                 16C5  3591 ;       r2 = address of the destination
                                 16C5  3592 ;       r3 = size or the precision of the destination
                                 16C5  3593 ;
                                 16C5  3594 ; outputs:
                                 16C5  3595 ;
                                 16C5  3596 ;       The destination is filled in
                                 16C5  3597 ; --
                           C030  16C5  3598          .entry  pli$vchachar_r6,^m<iv,dv,r4,r5>
                                 16C7  3599 vchachar:
                       80    B5  16C7  3600          tstw    (r0)+                   ;
     62   53   20   60   51    2C  16C9  3601          movc5   r1,(r0),#32,r3,(r2)     ; move it
                             04  16CF  3602          ret
```

```
                          16D0        .sbttl  vhcabit - character varying to bit string conversion
                          16D0  ; ++
                          16D0  ; vchabit - character varying to bit string conversion
                          16D0  ;
                          16D0  ; functional description:
                          16D0  ;
                          16D0  ; This routine converts a character varying string to a bit string.
                          16D0  ;
                          16D0  ; inputs:
                          16D0  ;
                          16D0  ;       r0 = address of the source
                          16D0  ;       r1 = size or precision of source
                          16D0  ;       r2 = address of the destination
                          16D0  ;       r3 = size or the precision of the destination
                          16D0  ;       r6 = bit offset to destination
                          16D0  ;
                          16D0  ; outputs:
                          16D0  ;
                          16D0  ;       The destination is filled in
                          16D0  ; --
                    C070  16D0        .entry  pli$vchaabit_r6,^m<iv,dv,r4,r5,r6>
                          16D2  vchaabit:
     6D   EE49 CF    9E   16D2        movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
             F89A    30   16D7        bsbw    clr_abit_trailer        ; clear abit last byte
               02    11   16DA        brb     vchabit
                    C030  16DC        .entry  pli$vchabit_r6,^m<iv,dv,r4,r5>
                          16DE  vchabit:
     6D   EE3D CF    9E   16DE        movab   pli$cnvrt_hnd,(fp)      ; conversion condition handler
               80    B5   16E3        tstw    (r0)+                   ;
             FF38    31   16E5        brw     charbit                 ;
```

```
                            16E8        .sbttl  bitpic - bit string to picture conversion
                            16E8   ; ++
                            16E8   ; bitpic - bit string to picture conversion
                            16E8   ;
                            16E8   ; functional description:
                            16E8   ;
                            16E8   ; This routine converts a bit string value to a picture value.
                            16E8   ;
                            16E8   ; inputs:
                            16E8   ;
                            16E8   ;       r0 = address of the source
                            16E8   ;       r1 = size or precision of source
                            16E8   ;       r2 = address of the destination
                            16E8   ;       r3 = size or the precision of the destination
                            16E8   ;       r5 = bit offset to source
                            16E8   ;
                            16E8   ; outputs:
                            16E8   ;
                            16E8   ;       The destination is filled in
                            16E8   ;
                            16E8   ; --
                     C010   16E8        .entry  pli$bitpic_r6,^m<iv,dv,r4>
                            16EA  bitpic:
          5E    10     C2   16EA        subl    #16,sp                          ; alloc packed temp
                52     DD   16ED        pushl   r2                              ; make frame for pic cvrt before regs go awa
        7E   04 A3     9A   16EF        movzbl  pic$b_byte_size(r3),-(sp); frame target size
        52   08 AE     9E   16F3        movab   8(sp),r2                        ; reset dest to temp
                52     DD   16F7        pushl   r2                              ; push it as pic cvrt src
          7E    63     3C   16F9        movzwl  pic$w_pq(r3),-(sp)              ; push target p,q as src p,q
                53     DD   16FC        pushl   r3                              ; pic node addr
          53    63     3C   16FE        movzwl  pic$w_pq(r3),r3                 ; reset dest size as pic p,q
              0067     30   1701        bsbw    cvrt_bit_fixd                   ; conv bit src to fix dec
   00000000'GF  05     FB   1704        calls   #5,g^pli$cvt_to_pic             ; frame all set, cvrt dec to pic
                       04   170B        ret
```

```
                                   170C         .sbttl  bitfixb - bit string to fixed binary conversion
                                   170C   ; ++
                                   170C   ; bitfixb - bit string to fixed binary conversion
                                   170C   ;
                                   170C   ; functional description:
                                   170C   ;
                                   170C   ; This routine converts a bit string value to a fixed binary value.
                                   170C   ;
                                   170C   ; inputs:
                                   170C   ;
                                   170C   ;       r0 = address of the source
                                   170C   ;       r1 = size or precision of source
                                   170C   ;       r2 = address of the destination
                                   170C   ;       r3 = size or the precision of the destination
                                   170C   ;       r5 = bit offset to source
                                   170C   ;
                                   170C   ; outputs:
                                   170C   ;
                                   170C   ;       The destination is filled in
                                   170C   ; --
                             C010  170C         .entry  pli$bitfixb_r6,^m<iv,dv,r4>
                                   170E   bitfixb:
                     01      10    170E         bsbb    cvrt_bits_fixb              ; use common routine
                             04    1710         ret
                                   1711   cvrt_bits_fixb:
                   EEA8      30    1711         bsbw    chk_bit_arith               ; check values
      50    60    51   55    EF    1714         extzv   r5,r1,(r0),r0               ; get bit string
                  55   5E    D0    1719         movl    sp,r5                       ; address a temp
                       7E    D4    171C         clrl    -(sp)                       ;
                                   171E   ;
                  54   50    9A    171E   10$:  movzbl  r0,r4                       ; get low order byte
        75   EBDA  CF44      90    1721         movb    reverse_bit_tbl[r4],-(r5)   ; get reversed byte
       50    50     F8 8F    78    1727         ashl    #-8,r0,r0                   ; shift src down a byte
                       F0    12    172C         bneq    10$
                                   172E   ;
            55    20   51    C3    172E         subl3   r1,#32,r5                   ; adjust for proper prec.
      6E    6E    51   55    EF    1732         extzv   r5,r1,(sp),(sp)             ; move it down
                  50   6E    9E    1737         movab   (sp),r0                     ; address src
                     F42F    30    173A         bsbw    cvrt_fixb_fixb              ; convrt to dest
                       8E    D4    173D         clrl    (sp)+                       ; clean stack
                             05    173F         rsb
```

