```
PPPPPPPPPPPP      LLL                IIIIIIIII    RRRRRRRRRRRR      TTTTTTTTTTTTTTT   LLL
PPPPPPPPPPPPP     LLL                IIIIIIIII    RRRRRRRRRRRRR     TTTTTTTTTTTTTTT   LLL
PPPPPPPPPPPPP     LLL                IIIIIIIII    RRRRRRRRRRRRR     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
PPPPPPPPPPPPP     LLL                   III       RRRRRRRRRRRRR         TTT           LLL
PPPPPPPPPPPPP     LLL                   III       RRRRRRRRRRRRR         TTT           LLL
PPPPPPPPPPPPP     LLL                   III       RRRRRRRRRRRRR         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
```

FILEID**PLIBIT

```
PPPPPPPP    LL            IIIIII    BBBBBBBB     IIIIII    TTTTTTTTTT
PPPPPPPP    LL            IIIIII    BBBBBBBB     IIIIII    TTTTTTTTTT
PP      PP  LL              II      BB      BB     II          TT
PP      PP  LL              II      BB      BB     II          TT
PP      PP  LL              II      BB      BB     II          TT
PP      PP  LL              II      BB      BB     II          TT
PPPPPPPP    LL              II      BBBBBBBB       II          TT
PPPPPPPP    LL              II      BBBBBBBB       II          TT
PP          LL              II      BB      BB     II          TT
PP          LL              II      BB      BB     II          TT
PP          LL              II      BB      BB     II          TT          ....
PP          LL              II      BB      BB     II          TT          ....
PP          LLLLLLLLLL    IIIIII    BBBBBBBB     IIIIII        TT          ....
PP          LLLLLLLLLL    IIIIII    BBBBBBBB     IIIIII        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
```

Table of contents

```
0000     1          .title pli$bit - pl1 runtime bit manipulation subroutines
0000     2          .ident  /1-003/                                         ; Edit DSB1003
0000     3  ;
0000     4  ;****************************************************************************
0000     5  ;*                                                                          *
0000     6  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     7  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000     8  ;*  ALL RIGHTS RESERVED.                                                    *
0000     9  ;*                                                                          *
0000    10  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    11  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    12  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    13  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    14  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    15  ;*  TRANSFERRED.                                                            *
0000    16  ;*                                                                          *
0000    17  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    18  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    19  ;*  CORPORATION.                                                            *
0000    20  ;*                                                                          *
0000    21  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    22  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    23  ;*                                                                          *
0000    24  ;*                                                                          *
0000    25  ;****************************************************************************
0000    26  ;
0000    27  ;++
0000    28  ; facility:
0000    29  ;
0000    30  ;       VAX/VMS PL1 runtime library.
0000    31  ;
0000    32  ; abstract:
0000    33  ;
0000    34  ;       Runtime routines to manipulate bits.
0000    35  ;
0000    36  ; author: R. Heinen     15-dec-1978
0000    37  ;
0000    38  ;
0000    39  ; Modifications:
0000    40  ;
0000    41  ;
0000    42  ;       1-002   Bill Matthews   29-September-1982
0000    43  ;
0000    44  ;               Invoke macros $defdat and rtshare instead of $defopr and share.
0000    45  ;
0000    46  ;       1-003   Dave Blickstein 19-June-1984
0000    47  ;
0000    48  ;               PLI$INDEXBIT was failing to match certain substrings larger
0000    49  ;               than 32 bits.  It looking at memory locations after the search
0000    50  ;               string because it wasn't updating the "bits left" count
0000    51  ;               correctly.  SPR:  #11-68090.  Test program: S68090.
0000    52  ;--
0000    53
0000    54  ;
0000    55  ; external definitions
0000    56  ;
0000    57
```

```
0000     58          $defdat                         ; define data types
0000     59
0000     60 ;
0000     61 ; local data
0000     62 ;
0000     63
0000     64          rtshare
```

```
                     0000     66          .sbttl common subroutines
                     0000     67 ;++
                     0000     68 ; get_op1_32bits - get next 32 bit field
                     0000     69 ;
                     0000     70 ; functional description:
                     0000     71 ;
                     0000     72 ; This subroutine returns the next 32 bit field from a bit descriptor.
                     0000     73 ; The bit descriptors are updated to address then next field.
                     0000     74 ; If the field is empty then a zero is returned.
                     0000     75 ;
                     0000     76 ; inputs:
                     0000     77 ;
                     0000     78 ;       r2 = size remaining in the field
                     0000     79 ;       r3 = base address of the field
                     0000     80 ;       r4 = offset to the field
                     0000     81 ;
                     0000     82 ; outputs:
                     0000     83 ;
                     0000     84 ;       r0 = value
                     0000     85 ;       r2 = size remaining in the field
                     0000     86 ;       r3 = base address of the field
                     0000     87 ;       r4 = offset to field
                     0000     88 ;--
                     0000     89 get_op1_32bits:
          50   20   D0  0000     90          movl    #32,r0                  ; assume 32 bit return
          52   50   D1  0003     91          cmpl    r0,r2                   ; enough to return 32?