```
                        1740  3711          .sbttl  bitfltb - bit string to floating binary conversion
                        1740  3712  ; ++
                        1740  3713  ; bitfltb - bit string to floating binary conversion
                        1740  3714  ;
                        1740  3715  ; functional description:
                        1740  3716  ;
                        1740  3717  ; This routine converts a bit string value to a floating binary value.
                        1740  3718  ;
                        1740  3719  ; inputs:
                        1740  3720  ;
                        1740  3721  ;       r0 = address of the source
                        1740  3722  ;       r1 = size or precision of source
                        1740  3723  ;       r2 = address of the destination
                        1740  3724  ;       r3 = size or the precision of the destination
                        1740  3725  ;       r5 = bit offset to source
                        1740  3726  ;
                        1740  3727  ; outputs:
                        1740  3728  ;
                        1740  3729  ;       The destination is filled in
                        1740  3730  ; --
                  C090  1740  3731          .entry  pli$bitfltb_r6,^m<iv,dv,r4,r7>
                        1742  3732 bitfltb:
          EEC7      30  1742  3733          bsbw    dest_fltb_prec          ; get dest context
          0001      30  1745  3734          bsbw    cvrt_bit_flt
                    04  1748  3735          ret
                        1749  3736 cvrt_bit_flt:
      7E    52      7D  1749  3737          movq    r2,-(sp)                ; save dest
      52    7E      DE  174C  3738          moval   -(sp),r2                ; allocate room for a temp
      53    1F      D0  174F  3739          movl    #31,r3                  ; specify max prec
          FFBC      30  1752  3740          bsbw    cvrt_bits_fixb          ; convert source to fixb
      50    5E      D0  1755  3741          movl    sp,r0                   ; temp is now source
      51    1F      D0  1758  3742          movl    #31,r1                  ; with max prec
  52   04 AE        7D  175B  3743          movq    4(sp),r2                ; restore dest
          F4C1      30  175F  3744          bsbw    cvrt_fixb_flt           ; convert temp to fltb
      5E    0C      C0  1762  3745          addl    #12,sp
                    05  1765  3746          rsb
```

```
                       1766          .sbttl  bitfixd - bit string to fixed decimal conversion
                       1766   ; ++
                       1766   ; bitfixd - bit string to fixed decimal conversion
                       1766   ;
                       1766   ; functional description:
                       1766   ;
                       1766   ; This routine converts a bit string value to a fixed decimal value.
                       1766   ;
                       1766   ; inputs:
                       1766   ;
                       1766   ;       r0 = address of the source
                       1766   ;       r1 = size or precision of source
                       1766   ;       r2 = address of the destination
                       1766   ;       r3 = size or the precision of the destination
                       1766   ;       r5 = bit offset to source
                       1766   ;
                       1766   ; outputs:
                       1766   ;
                       1766   ;       The destination is filled in
                       1766   ; --
                C010   1766          .entry  pli$bitfixd_r6,^m<iv,dv,r4>
                       1768 bitfixd:
           01     10   1768          bsbb    cvrt_bit_fixd
                  04   176A          ret
                       176B cvrt_bit_fixd:
           7E     D5   176B          tstl    -(sp)                   ; allocate some room for temp
           0C     BB   176D          pushr   #^m<r2,r3>              ; save real destination
   52   08 AE     DE   176F          moval   8(sp),r2                ; dest addr is on stack above r2,r3
      53   1F     D0   1773          movl    #31,r3                  ; length is max
         FF98     30   1776          bsbw    cvrt_bits_fixb          ; convert to fixb
           0C     BA   1779          popr    #^m<r2,r3>              ; restore dest
      50   5E     D0   177B          movl    sp,r0                   ; specify source is on stack
      51   1F     D0   177E          movl    #31,r1                  ; specify max precision for source
         F593     30   1781          bsbw    cvrt_fixb_fixd          ; convert to fixd
           8E     D5   1784          tstl    (sp)+                   ; clean stack
                  05   1786          rsb
```

```
                    1787  3785          .sbttl  bitfltd - bit to float decimal conversion
                    1787  3786  ; ++
                    1787  3787  ; bitfltd - bit to float decimal conversion
                    1787  3788  ;
                    1787  3789  ; functional description:
                    1787  3790  ;
                    1787  3791  ; This routine converts a bit value to a float decimal value.
                    1787  3792  ;
                    1787  3793  ; inputs:
                    1787  3794  ;
                    1787  3795  ;       r0 = address of the source
                    1787  3796  ;       r1 = size or precision of source
                    1787  3797  ;       r2 = address of the destination
                    1787  3798  ;       r3 = size or the precision of the destination
                    1787  3799  ;
                    1787  3800  ; outputs:
                    1787  3801  ;
                    1787  3802  ;       The destination is filled in
                    1787  3803  ; --
              C090  1787  3804          .entry  pli$bitfltd_r6,^m<iv,dv,r4,r7>
                    1789  3805 bitfltd:
      EEC2   30     1789  3806          bsbw    dest_fltd_prec          ; get dest context
      FFBA   30     178C  3807          bsbw    cvrt_bit_flt            ; cont in common
             04     178F  3808          ret
```

```
                              1790            .sbttl bitchar - bit string to character conversion
                              1790  ; ++
                              1790  ; bitchar - bit string to character conversion
                              1790  ;
                              1790  ; functional description:
                              1790  ;
                              1790  ; This routine converts a bit string value to a character string.
                              1790  ;
                              1790  ; inputs:
                              1790  ;
                              1790  ;       r0 = address of the source
                              1790  ;       r1 = size or precision of source
                              1790  ;       r2 = address of the destination
                              1790  ;       r3 = size or the precision of the destination
                              1790  ;       r5 = bit offset to source
                              1790  ;
                              1790  ; outputs:
                              1790  ;
                              1790  ;       The destination is filled in
                              1790  ; --
                        C190  1790            .entry  pli$bitchar_r6,^m<iv,dv,r4,r7,r8>
                              1792  bitchar:
               58   53    D0  1792            movl    r3,r8                    ;copy dest size
               53   51    D1  1795            cmpl    r1,r3                    ;see if blank fill needed in dest
                    03    18  1798            bgeq    2$                       ;if source geq dest, then no
               53   51    D0  179A            movl    r1,r3                    ;set dest size=source size
               58   53    C2  179D  2$:       subl2   r3,r8                    ;get count for blank fill
                              17A0  ;
                  0026    30  17A0  5$:       bsbw    get_next_32bits          ; get next field
               57   20    D0  17A3            movl    #32,r7                   ; set loop count
                    53    D7  17A6  10$:      decl    r3                       ; count target character position
                    14    19  17A8            blss    20$                      ; if lss then done
               62   30    90  17AA            movb    #^a/0/,(r2)              ; assume zero
               02   54    E9  17AD            blbc    r4,15$                   ; test bit
                    62    96  17B0            incb    (r2)                     ; set to a one
                    52    D6  17B2  15$:      incl    r2                       ; point to next character
     54   54    FF  8F    78  17B4            ashl    #-1,r4,r4                ; adjust value
               EA   57    F5  17B9            sobgtr  r7,10$                   ; continue until done
                    E2    11  17BC            brb     5$                       ; get next field
                              17BE  ;
                    58    D5  17BE  20$:      tstl    r8                       ;see if blank fill needed
                    06    13  17C0            beql    30$                      ;if not, br
62   58   20   60   00    2C  17C2            MOVC5   #0,(R0),#^A/ /,R8,(R2)   ;MOVE IN THE BLANKS
                          04  17C8  30$:      ret
                              17C9
                              17C9  ;
                              17C9  ; get_next_32bits - get next 32 bit field from source bit string
                              17C9  ;
                              17C9  ; inputs:
                              17C9  ;
                              17C9  ;       r0 = base address of string
                              17C9  ;       r1 = remaining size
                              17C9  ;       r5 = offset from base to string
                              17C9  ;
                              17C9  ; outputs:
                              17C9  ;
                              17C9  ;       r0,r1,r5 are updated to address then next field
```

```
                         17C9  3867 ;        r4 = value
                         17C9  3868 ;
                         17C9  3869 get_next_32bits:
          54   20   D0   17C9  3870         movl    #32,r4                  ; assume 32 bit return
          54   51   D1   17CC  3871         cmpl    r1,r4                   ; 32 bits remaining?
               05   14   17CF  3872         bgtr    10$                     ; if gtr then yes
          54   51   D0   17D1  3873         movl    r1,r4                   ;
               0B   13   17D4  3874         beql    20$                     ; if eql then done
          51   54   C2   17D6  3875 10$:    subl    r4,r1                   ; remove bits from count
54   60   54   55   EF   17D9  3876         extzv   r5,r4,(r0),r4           ; get the bits
          50   04   C0   17DE  3877         addl    #4,r0                   ; point to next field
                    05   17E1  3878 20$:    rsb                             ;
                         17E2  3879
                         17E2  3880 ;
                         17E2  3881 ; put_next 32 bits - insert next 32 bit field
                         17E2  3882 ;
                         17E2  3883 ; inputs:
                         17E2  3884 ;
                         17E2  3885 ;       r2 = base address of the field
                         17E2  3886 ;       r3 = size remaining
                         17E2  3887 ;       r6 = offset from base to field
                         17E2  3888 ;       r4 = value to insert
                         17E2  3889 ;
                         17E2  3890 ; outputs:
                         17E2  3891 ;
                         17E2  3892 ;       r2,r3,r6 are updated to address then next field
                         17E2  3893 ;
                         17E2  3894 put_next_32bits:
          57   20   D0   17E2  3895         movl    #32,r7                  ; assume 32 bit insert
          57   53   D1   17E5  3896         cmpl    r3,r7                   ; room for 32?
               05   14   17E8  3897         bgtr    10$                     ; if gtr then yes
          57   53   D0   17EA  3898         movl    r3,r7                   ; set low value
               0B   13   17ED  3899         beql    20$                     ; if eql then no room
          53   57   C2   17EF  3900 10$:    subl    r7,r3                   ; remove size
62   57   56   54   F0   17F2  3901         insv    r4,r6,r7,(r2)           ; insert field
          52   04   C0   17F7  3902         addl    #4,r2                   ; point to next field
                    05   17FA  3903 20$:    rsb
```

```
PLI
Sym

$$$
ABI
ABI
ABI
ABI
ABI
ABI
ABI
ABI
ABI
BIT
BIT
BIT
BIT
BIT
BIT
BIT
BIT
BIT
BLA
CAS
CAS
CHA
CHA
CHA
CHA
CHA
CHA
CHA
CHA
CHA
CHA
CHF
CHF
CHK
CHK
CHK
CLR
CLR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
CVR
```