               03   15  0006     92          bleq    10$                     ; if leq then yes
          50   52   D0  0008     93          movl    r2,r0                   ; return remainder
          52   50   C2  000B     94 10$:     subl    r0,r2                   ; remove taken bits
50   63   50   54   EF  000E     95          extzv   r4,r0,(r3),r0           ; get value
          53   04   C0  0013     96          addl    #4,r3                   ; point to next field *** don't access ***
               05   0016     97          rsb                             ; return
```

```
                                0017    99 ;++
                                0017   100 ; get_op2_32bits - get next 32 bit field
                                0017   101 ;
                                0017   102 ; functional description:
                                0017   103 ;
                                0017   104 ; This subroutine returns the next 32 bit field from a bit descriptor.
                                0017   105 ; The bit descriptors are updated to address then next field.
                                0017   106 ; If the field is empty then a zero is returned.
                                0017   107 ;
                                0017   108 ; inputs:
                                0017   109 ;
                                0017   110 ;       r5 = size remaining in the field
                                0017   111 ;       r6 = base address of the field
                                0017   112 ;       r7 = offset to field
                                0017   113 ;
                                0017   114 ; outputs:
                                0017   115 ;
                                0017   116 ;       r0 = value
                                0017   117 ;       r5 = size remaining in the field
                                0017   118 ;       r6 = base address of the field
                                0017   119 ;       r7 = offset to the field
                                0017   120 ;--
                                0017   121 get_op2_32bits:
                     50   20   D0  0017   122         movl    #32,r0                  ; assume 32 bit return
                     55   50   D1  001A   123         cmpl    r0,r5                   ; enough to return 32?
                          03   15  001D   124         bleq    10$                     ; if leq then yes
                     50   55   D0  001F   125         movl    r5,r0                   ; return remainder
                     55   50   C2  0022   126 10$:    subl    r0,r5                   ; remove taken bits
           50   66   50   57   EF  0025   127         extzv   r7,r0,(r6),r0           ; get value
                     56   04   C0  002A   128         addl    #4,r6                   ; point to next field *** don't access ***
                                05  002D   129         rsb                             ; return
```

```
                     002E   131 ;++
                     002E   132 ; set_opnd_1 - set up operand one
                     002E   133 ;
                     002E   134 ; functional description:
                     002E   135 ;
                     002E   136 ; This routine gathers the first operand data into registers r2-r4
                     002E   137 ;
                     002E   138 ; inputs:
                     002E   139 ;
                     002E   140 ;       common stack frame
                     002E   141 ;
                     002E   142 ; outputs:
                     002E   143 ;
                     002E   144 ;       r2 = size of operand 1
                     002E   145 ;       r3 = address of the operand
                     002E   146 ;       r4 = offset to field
                     002E   147 ;--
                     002E   148 set_opnd_1:
53   04 AC   D0      002E   149         movl    4(ap),r3                ; get address of source 1
52   08 AC   D0      0032   150         movl    8(ap),r2                ; fetch dope address
         54   D4     0036   151         clrl    r4                      ; assume not bit
    0C   82   B1     0038   152         cmpw    (r2)+,#dat_k_bit        ; bit type?
         03   12     003B   153         bneq    10$                     ; if neq then no
    53   63   7D     003D   154         movq    (r3),r3                 ; load address and offset
    52   62   3C     0040   155 10$:     movzwl  (r2),r2                 ; get size
              05     0043   156         rsb
```

```
                     0044   158 ;++
                     0044   159 ; set_opnd_2 - set up operand two
                     0044   160 ;
                     0044   161 ; functional description:
                     0044   162 ;
                     0044   163 ; This routine gathers the second operand data into registers r5-r7
                     0044   164 ;
                     0044   165 ; inputs:
                     0044   166 ;
                     0044   167 ;       common stack frame
                     0044   168 ;
                     0044   169 ; outputs:
                     0044   170 ;
                     0044   171 ;       r5 = size of operand 2
                     0044   172 ;       r6 = address of the operand
                     0044   173 ;       r7 = offset to field
                     0044   174 ;--
                     0044   175 set_opnd_2:
56   0C AC   D0      0044   176         movl    12(ap),r6               ; get address
55   10 AC   D0      0048   177         movl    16(ap),r5               ; fetch dope address
        57   D4      004C   178         clrl    r7                      ; assume not bit
   0C   85   B1      004E   179         cmpw    (r5)+,#dat_k_bit        ; bit type?
        03   12      0051   180         bneq    10$                     ; if neq then no
   56   66   7D      0053   181         movq    (r6),r6                 ; load address and offset
   55   65   3C      0056   182 10$:     movzwl  (r5),r5                 ; get size
             05      0059   183         rsb
```

```
                     005A   185 ;++
                     005A   186 ; set_result - setup result descriptors
                     005A   187 ;
                     005A   188 ; functional descriptor:
                     005A   189 ;
                     005A   190 ; This routine calculates the size of the result field based on the
                     005A   191 ; sizes of the inputs.
                     005A   192 ; The offset returned is always zero.