```
                   17FB          .sbttl  bitvcha - bit string to character varying conversion
                   17FB    ; ++
                   17FB    ; bitvcha - bit string to character varying conversion
                   17FB    ;
                   17FB    ; functional description:
                   17FB    ;
                   17FB    ; This routine converts a bit string value to a character varying string.
                   17FB    ;
                   17FB    ; inputs:
                   17FB    ;
                   17FB    ;       r0 = address of the source
                   17FB    ;       r1 = size or precision of source
                   17FB    ;       r2 = address of the destination
                   17FB    ;       r3 = size or the precision of the destination
                   17FB    ;       r5 = bit offset to source
                   17FB    ;
                   17FB    ; outputs:
                   17FB    ;
                   17FB    ;       The destination is filled in
                   17FB    ; --
             C190  17FB          .entry  pli$bitvcha_r6,^m<iv,dv,r4,r7,r8>
                   17FD    bitvcha:
    62   51   B0   17FD          movw    r1,(r2)                 ; insert source size
    53   51   B1   1800          cmpw    r1,r3                   ; enough room for source?
         03   1B   1803          blequ   10$                     ;
    62   53   B0   1805          movw    r3,(r2)                 ; use smaller size
         82   B5   1808    10$:  tstw    (r2)+                   ;
       FF85   31   180A          brw     bitchar                 ;
```

```
PLI
Sym

NO_
OTS
OTS
OTS
OTS
OTS
OTS
PIC
PIC
PIC
PIC
PIC
PIC
PIC
PIC
PIC
PIC
PIC
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
PLI
```

```
                        180D  3934          .sbttl  bitbit - bit string to bit string conversion
                        180D  3935  ; ++
                        180D  3936  ; bitbit - bit string to bit string conversion
                        180D  3937  ;
                        180D  3938  ; functional description:
                        180D  3939  ;
                        180D  3940  ; This routine converts a bit string value to a bit string.
                        180D  3941  ;
                        180D  3942  ; inputs:
                        180D  3943  ;
                        180D  3944  ;       r0 = address of the source
                        180D  3945  ;       r1 = size or precision of source
                        180D  3946  ;       r2 = address of the destination
                        180D  3947  ;       r3 = size or the precision of the destination
                        180D  3948  ;       r5 = bit offset to source
                        180D  3949  ;       r6 = bit offset to the destination
                        180D  3950  ;
                        180D  3951  ; outputs:
                        180D  3952  ;
                        180D  3953  ;       The destination is filled in
                        180D  3954  ; --
                  0090  180D  3955          .entry  pli$bitbit_r6,^m<iv,dv,r4,r7>
                        180F  3956  bitbit:
                        180F  3957
           FFB7     30  180F  3958  10$:    bsbw    get_next_32bits         ; move field
           FFCD     30  1812  3959          bsbw    put_next_32bits         ;
             51     D5  1815  3960          tstl    r1                      ; source remaining?
             F6     12  1817  3961          bneq    10$                     ; if neq then yes
             53     D5  1819  3962          tstl    r3                      ; target remaining?
             F2     12  181B  3963          bneq    10$                     ;
                    04  181D  3964          ret
```

```
PLI
Sym

SIZ
SRC
SRC
SS$
SS$
SS$
SS$
SS$
STK
STK
STK
STK
STK
STK
STK
STK
STK
VCH
VCH
VCH
VCH
VCH
VCH
VCH
VCH
VCH

PSE
---
.
$AB
_PL

Pha
---
Ini
Com
Pas
Sym
Pas
Sym
Pse
Cro
Ass

The
880
The
421
```

```
                    181E        .sbttl  bitabit - bit string to bit aligned conversion
                    181E  ; ++
                    181E  ; bitabit - bit string to bit aligned conversion
                    181E  ;
                    181E  ; functional description:
                    181E  ;
                    181E  ; This routine converts a bit string value to a bit aligned string.
                    181E  ;
                    181E  ; inputs:
                    181E  ;
                    181E  ;         r0 = address of the source
                    181E  ;         r1 = size or precision of source
                    181E  ;         r2 = address of the destination
                    181E  ;         r3 = size or the precision of the destination
                    181E  ;         r5 = bit offset to source
                    181E  ;
                    181E  ; outputs:
                    181E  ;
                    181E  ;         The destination is filled in
                    181E  ; --
              C0D0  181E        .entry  pli$bitabit_r6,^m<iv,dv,r4,r6,r7>
                    1820 bitabit:
        F751   30   1820        bsbw    clr_abit_trailer        ; clear abit last byte
          EA   11   1823        brb     bitbit                  ;
```

PLI
VAX

33

Mac

_$2
_$2
TOT

198

The

MAC

```
                     1825  3991          .sbttl  abitpic - bit aligned to picture conversion
                     1825  3992  ; ++
                     1825  3993  ; abitpic - bit aligned to picture conversion
                     1825  3994  ;
                     1825  3995  ; functional description:
                     1825  3996  ;
                     1825  3997  ; This routine converts a bit aligned string to a picture value.
                     1825  3998  ;
                     1825  3999  ; inputs:
                     1825  4000  ;
                     1825  4001  ;       r0 = address of the source
                     1825  4002  ;       r1 = size or precision of source
                     1825  4003  ;       r2 = address of the destination
                     1825  4004  ;       r3 = size or the precision of the destination
                     1825  4005  ;
                     1825  4006  ; outputs:
                     1825  4007  ;
                     1825  4008  ;       The destination is filled in
                     1825  4009  ; --
               C010  1825  4010          .entry  pli$abitpic_r6,^m<iv,dv,r4>
                     1827  4011  abitpic:
         55     D4   1827  4012          clrl    r5                      ; clr src bit offset
       FEBE     31   1829  4013          brw     bitpic
```

```
                         182C  4015          .sbttl  abitfixb - bit aligned to fixed binary conversion
                         182C  4016 ; ++
                         182C  4017 ; abitfixb - bit aligned to fixed binary conversion
                         182C  4018 ;
                         182C  4019 ; functional description:
                         182C  4020 ;
                         182C  4021 ; This routine converts a bit aligned string to a fixed binary value.
                         182C  4022 ;
                         182C  4023 ; inputs:
                         182C  4024 ;
                         182C  4025 ;        r0 = address of the source
                         182C  4026 ;        r1 = size or precision of source
                         182C  4027 ;        r2 = address of the destination
                         182C  4028 ;        r3 = size or the precision of the destination
                         182C  4029 ;
                         182C  4030 ; outputs:
                         182C  4031 ;
                         182C  4032 ;        The destination is filled in
                         182C  4033 ; --
                   C030  182C  4034          .entry  pli$abitfixb_r6,^m<iv,dv,r4,r5>
                         182E  4035 abitfixb:
              55    D4   182E  4036          clrl    r5                      ; set no source offset
            FEDB    31   1830  4037          brw     bitfixb                 ;
```

```
                    1833           .sbttl  abitfltb - bit aligned to floating binary conversion
                    1833   ; ++
                    1833   ; abitfltb - bit aligned to floating binary conversion
                    1833   ;
                    1833   ; functional description:
                    1833   ;
                    1833   ; This routine converts a bit aligned string to a floating binary value.
                    1833   ;
                    1833   ; inputs:
                    1833   ;
                    1833   ;       r0 = address of the source
                    1833   ;       r1 = size or precision of source
                    1833   ;       r2 = address of the destination
                    1833   ;       r3 = size or the precision of the destination
                    1833   ;
                    1833   ; outputs:
                    1833   ;
                    1833   ;       The destination is filled in
                    1833   ; --
               C0B0 1833           .entry  pli$abitfltb_r6,^m<iv,dv,r4,r5,r7>
                    1835   abitfltb:
          55    D4  1835           clrl    r5                      ; set no source offset
        FF08    31  1837           brw     bitfltb                 ; continue in common
```

```
                  183A          .sbttl  abitfixd - bit aligned to fixed decimal conversion
                  183A  ; ++
                  183A  ; abitfixd - bit aligned to fixed decimal conversion
                  183A  ;
                  183A  ; functional description:
                  183A  ;
                  183A  ; This routine converts a bit aligned string to a fixed decimal value.
                  183A  ;
                  183A  ; inputs:
                  183A  ;
                  183A  ;       r0 = address of the source
                  183A  ;       r1 = size or precision of source
                  183A  ;       r2 = address of the destination
                  183A  ;       r3 = size or the precision of the destination
                  183A  ;
                  183A  ; outputs:
                  183A  ;
                  183A  ;       The destination is filled in
                  183A  ; --
            C030  183A          .entry  pli$abitfixd_r6,^m<iv,dv,r4,r5>
                  183C  abitfixd:
         55   D4  183C          clrl    r5                      ; set no source offset
       FF27   30  183E          bsbw    bitfixd                 ; continue in common
              04  1841          ret
```

```
                1842        .sbttl  abitfltd - bit aligned to float decimal conversion
                1842  ; ++
                1842  ; abitfltd - bit aligned to float decimal conversion
                1842  ;
                1842  ; functional description:
                1842  ;
                1842  ; This routine converts a bit aligned value to a float decimal value.
                1842  ;
                1842  ; inputs:
                1842  ;
                1842  ;       r0 = address of the source
                1842  ;       r1 = size or precision of source
                1842  ;       r2 = address of the destination
                1842  ;       r3 = size or the precision of the destination
                1842  ;
                1842  ; outputs:
                1842  ;
                1842  ;       The destination is filled in
                1842  ; --
          C0B0  1842        .entry  pli$abitfltd_r6,^m<iv,dv,r4,r5,r7>
                1844  abitfltd:
      55    D4  1844        clrl    r5                      ; clr bit offset
    EE05    30  1846        bsbw    dest_fltd_prec          ; get dest context
    FEFD    30  1849        bsbw    cvrt_bit_flt            ; cont in common
            04  184C        ret
```

```
                     184D          .sbttl abitchar - bit aligned to character conversion
                     184D   ; ++
                     184D   ; abitchar - bit aligned to character conversion
                     184D   ;
                     184D   ; functional description:
                     184D   ;
                     184D   ; This routine converts a bit aligned string to a character string.
                     184D   ;
                     184D   ; inputs:
                     184D   ;
                     184D   ;       r0 = address of the source
                     184D   ;       r1 = size or precision of source
                     184D   ;       r2 = address of the destination
                     184D   ;       r3 = size or the precision of the destination
                     184D   ;
                     184D   ; outputs:
                     184D   ;
                     184D   ;       The destination is filled in
                     184D   ; --
               C1B0  184D          .entry  pli$abitchar_r6,^m<iv,dv,r4,r5,r7,r8>
                     184F   abitchar:
          55    D4   184F          clrl    r5                      ; set no source offset
        FF3E    31   1851          brw     bitchar                 ; continue in common
```

```
                        1854        .sbttl  abitvcha - bit aligned to character varying conversion
                        1854  ; ++
                        1854  ; abitvcha - bit aligned to character varying conversion
                        1854  ;
                        1854  ; functional description:
                        1854  ;
                        1854  ; This routine converts a bit aligned string to a character varying string.
                        1854  ;
                        1854  ; inputs:
                        1854  ;
                        1854  ;       r0 = address of the source
                        1854  ;       r1 = size or precision of source
                        1854  ;       r2 = address of the destination
                        1854  ;       r3 = size or the precision of the destination
                        1854  ;
                        1854  ; outputs:
                        1854  ;
                        1854  ;       The destination is filled in
                        1854  ; --
                  C1B0  1854        .entry  pli$abitvcha_r6,^m<iv,dv,r4,r5,r7,r8>
                        1856 abitvcha:
           55     D4    1856        clrl    r5                      ; set no source offset
     62    51     B0    1858        movw    r1,(r2)                 ; assume that source will fit
     53    51     D1    185B        cmpl    r1,r3                   ; fit?
           03     1B    185E        blequ   10$                     ; if lequ then ok
     62    53     B0    1860        movw    r3,(r2)                 ; set max size
           82     B5    1863 10$:   tstw    (r2)+                   ; address string
         FF2A     31    1865        brw     bitchar                 ; continue in common
```