                     005A   193 ;
                     005A   194 ; inputs:
                     005A   195 ;
                     005A   196 ;       r1 = address of the result string
                     005A   197 ;       r2 = size of source one
                     005A   198 ;       r5 = size of source two
                     005A   199 ;
                     005A   200 ; outputs:
                     005A   201 ;
                     005A   202 ;       r1,r2,r3,r4,r5,r6,r7 are saved.
                     005A   203 ;
                     005A   204 ;       r8 = size of output
                     005A   205 ;       r9 = offset (0)
                     005A   206 ;--
                     005A   207 set_result:
            58   52  D0 005A   208         movl    r2,r8                   ; assume source 1 dictates size
            55   52  D1 005D   209         cmpl    r2,r5                   ; actually true?
                 03  14 0060   210         bgtr    10$                     ; if gtr then true
            58   55  D0 0062   211         movl    r5,r8                   ; if not then source 2 does
      59    58   01  C3 0065   212 10$:    subl3   #1,r8,r9                ; find the last byte boundary
            59   07  CA 0069   213         bicl2   #7,r9
61    08    59   00  F0 006C   214         insv    #0,r9,#8,(r1)           ; clear the byte
            59   D4  0071   215         clrl    r9                      ; initialize offset
                 05  0073   216         rsb
```

```
                          0074   218 ;++
                          0074   219 ; put_bits - output bits to result
                          0074   220 ;
                          0074   221 ; functional description:
                          0074   222 ;
                          0074   223 ; This routine inserts a value in the result string and then updates
                          0074   224 ; the result string pointers.  If the result is filled then a ret is done
                          0074   225 ; to complete the operation.
                          0074   226 ;
                          0074   227 ; inputs:
                          0074   228 ;
                          0074   229 ;       r0 = value to insert
                          0074   230 ;       r1 = address of the result
                          0074   231 ;       r8 = size remaining
                          0074   232 ;       r9 = simple offset value
                          0074   233 ;
                          0074   234 ; outputs:
                          0074   235 ;
                          0074   236 ;       r1,r8 are updated
                          0074   237 ;--
                          0074   238 put_bits:
                    50    DD    0074   239         pushl   r0                      ; save value to insert
               50   20    D0    0076   240         movl    #32,r0                  ; assume 32 bit insert
               20   58    D1    0079   241         cmpl    r8,#32                  ; actually 32 bits left?
                    03    14    007C   242         bgtr    10$                     ; if gtr then yes
               50   58    D0    007E   243         movl    r8,r0                   ; set remainder size
          61   50   59    8E    F0    0081   244 10$:    insv    (sp)+,r9,r0,(r1)        ; insert value
               51   04    C0    0086   245         addl    #4,r1                   ; address next field
               58   50    C2    0089   246         subl    r0,r8                   ; remove stored size
                    01    12    008C   247         bneq    20$                     ; if neq then more to go
                          04    008E   248         ret                             ; operation done
                          05    008F   249 20$:    rsb                             ; return to processor
```

```
                     0090   251          .sbttl pli$andbit - and bit string
                     0090   252 ;++
                     0090   253 ; pli$andbit -  and two bit strings
                     0090   254 ;
                     0090   255 ; functional description:
                     0090   256 ;
                     0090   257 ; This routine performs a logical and operation on two bit strings.
                     0090   258 ;
                     0090   259 ; inputs:
                     0090   260 ;
                     0090   261 ;       r1 = address to return string
                     0090   262 ;               ( the size is always max(length(source1),length(source2)) )
                     0090   263 ;
                     0090   264 ;       0(ap) = 4
                     0090   265 ;       4(ap) = address of source 1
                     0090   266 ;       8(ap) = address of the dope for source 1
                     0090   267 ;       12(ap) = address of source 2
                     0090   268 ;       16(ap) = address of the dope for source 2
                     0090   269 ;
                     0090   270 ; If the type of the operand is bit then the address is the address of
                     0090   271 ; a descriptor having the base address in the first longword and the offset
                     0090   272 ; in the second.
                     0090   273 ;
                     0090   274 ; outputs:
                     0090   275 ;
                     0090   276 ;       result string is filled in
                     0090   277 ;--
           07FC      0090   278          .entry  pli$andbit,^m<r2,r3,r4,r5,r6,r7,r8,r9,r10>
                     0092   279
     FF99    30      0092   280          bsbw    set_opnd_1                      ; setup operands
     FFAC    30      0095   281          bsbw    set_opnd_2                      ;
     FFBF    30      0098   282          bsbw    set_result                      ;
     FF62    30      009B   283 10$:      bsbw    get_op1_32bits                  ;
5A     50    D2      009E   284          mcoml   r0,r10                          ; save value complemented
     FF73    30      00A1   285          bsbw    get_op2_32bits                  ;
50     5A    CA      00A4   286          bicl2   r10,r0                          ; compute value
     FFCA    30      00A7   287          bsbw    put_bits                        ; insert into result
       EF    11      00AA   288          brb     10$                             ; continue
```

```
                                00AC   290          .sbttl pli$boolbit - perform bool bif on bits
                                00AC   291 ;++
                                00AC   292 ; pli$boolbit - perform bool bif on bit strings
                                00AC   293 ;
                                00AC   294 ; functionla description:
                                00AC   295 ;
                                00AC   296 ; This routine performs the bool bif operation on three bit strings.