```
                    1868          .sbttl  abitbit - bit aligned to bit string conversion
                    1868   ; ++
                    1868   ; abitbit - bit aligned to bit string conversion
                    1868   ;
                    1868   ; functional description:
                    1868   ;
                    1868   ; This routine converts a bit aligned string to a bit string.
                    1868   ;
                    1868   ; inputs:
                    1868   ;
                    1868   ;      r0 = address of the source
                    1868   ;      r1 = size or precision of source
                    1868   ;      r2 = address of the destination
                    1868   ;      r3 = size or the precision of the destination
                    1868   ;      r6 = bit offset to the destination
                    1868   ;
                    1868   ; outputs:
                    1868   ;
                    1868   ;      The destination is filled in
                    1868   ; --
              C0B0  1868          .entry  pli$abitbit_r6,^m<iv,dv,r4,r5,r7>
                    186A  abitbit:
         55    D4   186A          clrl    r5                    ; set no source offset
       FFA0    31   186C          brw     bitbit                ;
```

```
                 186F          .sbttl  abitabit - bit aligned to bit aligned conversion
                 186F     ; ++
                 186F     ; abitabit - bit aligned to bit aligned convers
                 186F     ;
                 186F     ; functional description:
                 186F     ;
                 186F     ; This routine converts a bit aligned string to a bit aligned string.
                 186F     ;
                 186F     ; inputs:
                 186F     ;
                 186F     ;        r0 = address of the source
                 186F     ;        r1 = size or precision of source
                 186F     ;        r2 = address of the destination
                 186F     ;        r3 = size or the precision of the destination
                 186F     ;        r6 = bit offset to the destination
                 186F     ;
                 186F     ; outputs:
                 186F     ;
                 186F     ;        The destination is filled in
                 186F     ; --
           C0F0  186F          .entry  pli$abitabit_r6,^m<iv,dv,r4,r5,r6,r7>
                 1871   abitabit:
      55     D4  1871          clrl    r5                     ; set no source offset
    F6FE     30  1873          bsbw    clr_abit_trailer       ; clear abit last byte
    FF96     31  1876          brw     bitbit                 ;
                 1879
                 1879          .end
```

```
$$$T1                           = 000014D0 R      02           DAT_K_BIT_ALIGN                 = 0000000E
ABITABIT                          00001871 R      02           DAT_K_FLT_DEC                   = 00000005
ABITBIT                           0000186A R      02           DEST_FLTB_PREC                    0000060C R      02
ABITCHAR                          0000184F R      02           DEST_FLTD_PREC                    0000064E R      02
ABITFIXB                          0000182E R      02           D_POWER_OF_10                     00000000 R      02
ABITFIXD                          0000183C R      02           EDFRAC                            000011A9 R      02
ABITFLTB                          00001835 R      02           EDINT                             000011A2 R      02
ABITFLTD                          00001844 R      02           EDIT_BEG                          0000119E R      02
ABITPIC                           00001827 R      02           EDIT_END                          000011AE R      02
ABITVCHA                          00001856 R      02           EDIT_FRAC                       = 0000000B
BITABIT                           00001820 R      02           EDIT_INT                        = 00000004
BITBIT                            0000180F R      02           EDIT_PT                         = 00000009
BITCHAR                           00001792 R      02           EDPT                              000011A7 R      02
BITFIXB                           0000170E R      02           ERROR                             00000455 R      02
BITFIXD                           00001768 R      02           EXP                             = 00000004
BITFLTB                           00001742 R      02           FIXBABIT                          00000F09 R      02
BITFLTD                           00001789 R      02           FIXBBIT                           00000F10 R      02
BITPIC                            000016EA R      02           FIXBCHAR                          00000DD4 R      02
BITVCHA                           000017FD R      02           FIXBFIXB                          00000B69 R      02
BLANK                           = 00000001                     FIXBFIXD                          00000D14 R      02
CASEBASE                        = 0000047A R      02           FIXBFIXDTEMP                      00000E2A R      02
CASE_ON_TYPE                      00000475 R      02           FIXBFLTB                          00000C1D R      02
CHARABIT                          00001614 R      02           FIXBFLTD                          00000DCB R      02
CHARBIT                           00001620 R      02           FIXBPIC                           00000B45 R      02
CHARCHAR                          000015F5 R      02           FIXBVCHA                          00000ED7 R      02
CHARFIX                           000014E9 R      02           FIXDABIT                          00001255 R      02
CHARFIXB                          0000133A R      02           FIXDBIT                           0000125C R      02
CHARFIXD                          000014D2 R      02           FIXDCHAR                          000011B4 R      02
CHARFLTB                          00001349 R      02           FIXDFIXB                          00000FEB R      02
CHARFLTD                          000015D9 R      02           FIXDFIXD                          0000117A R      02
CHARPIC                           00001312 R      02           FIXDFLTB                          00001090 R      02
CHARVCHA                          000015FE R      02           FIXDFLTD                          00001197 R      02
CHF$L_SIGARGLST                 = 00000004                     FIXDPIC                           00000FD5 R      02
CHF$L_SIG_NAME                  = 00000004                     FIXDVCHA                          0000123D R      02
CHK_ABIT_ARITH                    000005BE R      02           FLTBABIT                          00000B16 R      02
CHK_BIT_ARITH                     000005BC R      02           FLTBBIT                           00000B1D R      02
CHK_FIXB_STRING                   00000572 R      02           FLTBCHAR                          00000A0B R      02
CLR_ABIT_TRAILER                  00000F74 R      02           FLTBFIXB                          000007B4 R      02
CLR_BIT_DEST                      00000F8A R      02           FLTBFIXD                          00000933 R      02
CVRT_BITS_FIXB                    00001711 R      02           FLTBFLTB                          000008AC R      02
CVRT_BIT_FIXD                     0000176B R      02           FLTBFLTD                          000009FF R      02
CVRT_BIT_FLT                      00001749 R      02           FLTBPIC                           00000787 R      02
CVRT_CHAR_FLT                     00001354 R      02           FLTBVCHA                          00000AEB R      02
CVRT_FCHR_FLT                     00001356 R      02           FLTDABIT                          0000130A R      02
CVRT_FIXB_BIT                     00000F14 R      02           FLTDBIT                           000012F0 R      02
CVRT_FIXB_CHAR                    00000DFB R      02           FLTDCHAR                          000012D9 R      02
CVRT_FIXB_FIXB                    00000B6C R      02           FLTDFIXB                          000012AF R      02
CVRT_FIXB_FIXD                    00000D17 R      02           FLTDFIXD                          000012C4 R      02
CVRT_FIXB_FLT                     00000C23 R      02           FLTDFLTB                          000012B8 R      02
CVRT_FIXD_FIXB                    00000FEE R      02           FLTDFLTD                          000012CD R      02
CVRT_FIXD_FLT                     00001096 R      02           FLTDPIC                           000012A6 R      02
CVRT_FLT_CHAR                     00000A2E R      02           FLTDVCHA                          000012E2 R      02
CVRT_FLT_FIXB                     000007BA R      02           GEN_LEAD_SEP                      000015A4 R      02
CVRT_FLT_FIXD                     0000093A R      02           GET_NEXT_32BITS                   000017C9 R      02
CVRT_FLT_FLT                      000008B5 R      02           GET_SRC_FIXPREC                   00000DA5 R      02
CVRT_FLT_PIC                      0000078D R      02           H_POWER_OF_10                     00000100 R      02
CVRT_PIC_FLT                      000006CB R      02           LIB$SIGNAL                        ********   X    02
```

```
NO_INT                            000011AE R      02           PLI$FIXBFLTB_R6                   00000C1B RG     02
OTS$$CVT_D_T_R8                   ********      X 02           PLI$FIXBFLTD_R6                   00000DC9 RG     02
OTS$$CVT_G_T_R8                   ********      X 02           PLI$FIXBPIC_R6                    00000B43 RG     02
OTS$$CVT_H_T_R8                   ********      X 02           PLI$FIXBVCHA_R6                   00000ED5 RG     02
OTS$CVT_T_D                       ********      X 02           PLI$FIXDABIT_R6                   00001253 RG     02
OTS$CVT_T_G                       ********      X 02           PLI$FIXDBIT_R6                    0000125A RG     02
OTS$CVT_T_H                       ********      X 02           PLI$FIXDCHAR_R6                   000011B2 RG     02
PIC$B_BYTE_SIZE                 = 00000004                     PLI$FIXDFIXB_R6                   00000FE9 RG     02
PIC$W_PQ                        = 00000000                     PLI$FIXDFIXD_R6                   00001178 RG     02
PICABIT                           0000075E R      02           PLI$FIXDFLTB_R6                   0000108E RG     02
PICBIT                            00000738 R      02           PLI$FIXDFLTD_R6                   00001195 RG     02
PICCHAR                           00000711 R      02           PLI$FIXDPIC_R6                    00000FD3 RG     02
PICFIXB                           000006A3 R      02           PLI$FIXDVCHA_R6                   0000123B RG     02
PICFIXD                           000006F3 R      02           PLI$FLTBABIT_R6                   00000B14 RG     02
PICFLTB                           000006C5 R      02           PLI$FLTBBIT_R6                    00000B1B RG     02
PICFLTD                           00000709 R      02           PLI$FLTBCHAR_R6                   00000A09 RG     02
PICPIC                            00000671 R      02           PLI$FLTBFIXB_R6                   000007B2 RG     02
PICVCHA                           00C0071E R      02           PLI$FLTBFIXD_R6                   00000931 RG     02
PLI$ABITABIT_R6                   0000186F RG     02           PLI$FLTBFLTB_R6                   000008AA RG     02
PLI$ABITBIT_R6                    00001868 RG     02           PLI$FLTBFLTD_R6                   000C09FD RG     02
PLI$ABITCHAR_R6                   0000184D RG     02           PLI$FLTBPIC_R6                    00000785 RG     02
PLI$ABITFIXB_R6                   0000182C RG     02           PLI$FLTBVCHA_R6                   00000AE9 RG     02
PLI$ABITFIXD_R6                   0000183A RG     02           PLI$FLTDABIT_R6                   00001308 RG     02
PLI$ABITFLTB_R6                   00001833 RG     02           PLI$FLTDBIT_R6                    000012EE RG     02
PLI$ABITFLTD_R6                   00001842 RG     02           PLI$FLTDCHAR_R6                   000012D7 RG     02
PLI$ABITPIC_R6                    00001825 RG     02           PLI$FLTDFIXB_R6                   000012AD RG     02
PLI$ABITVCHA_R6                   00001854 RG     02           PLI$FLTDFIXD_R6                   000012C2 RG     02
PLI$BITABIT_R6                    0C00181E RG     02           PLI$FLTDFLTB_R6                   000012B6 RG     02
PLI$BITBIT_R6                     0000180D RG     02           PLI$FLTDFLTD_R6                   000012CB RG     02
PLI$BITCHAR_R6                    00001790 RG     02           PLI$FLTDPIC_R6                    000012A4 RG     02
PLI$BITFIXB_R6                    0000170C RG     02           PLI$FLTDVCHA_R6                   000012E0 RG     02
PLI$BITFIXD_R6                    00001766 RG     02           PLI$PICABIT_R6                    0000075C RG     02
PLI$BITFLTB_R6                    00001740 RG     02           PLI$PICBIT_R6                     00000736 RG     02
PLI$BITFLTD_R6                    00001787 RG     02           PLI$PICCHAR_R6                    0000070F RG     02
PLI$BITPIC_R6                     000016E8 RG     02           PLI$PICFIXB_R6                    000006A1 RG     02
PLI$BITVCHA_R6                    000017FB RG     02           PLI$PICFIXD_R6                    000006F1 RG     02
PLI$B_PAC_2_POWER_00              ********      X 02           PLI$PICFLTB_R6                    000006C3 RG     02
PLI$CHARABIT_R6                   00001612 RG     02           PLI$PICFLTD_R6                    00000707 RG     02
PLI$CHARBIT_R6                    0000161E RG     02           PLI$PICPIC_R6                     0000066F RG     02
PLI$CHARCHAR_R6                   000015F3 RG     02           PLI$PICVCHA_R6                    0000071C RG     02
PLI$CHARFIXB_R6                   00001338 RG     02           PLI$VCHAABIT_R6                   000016D0 RG     02
PLI$CHARFIXD_R6                   000014D0 RG     02           PLI$VCHABIT_R6                    000016DC RG     02
PLI$CHARFLTB_R6                   00001347 RG     02           PLI$VCHACHAR_R6                   000016C5 RG     02
PLI$CHARFLTD_R6                   000015D7 RG     02           PLI$VCHAFIXB_R6                   0000167A RG     02
PLI$CHARPIC_R6                    00001310 RG     02           PLI$VCHAFIXD_R6                   00001692 RG     02
PLI$CHARVCHA_R6                   000015FC RG     02           PLI$VCHAFLTB_R6                   00001686 RG     02
PLI$CNVRT_HND                     0000051F RG     02           PLI$VCHAFLTD_R6                   0000169E RG     02
PLI$CVRT_ANY                      00000400 RG     02           PLI$VCHAPIC_R6                    0000166E RG     02
PLI$CVRT_CG_R3                    0000046F RG     02           PLI$VCHAVCHA_R6                   000016AE RG     02
PLI$CVT_FR_PIC                    ********      X 02           PLI$_CNVERR                       ********      X 02
PLI$CVT_TO_PIC                    ********      X 02           PLI$_ERROR                        ********      X 02
PLI$FCHRFLTD_R6                   000015E5 RG     02           PSL$M_FU                        = 00000040
PLI$FIXBABIT_R6                   00000F07 RG     02           PSL$M_IV                        = 00000020
PLI$FIXBBIT_R6                    00000F0E RG     02           PT                              = 00000002
PLI$FIXBCHAR_R6                   00000DD2 RG     02           PUT_NEXT_32BITS                   000017E2 R      02
PLI$FIXBFIXB_R6                   00000B67 RG     02           REVERSE_BIT_TBL                   00000300 R      02
PLI$FIXBFIXD_R6                   00000D12 RG     02           SCANTBL                           000013D0 R      02
```