                                00AC   297 ;
                                00AC   298 ;
                                00AC   299 ; The operation is as follows:
                                00AC   300 ;
                                00AC   301 ;       bool(x,y,z)
                                00AC   302 ;
                                00AC   303 ;       result(i) = z(1) if x(i) = 0 and y(i) = 0
                                00AC   304 ;                   z(2) if x(i) = 0 and y(i) = 1
                                00AC   305 ;                   z(3) if x(i) = 1 and y(i) = 0
                                00AC   306 ;                   z(4) if x(i) = 1 and y(i) = 1
                                00AC   307 ;
                                00AC   308 ; inputs:
                                00AC   309 ;
                                00AC   310 ;       r1 = address to return string
                                00AC   311 ;               ( the size is always max(length(source1),length(source2)) )
                                00AC   312 ;
                                00AC   313 ;       0(ap) = 6
                                00AC   314 ;       4(ap) = address of source 1
                                00AC   315 ;       8(ap) = address of the dope for source 1
                                00AC   316 ;       12(ap) = address of source 2
                                00AC   317 ;       16(ap) = address of the dope for source 2
                                00AC   318 ;       20(ap) = address of source 3
                                00AC   319 ;       24(ap) = address of the dope for source 3
                                00AC   320 ;
                                00AC   321 ; outputs:
                                00AC   322 ;
                                00AC   323 ;       none
                                00AC   324 ;--
                        0FFC    00AC   325          .entry  pli$boolbit,^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
                    FF7D    30  00AE   326          bsbw    set_opnd_1              ; load operand 1
                    FF90    30  00B1   327          bsbw    set_opnd_2              ; load operand 2
                58  18 AC     D0  00B4   328          movl    24(ap),r8               ; address function string dope
50  02 A8  03      00     EF   00B8   329          extzv   #0,#3,2(r8),r0          ; get size up to 4 bits
                    05     13   00BE   330          beql    5$                      ; if eql then set to 4
                04  50     D1   00C0   331          cmpl    r0,#4                   ;
                    03     1B   00C3   332          blequ   10$                     ;
                50  04     D0   00C5   333 5$:      movl    #4,r0                   ;
                5A  14 AC  D0   00C8   334 10$:     movl    20(ap),r10              ; address string
                    5B     D4   00CC   335          clrl    r11                     ; set zero offset
                0C  68     B1   00CE   336          cmpw    (r8),#dat_k_bit         ; bit data type?
                    03     12   00D1   337          bneq    15$                     ; if neq then no
                5A  6A     7D   00D3   338          movq    (r10),r10               ; get address and offset
5A  6A          50  5B     EF   00D6   339 15$:     extzv   r11,r0,(r10),r10        ; get up to 4 bits
                    FF7C   30   00DB   340          bsbw    set_result              ; setup result
                06  5A     D1   00DE   341          cmpl    r10,#6                  ; is this a xor?
                    38     13   00E1   342          beql    100$                    ; if eql then yes
                                00E3   343 ;
                                00E3   344 ; bool operation loop
                                00E3   345 ;
                    FF1A    30  00E3   346 20$:     bsbw    get_op1_32bits          ; get op1 field
```

```
                5B   50    D0   00E6   347           movl    r0,r11                  ; save value
                  FF2B    30   00E9   348           bsbw    get_op2_32bits          ; get op2 field
                    3C    BB   00EC   349           pushr   #^m<r2,r3,r4,r5>        ; save regs
                54   50    D0   00EE   350           movl    r0,r4                   ; save value
                     50    D4   00F1   351           clrl    r0                      ; zero accum
                     53    D4   00F3   352           clrl    r3                      ; setup loop count
          52    5B   01     53   EF   00F5   353 30$:    extzv   r3,#1,r11,r2            ; get op1 bit
          52    52   01     52   78   00FA   354         ashl    #1,r2,r2                ; shift up
          55    54   01     53   EF   00FE   355         extzv   r3,#1,r4,r5             ; get op2 bit
          55    52   C8   0103   356         bisl    r2,r5                   ; merge bits
          55    5A   01     55   EF   0106   357         extzv   r5,#1,r10,r5            ; get the op3 bit value
          50    01   53     55   F0   010B   358         insv    r5,r3,#1,r0             ; accumulate result
                  E1 53     1F   F3   0110   359         aobleq  #31,r3,30$              ; continue for 32 bits
                       3C   BA   0114   360         popr    #^m<r2,r3,r4,r5>        ; restore bits
                  FF5B    30   0116   361         bsbw    put_bits                ;
                    C8    11   0119   362         brb     20$                     ; continue
                               011B   363 ;
                               011B   364 ; simple xor function
                               011B   365 ;
                  FEE2    30   011B   366 100$:   bsbw    get_op1_32bits          ; get op1 field
                     50   DD   011E   367         pushl   r0                      ; save value
                  FEF4    30   0120   368         bsbw    get_op2_32bits          ; get op2 field
                50   8E    CC   0123   369         xorl    (sp)+,r0                ; perform operation
                  FF4B    30   0126   370         bsbw    put_bits                ; insert result field
                    F0    11   0129   371         brb     100$                    ; continue
                               012B   372
```

```
                     012B   374          .sbttl pli$orbit - or bit string
                     012B   375 ;++
                     012B   376 ; pli$orbit -  or two bit strings
                     012B   377 ;
                     012B   378 ; functional description:
                     012B   379 ;
                     012B   380 ; This routine performs a logical or operation on two bit strings.
                     012B   381 ;
                     012B   382 ; inputs:
                     012B   383 ;
                     012B   384 ;       r1 = address to return string
                     012B   385 ;               ( the size is always max(length(source1),length(source2)) )
                     012B   386 ;
                     012B   387 ;       0(ap) = 4
                     012B   388 ;       4(ap) = address of source 1
                     012B   389 ;       8(ap) = address of the dope for source 1
                     012B   390 ;       12(ap) = address of source 2
                     012B   391 ;       16(ap) = address of the dope for source 2
                     012B   392 ;
                     012B   393 ; outputs:
                     012B   394 ;
                     012B   395 ;       result string is filled in
                     012B   396 ;--
             07FC    012B   397          .entry  pli$orbit,^m<r2,r3,r4,r5,r6,r7,r8,r9,r10>
                     012D   398
        FEFE   30    012D   399          bsbw    set_opnd_1              ; setup operands
        FF11   30    0130   400          bsbw    set_opnd_2              ;
        FF24   30    0133   401          bsbw    set_result              ;
        FEC7   30    0136   402 10$:      bsbw    get_op1_32bits          ;
   5A     50   D0    0139   403          movl    r0,r10                  ; save value
        FED8   30    013C   404          bsbw    get_op2_32bits          ;
   50     5A   C8    013F   405          bisl2   r10,r0                  ; compute value
        FF2F   30    0142   406          bsbw    put_bits                ; insert into result
          EF   11    0145   407          brb     10$                     ; continue
```

```
                     0147   409          .sbttl pli$notbit - not bit string
                     0147   410 ;++
                     0147   411 ; pli$notbit -  not a bit string
                     0147   412 ;
                     0147   413 ; functional description:
                     0147   414 ;
                     0147   415 ; This routine performs a logical not operation on a bit string.
                     0147   416 ;
                     0147   417 ; inputs:
                     0147   418 ;
                     0147   419 ;       r1 = address to return string
                     0147   420 ;               ( the size is always max(length(source1),length(source2)) )
                     0147   421 ;
                     0147   422 ;       0(ap) = 2
                     0147   423 ;       4(ap) = address of source 1
                     0147   424 ;       8(ap) = address of the dope for source 1
                     0147   425 ;
                     0147   426 ; outputs:
                     0147   427 ;
                     0147   428 ;       result string is filled in
                     0147   429 ;--
           03FC      0147   430          .entry  pli$notbit,^m<r2,r3,r4,r5,r6,r7,r8,r9>
                     0149   431
      FEE2   30      0149   432          bsbw    set_opnd_1              ; setup operands
        55   D4      014C   433          clrl    r5                      ; set no source two
      FF09   30      014E   434          bsbw    set_result              ;
      FEAC   30      0151   435 10$:     bsbw    get_op1_32bits          ;
50      50   D2      0154   436          mcoml   r0,r0                   ; set value complemented
      FF1A   30      0157   437          bsbw    put_bits                ; insert into result
        F5   11      015A   438          brb     10$                     ; continue
```

```
                     015C   440          .sbttl pli$movbit - move bit string
                     015C   441 ;++
                     015C   442 ; pli$movbit -  move a bit string
                     015C   443 ;
                     015C   444 ; functional description:
                     015C   445 ;
                     015C   446 ; This routine performs a move operation on a bit string.
                     015C   447 ;
                     015C   448 ; inputs:
                     015C   449 ;
                     015C   450 ;       r1 = address to return string - if one arg set
                     015C   451 ;               ( the size is always max(length(source1),length(source2)) )
                     015C   452 ;
                     015C   453 ;       0(ap) = 2 or 4
                     015C   454 ;       4(ap) = address of source 1
                     015C   455 ;       8(ap) = address of the dope for source 1
                     015C   456 ;       12(ap) = address of the target if not bit aligned
                     015C   457 ;       16(ap) = address of the target dope
                     015C   458 ;
                     015C   459 ; outputs:
                     015C   460 ;
                     015C   461 ;       result string is filled in
                     015C   462 ;--
             00FC    015C   463          .entry  pli$movbit,^m<r2,r3,r4,r5,r6,r7>
        FECD   30    015E   464          bsbw    set_opnd_1                  ; setup operands
02      6C   D1     0161   465          cmpl    (ap),#2                     ; bit aligned target?
          0A 12      0164   466          bneq    5$                          ; if neq then no
56      51   D0     0166   467          movl    r1,r6                       ; set up operand address
          57 D4      0169   468          clrl    r7                          ; set no offset
55        52 D0      016B   469          movl    r2,r5                       ; copy op1 size
          09 11      016E   470          brb     15$                         ; continue
           FED1      0170   471 5$:      bsbw    set_opnd_2                  ;
      10 BC   0C     0173   472 10$:     cmpw    #dat_k_bit,a16(ap)          ; bit aligned target?
          0D 13      0177   473          beql    20$                         ; if eql then no
   51     55 07      0179   474 15$:     addl3   #7,r5,r1                    ;
51 51      FD 8F    017D   475          ashl    #-3,r1,r1                   ; calc as byte displ
           FF A641  0182   476          clrb    -1(r6)[r1]                  ; insert 0 in last byte
           FE77     0186   477 20$:     bsbw    get_op1_32bits              ;
        51   20 D0      0189   478          movl    #32,r1                      ; assume max insert
        55   51 D1      018C   479          cmpl    r1,r5                       ; room enough?
             03 15      018F   480          bleq    25$                         ; if leq then yes
        51   55 D0      0191   481          movl    r5,r1                       ; set max size
66 51   57   50 F0      0194   482 25$:     insv    r0,r7,r1,(r6)               ; insert target
        56   04 C0      0199   483          addl    #4,r6                       ; point to next field
        55   51 C2      019C   484          subl    r1,r5                       ; remove moved size
             E5 12      019F   485          bneq    20$                         ; if neq then not done yet
                04      01A1   486          ret                                 ; done
                        01A2   487
```

```
                     01A2   489          .sbttl pli$catbit - concatenate bit string
                     01A2   490 ;++
                     01A2   491 ; pli$catbit -  concatenate two bit strings
                     01A2   492 ;
                     01A2   493 ; functional description:
                     01A2   494 ;
                     01A2   495 ; This routine performs a concatenate operation on two bit strings.
                     01A2   496 ;
                     01A2   497 ; inputs:
                     01A2   498 ;
                     01A2   499 ;       r1 = address to return string
                     01A2   500 ;               ( the size is always max(length(source1),length(source2)) )
                     01A2   501 ;
                     01A2   502 ;       0(ap) = 4
                     01A2   503 ;       4(ap) = address of source 1
                     01A2   504 ;       8(ap) = address of the dope for source 1
                     01A2   505 ;       12(ap) = address of source 2
                     01A2   506 ;       16(ap) = address of the dope for source 2
                     01A2   507 ;
                     01A2   508 ; outputs:
                     01A2   509 ;
                     01A2   510 ;       result string is filled in
                     01A2   511 ;--
             07FC    01A2   512          .entry  pli$catbit,^m<r2,r3,r4,r5,r6,r7,r8,r9,r10>
                     01A4   513
        FE87   30    01A4   514          bsbw    set_opnd_1                  ; set up operands
        FE9A   30    01A7   515          bsbw    set_opnd_2                  ;
        FEAD   30    01AA   516          bsbw    set_result                  ;
5A       52    D0    01AD   517 10$:     movl    r2,r10                      ; copy size of op1 remaining
         15    13    01B0   518          beql    20$                         ; if eql then op1 done
20       52    D1    01B2   519          cmpl    r2,#32                      ; are there 32 bits left in source?
          03   15    01B5   520          bleq    15$                         ; if leq then no
5A       20    D0    01B7   521          movl    #32,r10                     ; assume insert of just what's left
           FE43 30    01BA   522 15$:     bsbw    get_op1_32bits              ;
61  5A   59    50    F0    01BD   523          insv    r0,r9,r10,(r1)              ; insert it
         59    5A    C0    01C2   524          addl    r10,r9                      ; adjust target offset
               E6    11    01C5   525          brb     10$                         ; continue till done with op1
                           01C7   526 ;
                           01C7   527 ; move op2
                           01C7   528 ;
         5A    55    D0    01C7   529 20$:     movl    r5,r10                      ; copy size of op1 remaining
               15    13    01CA   530          beql    30$                         ; if eql then op1 done
         20    55    D1    01CC   531          cmpl    r5,#32                      ; are there 32 bits left in source?
               03    15    01CF   532          bleq    25$                         ; if leq then no
         5A    20    D0    01D1   533          movl    #32,r10                     ; assume insert of just what's left
           FE40 30    01D4   534 25$:     bsbw    get_op2_32bits              ; get string two bits
61  5A   59    50    F0    01D7   535          insv    r0,r9,r10,(r1)              ; insert it
         59    5A    C0    01DC   536          addl    r10,r9                      ; adjust target offset
               E6    11    01DF   537          brb     20$                         ; continue
                     04    01E1   538 30$:     ret                         ; done
```

```
                     01E2   540          .sbttl pli$cmpbit - compare bit strings
                     01E2   541 ;++
                     01E2   542 ; pli$cmpbit - compare bit strings
                     01E2   543 ;
                     01E2   544 ; functional description:
                     01E2   545 ;
                     01E2   546 ; This routine compares two bit strings and returns r0 with the proper
                     01E2   547 ; condition code.
                     01E2   548 ;
                     01E2   549 ; inputs:
                     01E2   550 ;
                     01E2   551 ;       0(ap) = 4
                     01E2   552 ;       4(ap) = address of source 1
                     01E2   553 ;       8(ap) = address of the dope for source 1
                     01E2   554 ;       12(ap) = address of source 2
                     01E2   555 ;       16(ap) = address of the dope for source 2
                     01E2   556 ;
                     01E2   557 ; outputs:
                     01E2   558 ;
                     01E2   559 ;       r0 = if tested with a tst the condition codes are set
                     01E2   560 ;--
             03FC    01E2   561          .entry pli$cmpbit,^m<r2,r3,r4,r5,r6,r7,r8,r9>
         FE47  30    01E4   562          bsbw    set_opnd_1                  ; get op1
         FE5A  30    01E7   563          bsbw    set_opnd_2                  ; get op2
           52  D5    01EA   564 10$:     tstl    r2                          ; op1 exhausted?
           04  12    01EC   565          bneq    15$                         ; if neq then no
           55  D5    01EE   566          tstl    r5                          ; op2 exhausted?
           1E  13    01F0   567          beql    30$                         ; if eql then done
         FE22  30    01F2   568 15$:     bsbw    get_op2_32bits              ; get an op1 field
           50  DD    01F5   569          pushl   r0                          ; save value
         FE06  30    01F7   570          bsbw    get_op1_32bits              ; get an op2 field
     58  50  8E  CD  01FA   571          xorl3   (sp)+,r0,r8                 ; get difference
         EA  13      01FE   572          beql    10$                         ; if eql then continue
 51  58  20  00  EA  0200   573          ffs     #0,#32,r8,r1                ; find the different bit
 50  50  01  51  EF  0205   574          extzv   r1,#1,r0,r0                 ; get answer value
         03  12      020A   575          bneq    20$                         ; if neq then op1 gtr op2
     50  01  CE      020C   576          mnegl   #1,r0                       ; set op1 lss op2
             04      020F   577 20$:     ret
         50  D4      0210   578 30$:     clrl    r0                          ; set eql
             04      0212   579          ret
```

```
                     0213   581          .sbttl pli$indexbit - index built-in function for bit
                     0213   582 ;++
                     0213   583 ; pli$indexbit - perform index built-in function for bit strings
                     0213   584 ;
                     0213   585 ; functional description:
                     0213   586 ;
                     0213   587 ; This routine supports the index built-in function for bit strings.
                     0213   588 ; The action is to search string2 for an occurance of string1 and to return
                     0213   589 ; the offset of the match. If no match occurs then 0 is returned.
                     0213   590 ;
                     0213   591 ; inputs:
                     0213   592 ;
                     0213   593 ;       0(ap) = 4
                     0213   594 ;       4(ap) = address of source 1
                     0213   595 ;       8(ap) = address of the dope for source 1
                     0213   596 ;       12(ap) = address of source 2
                     0213   597 ;       16(ap) = address of the dope for source 2
                     0213   598 ;
                     0213   599 ; outputs:
                     0213   600 ;
                     0213   601 ;       r0 = index value
                     0213   602 ;--
              03FC   0213   603          .entry pli$indexbit,^m<r2,r3,r4,r5,r6,r7,r8,r9>
          FE16  30   0215   604          bsbw    set_opnd_1                      ; setup r2,r3,r4
          FE29  30   0218   605          bsbw    set_opnd_2                      ; setup r5,r6,r7
            55  D5   021B   606          tstl    r5                              ; match string size 0?
            40  13   021D   607          beql    35$                             ; if eql then done
       50   20  D0   021F   608          movl    #32,r0                          ; get up to 32 bits of search string
       55   50  D1   0222   609          cmpl    r0,r5                           ;
            03  15   0225   610          bleq    10$                             ;
       50   55  D0   0227   611          movl    r5,r0                           ;
51     66   50  57  EF 022A   612 10$:    extzv   r7,r0,(r6),r1                   ; extract the value
            58  54  D0 022F   613          movl    r4,r8                           ; save initial offset of searcher
            59  20  D0 0232   614          movl    #32,r9                          ; assume 32 bit search
            59  52  D1 0235   615 15$:    cmpl    r2,r9                           ; 32 bits left?
                03  14 0238   616          bgtr    20$                             ; if gtr then yes
            59  52  D0 023A   617          movl    r2,r9                           ; set size to remainder
            52  55  D1 023D   618 20$:    cmpl    r5,r2                           ; enough search bits to match?
                1D  1A 0240   619          bgtru   35$                             ; if gtr then no - no match
              09 51  E9 0242   620          blbc    r1,25$                          ; select proper search instruction
50     63   59  54  EA 0245   621          ffs     r4,r9,(r3),r0                   ; look for field start
                16  12 024A   622          bneq    40$                             ; if neq then posible match found
                07  11 024C   623          brb     30$                             ; else no match found
50     63   59  54  EB 024E   624 25$:    ffc     r4,r9,(r3),r0                   ; look for field start
                0D  12 0253   625          bneq    40$                             ; if neq then posible match found
            52  59  C2 0255   626 30$:    subl    r9,r2                           ; remove searched size
                05  13 0258   627          beql    35$                             ; if eql then done - no match
            54  59  C0 025A   628          addl    r9,r4                           ; adjust offset for next field
                D6  11 025D   629          brb     15$                             ; continue
                50  D4 025F   630 35$:    clrl    r0                              ; done - no match
                04  04 0261   631          ret
            50  54  C2 0262   632 40$:    subl    r4,r0                           ; calculate skipped portion
            52  50  C2 0265   633          subl    r0,r2                           ; update remaining bits left count
            54  50  C0 0268   634          addl    r0,r4                           ; setup matched offset
            01  55  D1 026B   635          cmpl    r5,#1                           ; simple match?
                33  13 026E   636          beql    100$                            ; if eql then done
            20  55  D1 0270   637          cmpl    r5,#32                          ; easy match?
```

```
                     27   15  0273   638          bleq    75$                     ; if leq then do it
                                 0275   639 ;
                                 0275   640 ; complex match
                                 0275   641 ;
                                 0275   642
                00ED 8F   BB     0275   643          pushr   #^m<r0,r2,r3,r5,r6,r7>
                     52   D5     0279   644 50$:      tstl    r2                      ; string 1 exhausted?
                     08   12     027B   645          bneq    55$                     ; if neq then no
                     55   D5     027D   646          tstl    r5                      ; string 2 exhausted?
                     04   12     027F   647          bneq    55$                     ; if neq then no
                     01   BA     0281   648          popr    #^m<r0>                 ; restore r0 only
                     1E   11     0283   649          brb     100$                    ; done - match found
                   FD78   30     0285   650 55$:      bsbw    get_op1_32bits          ; get next 32 bit field
                     50   DD     0288   651          pushl   r0                      ; save value
                   FD8A   30     028A   652          bsbw    get_op2_32bits          ;
                50   8E   D1     028D   653          cmpl    (sp)+,r0                ; match?
                     E7   13     0290   654          beql    50$                     ; if eql then continue match
                00ED 8F   BA     0292   655          popr    #^m<r0,r2,r3,r5,r6,r7>; no match found
                     54   D6     0296   656 60$:      incl    r4                      ; update new search location
                     52   D7     0298   657          decl    r2                      ; adjust remaining size
                     99   11     029A   658          brb     15$                     ; continue
                                 029C   659 ;
                                 029C   660 ; simple match
                                 029C   661 ;
51   63   55   54   ED   029C   662 75$:      cmpzv   r4,r5,(r3),r1           ; match?
                     F3   12     02A1   663          bneq    60$                     ; if neq then continue search
          50   54   01   C1     02A3   664 100$:     addl3   #1,r4,r0                ; calc relitive bit
               50   58   C2     02A7   665          subl    r8,r0                   ; remove initial offset
                          04     02AA   666          ret
                                 02AB   667          .end
```

```
DAT_K_BIT       =  0000000C
GET_OP1_32BITS     00000000 R      01
GET_OP2_32BITS     00000017 R      01
PLI$ANDBIT         00000090 RG     01
PLI$BOOLBIT        000000AC RG     01
PLI$CATBIT         000001A2 RG     01
PLI$CMPBIT         000001E2 RG     01
PLI$INDEXBIT       00000213 RG     01
PLI$MOVBIT         0000015C RG     01
PLI$NOTBIT         00000147 RG     01
PLI$ORBIT          0000012B RG     01
PUT_BITS           00000074 R      01
SET_OPND_1         0000002E R      01
SET_OPND_2         00000044 R      01
SET_RESULT         0000005A R      01
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
_PLI$CODE	000002AB (683.)	01 (1.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	9	00:00:00.07	00:00:00.79
Command processing	85	00:00:00.55	00:00:02.43
Pass 1	85	00:00:01.79	00:00:06.12
Symbol table sort	0	00:00:00.04	00:00:00.46
Pass 2	112	00:00:01.21	00:00:04.75
Symbol table output	2	00:00:00.02	00:00:00.02
Psect synopsis output	1	00:00:00.01	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	294	00:00:03.70	00:00:14.59

The working set limit was 900 pages.
12088 bytes (24 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 41 non-local and 42 local symbols.
667 source lines were read in Pass 1, producing 31 object records in Pass 2.
3 pages of virtual memory were used to define 2 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[PLIRTL.OBJ]PLIRTMAC.MLB;1	2
_255DUA28:[SYSLIB]STARLET.MLB;2	0
TOTALS (all libraries)	2

32 GETS were required to define 2 macros.

There were no errors, warnings or information messages.

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

PLICASERR
LIS

PLICONDIT
LIS

DATDEF
SDL

FILEDEF
IN

PLIRTDEF
MAR

PLICHRSTR
LIS

PLICHAR
LIS

PLICLOSE
LIS

PLIRTL

PLIFILDSP
IN

PLIBYTSIZ
LIS

PLIRTL
MAP

ENVCODES
IN

PLIBIT
LIS