```
SIZ...                         = 00000001
SRC_FLTB_PREC                    000005EB R       02
SRC_FLTD_PREC                    0000062D R       02
SS$_CONTINUE                     ********    X    02
SS$_DECOVF                       ********    X    02
SS$_INTOVF                       ********    X    02
SS$_RESIGNAL                     ********    X    02
SS$_ROPRAND                      ********    X    02
STK_L_AP                         00000008
STK_L_ARG_LIST                   FFFFFFF8
STK_L_CND_HND                    00000000
STK_L_CND_LST                    FFFFFFF4
STK_L_DISPLAY                    FFFFFFFC
STK_L_FP                         0000000C
STK_L_PC                         00000010
STK_L_PSL                        00000004
STK_L_REGS                       00000014
VCHAABIT                         000016D2 R       02
VCHABIT                          000016DE R       02
VCHACHAR                         000016C7 R       02
VCHAFIXB                         0000167C R       02
VCHAFIXD                         00001694 R       02
VCHAFLTB                         00001688 R       02
VCHAFLTD                         000016A0 R       02
VCHAPIC                          00001670 R       02
VCHAVCHA                         000016B0 R       02

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

PSECT name                      Allocation          PSECT No.  Attributes
----------                      ----------          ---------  ----------
.  ABS  .                       00000000  (     0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT NOVEC BYTE
$ABS$                           FFFFFFFC  (     0.)  01 (  1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD    WRT NOVEC BYTE
_PLI$CODE                       00001879  (  6265.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG

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

Phase                    Page faults    CPU Time       Elapsed Time
-----                    -----------    --------       ------------
Initialization                   11    00:00:00.04    00:00:02.07
Command processing               75    00:00:00.48    00:00:05.83
Pass 1                          320    00:00:13.30    00:00:36.84
Symbol table sort                 0    00:00:00.75    00:00:01.44
Pass 2                          404    00:00:07.65    00:00:26.93
Symbol table output               0    00:00:00.18    00:00:01.22
Psect synopsis output             0    00:00:00.01    00:00:00.02
Cross-reference output            0    00:00:00.00    00:00:00.00
Assembler run totals            810    00:00:22.43    00:01:14.35

The working set limit was 1800 pages.
88039 bytes (172 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 352 non-local and 240 local symbols.
4218 source lines were read in Pass 1, producing 268 object records in Pass 2.
```

```
33 pages of virtual memory were used to define 31 macros.

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

Macro library name                       Macros defined
------------------                       --------------
_$255$DUA28:[PLIRTL.OBJ]PLIRTMAC.MLB;1            5
_$255$DUA28:[SYSLIB]STARLET.MLB;2                 6
TOTALS (all libraries)                           11

198 GETS were required to define 11 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=TRACEBACK/LIS=LIS$:PLICONVRT/OBJ=OBJ$:PLICONVRT MSRC$:PLICONVRT/UPDATE=(ENH$:PLICONVRT)+LIB$:PLIRTM
```

0307 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY