


```

DDDDDDDD  BBBB8888  GGGGGGGG  EEEEEEEEE  XX  XX  AAAAAA  DDDDDDDD  EEEEEEEEE  PPPPPPPP
DDDDDDDD  BBBB8888  GGGGGGGG  EEEEEEEEE  XX  XX  AAAAAA  DDDDDDDD  EEEEEEEEE  PPPPPPPP
DD  DD  BB  BB  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP  PP
DD  DD  BB  BB  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP  PP
DD  DD  BB  BB  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP  PP
DD  DD  BB  BB  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP  PP
DD  DD  BBBB8888  GG  EE  XX  XX  AA  AA  DD  DD  EEEEEEE  PPPPPPPP
DD  DD  BBBB8888  GG  EE  XX  XX  AA  AA  DD  DD  EEEEEEE  PPPPPPPP
DD  DD  BB  BB  GG  GGGGGG  EE  XX  XX  AAAAAAAAAA  DD  DD  EE  PP
DD  DD  BB  BB  GG  GGGGGG  EE  XX  XX  AAAAAAAAAA  DD  DD  EE  PP
DD  DD  BB  BB  GG  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP
DD  DD  BB  BB  GG  GG  EE  XX  XX  AA  AA  DD  DD  EE  PP
DDDDDDDD  BBBB8888  GGGGGG  EEEEEEEEE  XX  XX  AA  AA  DDDDDDDD  EEEEEEEEE  PP
DDDDDDDD  BBBB8888  GGGGGG  EEEEEEEEE  XX  XX  AA  AA  DDDDDDDD  EEEEEEEEE  PP

```

```

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
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS

```

```
1 0001 0 MODULE DBGEXADep (IDENT = 'V04-000') =
2 0002 1 BEGIN
3 0003 1
4 0004 1
5 0005 1 *****
6 0006 1 *
7 0007 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
8 0008 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
9 0009 1 * ALL RIGHTS RESERVED. *
10 0010 1 *
11 0011 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
12 0012 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
13 0013 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
14 0014 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
15 0015 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
16 0016 1 * TRANSFERRED. *
17 0017 1 *
18 0018 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
19 0019 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
20 0020 1 * CORPORATION. *
21 0021 1 *
22 0022 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
23 0023 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
24 0024 1 *
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1
29 0029 1 ++
30 0030 1 FACILITY:
31 0031 1
32 0032 1     DEBUG
33 0033 1
34 0034 1 ABSTRACT:
35 0035 1
36 0036 1     Parse networks for EXAMINE, DEPOSIT, and EVALUATE.
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1     VAX/VMS
41 0041 1
42 0042 1 AUTHOR:
43 0043 1
44 0044 1     David Plummer
45 0045 1
46 0046 1 CREATION DATE:
47 0047 1
48 0048 1     9-Jul_80
49 0049 1
50 0050 1 VERSION:
51 0051 1
52 0052 1     V02.2-001
53 0053 1
54 0054 1 MODIFIED BY:
55 0055 1     Richard Title 31-Jul-81
56 0056 1
57 0057 1 REVISION HISTORY:
```

:	58	0058	1	:	3.0	31-JUL-81	RT	Added type qualifiers /FLOAT, /D_FLOAT, etc.
:	59	0059	1	:	3.1	23-Jun-83	WC3	Added support for EXAMINE/PAKED, EXAMINE/DATE_TIME
:	60	0060	1	:	3.2	11-Jul-83	WC3	Added support for DEP/PAKED:n
:	61	0061	1	:	3.3	15-Jul-83	WC3	Added support for DEP/DATE_TYPE and Fix EXAMINE/DATE_TIME
:	62	0062	1	:				to use DBGSCVT_DX_DX.
:	63	0063	1	:	3.4	20-JAN-84	BAB	Made DEP/INSTR work right.
:	64	0064	1	:				
:	65	0065	1	:				
:	66	0066	1	:	--			

```

68 0067 1  |
69 0068 1  | TABLE OF CONTENTS:
70 0069 1  |
71 0070 1  |
72 0071 1  | FORWARD ROUTINE
73 0072 1  |     DBG$NPARSE_DEPOSIT,           | Parse network for DEPOSIT
74 0073 1  |     DBG$SET_PAGE_PROT,           | Sets protection for a page list
75 0074 1  |     DBG$NPARSE_EVALUATE,        | Parse network
76 0075 1  |     DBG$NPARSE_EXAMINE,         | Parse network
77 0076 1  |     DBG$NFORMAT_WITH_RADIX: NOVALUE; | Formats a value with radix override
78 0077 1  |
79 0078 1  | ! REQUIRE FILES:
80 0079 1  |
81 0080 1  | REQUIRE 'SRC$:DBGPROLOG.REQ';
82 0214 1  | LIBRARY 'LIB$:DBGGEN.L32';
83 0215 1  |
84 0216 1  | ! EQUATED SYMBOLS
85 0217 1  |
86 0218 1  | LITERAL
87 0219 1  |     DEPOSIT                       = 1,
88 0220 1  |     DEPOSIT_INSTRUCTION          = 2,
89 0221 1  |     DEPOSIT_REGISTER             = 3;
90 0222 1  |
91 0223 1  | LITERAL
92 0224 1  |     EVALUATE                     = 1,           | Plain jane EVALUATE
93 0225 1  |     EVALUATE_ADDR                = 2,           | EVALUATE/ADDRESS
94 0226 1  |     EVALUATE_COND                = 3;           | EVALUATE/CONDITION_VALUE
95 0227 1  |
96 0228 1  | LITERAL
97 0229 1  |     EXAMINE                      = 1,
98 0230 1  |     EXAMINE_INSTRUCTION          = 2,
99 0231 1  |     EXAMINE_REGISTER             = 3,
100 0232 1  |     EXAMINE_SOURCE               = 4,
101 0233 1  |     EXAMINE_CONDITION_VALUE     = 5,
102 0234 1  |     EXAMINE_PSL                  = 6,
103 0235 1  |     EXAMINE_PSW                  = 7;
104 0236 1  |
105 0237 1  | LITERAL
106 0238 1  |     DUMMY_TYPE = dbg$notype,      | Used in type node
107 0239 1  |     MAX_STRING_SIZE = 256;        | Max string size for SYS$GETMSG.
108 0240 1  |
109 0241 1  |
110 0242 1  | ! EXTERNAL REFERENCES:
111 0243 1  |
112 0244 1  | EXTERNAL ROUTINE
113 0245 1  |     DBG$CVT_DX DX: NOVALUE,       | Convert any-any
114 0246 1  |     DBG$DEF_SYM FIND,            | Look up defined symbol
115 0247 1  |     DBG$FLUSHBUF: NOVALUE,       | Initializes a new print line.
116 0248 1  |     DBG$GET_TEMPMEM,            | Allocates command temporary dynamic storage
117 0249 1  |     DBG$INS_DECODE,             | Decodes an instruction
118 0250 1  |     DBG$INS_ENCODE,            | Encodes an instruction
119 0251 1  |     DBG$NCOPY_DESC,            | Copy a descriptor
120 0252 1  |     DBG$NEWLINE: NOVALUE,        | Prints a buffer
121 0253 1  |     DBG$NGET_DEFAULT_TYPE,      | Obtains user set default type
122 0254 1  |     DBG$NGET_LENGTH,           | Obtains a primary's rvalue length
123 0255 1  |     DBG$NGET_LVAL,             | Obtains a primary's lvalue
124 0256 1  |     DBG$NGET_MODE,             | Returns true if SYMBOLS, false otherwise

```

125	0257	1	DBG\$NGET_OVERRIDE_TYPE,	Obtains user set override type
126	0258	1	DBG\$NGET_PAGES,	Obtains an rvalue page list
127	0259	1	DBG\$NGET_POTENTIAL_TYPE : NOVALUE,	Returns potential type and length for '.'
128	0260	1	DBG\$NGET_TYPE,	Obtains type of primary descriptor
129	0261	1	DBG\$NMAKE_ARG_VECT,	Creates a message arguent vector
130	0262	1	DBG\$NMAKE_VAL_DESC,	Constructs a value descriptor
131	0263	1	DBG\$NMATCH,	Matches input against counted strings
132	0264	1	DBG\$NNEXT_WORD,	Obtains next word of input
133	0265	1	DBG\$NOUT_INFO,	Outputs and informational message
134	0266	1	DBG\$NPARSE_ADDRESS,	Interface to Address Expression Interpreter
135	0267	1	DBG\$NPARSE_EXPRESSION,	Interface to Expression Interpreter
136	0268	1	DBG\$NSAVE_INTEGER,	Converts current radix input to integer
137	0269	1	DBG\$NSAVE_LAST_LOC : NOVALUE,	Saves '.'
138	0270	1	DBG\$NSAVE_LAST_VAL : NOVALUE,	Saves '\'
139	0271	1	DBG\$NSET_CAST_TYPLEN : NOVALUE,	Saves pseudo type and length
140	0272	1	DBG\$NSYMBOLIZE,	Symbolizes from a primary descriptor
141	0273	1	DBG\$NSYNTAX_ERROR,	Formats a syntax error
142	0274	1	DBG\$NTYPE_CONV,	Type converter
143	0275	1	DBG\$OUTPUT_PSL : NOVALUE,	Outputs the PSL in special format
144	0276	1	DBG\$PRINT: NOVALUE,	Formats into output buffer
145	0277	1	DBG\$READ_ACCESS,	Probes for read access of virtual address
146	0278	1	DBG\$SET_MOD_LVL,	Sets MODE pointer
147	0279	1	DBG\$SRC_TYPE_PC_SOURCE : NOVALUE,	Implements EX/SOURCE
148	0280	1	DBG\$STA_SETREGISTERS : NOVALUE,	Resets registers context
149	0281	1	DBG\$TRANS_TO_REGNAME,	Translate address to register name
150	0282	1	SYSSGETMSG;	Outputs system-type message.
151	0283	1		
152	0284	1	EXTERNAL	
153	0285	1	DBG\$GB_MOD_PTR : REF VECTOR [,BYTE],	Pointer to MODE structure
154	0286	1	DBG\$GB_RADIX: VECTOR[3,BYTE],	Radix settings
155	0287	1	DBG\$GL_CMND_RADIX,	Holds radix of command
156	0288	1	DBG\$RUNFRAME : BLOCK [,BYTE];	Pointer to current runframe
157	0289	1		

```

159 0290 1 GLOBAL ROUTINE DBG$NPARSE_DEPOSIT (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
160 0291 1
161 0292 1
162 0293 1 ++
163 0294 1 FUNCTIONAL DESCRIPTION:
164 0295 1 This routine comprises the ATN parse network for the DEPOSIT verb.
165 0296 1 During parsing, calls are made to other routines to parse expressions
166 0297 1 and address expressions for inclusion as operands in the command
167 0298 1 execution tree. The completed tree is used as input to the command
168 0299 1 execution network which performs the associated semantic actions.
169 0300 1
170 0301 1 DEPOSITS are affected by command override types, user set override types,
171 0302 1 and user set default types. The strategy of this routine is to construct
172 0303 1 a command execution tree that reflects this situation. Thus, whenever the
173 0304 1 target of the DEPOSIT is described by anything other than a primary
174 0305 1 descriptor, adverb nodes are created to describe the type for the
175 0306 1 target. Primary descriptor entity types are overridden only by command
176 0307 1 and user set override types, - not the default type.
177 0308 1
178 0309 1 When an instruction DEPOSIT is indicated, a kernal DEBUG routine is
179 0310 1 called to obtain the source of the target. This MUST be a quoted string.
180 0311 1 The source for all other DEPOSITS is obtained through the Expression
181 0312 1 Interpreter.
182 0313 1
183 0314 1 FORMAL PARAMETERS:
184 0315 1
185 0316 1 INPUT_DESC - A longword containing the address of a standard ascii
186 0317 1 string descriptor representing the user's input
187 0318 1
188 0319 1 VERB_NODE - A longword containing the address of the verb (head)
189 0320 1 node of the command execution tree
190 0321 1
191 0322 1 MESSAGE_VECT - The address of a longword to contain the address of a
192 0323 1 message argument vector upon detection of errors
193 0324 1
194 0325 1 IMPLICIT INPUTS:
195 0326 1
196 0327 1 NONE
197 0328 1
198 0329 1 IMPLICIT OUTPUTS:
199 0330 1
200 0331 1 On success, the entire command execution tree is constructed.
201 0332 1
202 0333 1 On failure, a message argument vector is constructed and returned.
203 0334 1
204 0335 1 ROUTINE VALUE:
205 0336 1
206 0337 1 An unsigned integer longword completion code
207 0338 1
208 0339 1 COMPLETION CODES:
209 0340 1
210 0341 1 ST$K_SUCCESS (1) - Success. Input parsed and execution tree constructed.
211 0342 1
212 0343 1 ST$K_SEVERE (4) - Failure. Error detected. Message argument vector
213 0344 1 constructed and returned.
214 0345 1
215 0346 1 SIDE EFFECTS:

```

```

216 0347 1 1
217 0348 1 1
218 0349 1 1
219 0350 1 1
220 0351 1 1
221 0352 2 BEGIN
222 0353 2
223 0354 2 MAP
224 0355 2 VERB_NODE : REF dbg$verb_node;
225 0356 2
226 0357 2 BIND
227 0358 2
228 0359 2 ! Strings used at this level of parsing
229 0360 2 !
230 0361 2 DBG$CS_INSTRUCTION = UPLIT BYTE (11, 'INSTRUCTION'),
231 0362 2 DBG$CS_PACKED = UPLIT BYTE (6, 'PACKED'), ! A3.2
232 0363 2 DBG$CS_DATE_TIME = UPLIT BYTE (9, 'DATE_TIME'), ! A3.3
233 0364 2 DBG$CS_ASCII = UPLIT BYTE (5, 'ASCII'),
234 0365 2 DBG$CS_ASCIC = UPLIT BYTE (5, 'ASCIC'),
235 0366 2 DBG$CS_AC = UPLIT BYTE (2, 'AC'),
236 0367 2 DBG$CS_ASCIIW = UPLIT BYTE (5, 'ASCIIW'),
237 0368 2 DBG$CS_AW = UPLIT BYTE (2, 'AW'),
238 0369 2 DBG$CS_ASCIIZ = UPLIT BYTE (5, 'ASCIIZ'),
239 0370 2 DBG$CS_AZ = UPLIT BYTE (2, 'AZ'),
240 0371 2 DBG$CS_ASCID = UPLIT BYTE (5, 'ASCID'),
241 0372 2 DBG$CS_AD = UPLIT BYTE (2, 'AD'),
242 0373 2 DBG$CS_BYTE = UPLIT BYTE (4, 'BYTE'),
243 0374 2 DBG$CS_WORD = UPLIT BYTE (4, 'WORD'),
244 0375 2 DBG$CS_LONGWORD = UPLIT BYTE (8, 'LONGWORD'),
245 0376 2 DBG$CS_QUADWORD = UPLIT BYTE (8, 'QUADWORD'),
246 0377 2 DBG$CS_OCTAWORD = UPLIT BYTE (8, 'OCTAWORD'),
247 0378 2 DBG$CS_FLOAT = UPLIT BYTE (5, 'FLOAT'),
248 0379 2 DBG$CS_F_FLOAT = UPLIT BYTE (7, 'F_FLOAT'),
249 0380 2 DBG$CS_D_FLOAT = UPLIT BYTE (7, 'D_FLOAT'),
250 0381 2 DBG$CS_G_FLOAT = UPLIT BYTE (7, 'G_FLOAT'),
251 0382 2 DBG$CS_H_FLOAT = UPLIT BYTE (7, 'H_FLOAT'),
252 0383 2 DBG$CS_COLON = UPLIT BYTE (1, dbg$k_colon),
253 0384 2 DBG$CS_SLASH = UPLIT BYTE (1, dbg$k_slash),
254 0385 2 DBG$CS_EQUAL_SIGN = UPLIT BYTE (1, dbg$k_equal),
255 0386 2 DBG$CS_CR = UPLIT BYTE (1, dbg$k_car_return);
256 0387
257 0388 LOCAL
258 0389 ADDR_EXP_DESC : REF dbg$aed, ! Address expression descriptor
259 0390 NOUN_NODE : REF dbg$noun_node, ! Object
260 0391 ADVERB_NODE : REF dbg$adverb_node, ! Adverb
261 0392 LINK, ! Pointer
262 0393 INP_RADIX, ! Radix for numeric literals
263 0394 TYPE, ! Type of target
264 0395 LENGTH, ! Length of target
265 0396 ADDRESS : VECTOR [2], ! Address of target
266 0397 STATUS, ! Return status
267 0398 VALPTR : REF dbg$valdesc; ! Value descriptor
268 0399
269 0400 ! Create and link a noun node to contain the target
270 0401 !
271 0402 noun_node = dbg$get_tempmem (dbg$k_noun_node_size);
272 0403 verb_node [dbg$l_verb_object_ptr] = .noun_node;

```

```

: 273 0404
: 274 0405
: 275 0406
: 276 0407
: 277 0408
: 278 0409
: 279 0410
: 280 0411
: 281 0412
: 282 0413
: 283 0414
: 284 0415
: 285 0416
: 286 0417
: 287 0418
: 288 0419
: 289 0420
: 290 0421
: 291 0422
: 292 0423
: 293 0424
: 294 0425
: 295 0426
: 296 0427
: 297 0428
: 298 0429
: 299 0430
: 300 0431
: 301 0432
: 302 0433
: 303 0434
: 304 0435
: 305 0436
: 306 0437
: 307 0438
: 308 0439
: 309 0440
: 310 0441
: 311 0442
: 312 0443
: 313 0444
: 314 0445
: 315 0446
: 316 0447
: 317 0448
: 318 0449
: 319 0450
: 320 0451
: 321 0452
: 322 0453
: 323 0454
: 324 0455
: 325 0456
: 326 0457
: 327 0458
: 328 0459
: 329 0460

```

```

: Initialize target type and length
type = -1;           ! -1 indicates that we have not found an override type
length = 0;

: Accept any command switches. We start out with a verb composite representing
: a plain DEPOSIT. This may change as a result of command or user set overrides.
: We changes the composite to DEPOSIT_INSTRUCTION if we find /INSTRUCTION.
: We do not create an adverb node for radix switches - only types.
verb_node [dbg$b verb_composite] = deposit;
inp_radix = .dbg$gb_radix[dbg$b_radix_input];

WHILE dbg$match (.input_desc, dbg$cs_slash, 1)
DO
  BEGIN
  SELECTONE true
  OF
  SET
    [dbg$match (.input_desc, dbg$cs_instruction, 1)] : !DEPOSIT/INSTRUCION
    BEGIN
      type = dsc$k_dtype_zi;
    END;

    [dbg$match (.input_desc, dbg$cs_byte, 1)] : ! DEPOSIT/BYTE
    BEGIN
      type = dsc$k_dtype_b;
      length = byte_length;
    END;

    [dbg$match (.input_desc, dbg$cs_word, 1)] : ! DEPOSIT/WORD
    BEGIN
      type = dsc$k_dtype_w;
      length = word_length;
    END;

    [dbg$match (.input_desc, dbg$cs_longword, 1)] : ! DEPOSIT/LONGWORD
    BEGIN
      type = dsc$k_dtype_l;
      length = long_length;
    END;

    [dbg$match (.input_desc, dbg$cs_quadword, 1)] : ! DEPOSIT/QUAD
    BEGIN
      type = dsc$k_dtype_q;
      length = 8;
    END;

    [dbg$match (.input_desc, dbg$cs_date_time, 2)] : ! DEPOSIT/DATE_TIME
    BEGIN
      type = dsc$k_dtype_adt;
      length = 8;
    END;
  END

```



```

387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443

```

```

END;
END;
[dbg$nmacth (.input_desc, dbg$cs_ascii, 1)] : ! DEPOSIT/ASCII
BEGIN
! Accept the ':' and integer. If there is no colon, we assume
! a length of 4
IF dbg$nmacth (.input_desc, dbg$cs_colon, 1)
THEN
BEGIN
! We found the colon. Accept the integer
!
IF NOT dbg$nsave_integer (.input_desc, length)
THEN
RETURN sts$k_severe;

! Check to see that the length of the ascii deposit is not 0
IF .length LEQ 0
THEN
BEGIN
OWN num_desc : dbg$stg_desc;

num_desc [dsc$w_length] = 1;
num_desc [dsc$a_pointer] = UPLIT TYPE ('0');
.message_vect = dbg$nmacth_arg_vect (dbg$_invnumber, 1, num_desc);
RETURN sts$k_severe;
END;
END;

type = dsc$k_dtype_t;

END;

[dbg$nmacth (.input_desc, dbg$cs_ascic, 5),
dbg$nmacth (.input_desc, dbg$cs_ac, 2)] :
BEGIN
type = dsc$k_dtype_ac;
END;

[dbg$nmacth (.input_desc, dbg$cs_asciiw, 5),
dbg$nmacth (.input_desc, dbg$cs_aw, 2)] :
BEGIN
type = dsc$k_dtype_vt;
END;

[dbg$nmacth (.input_desc, dbg$cs_asciz, 5),
dbg$nmacth (.input_desc, dbg$cs_az, 2)] :
BEGIN
type = dsc$k_dtype_az;
END;

```

```

! A3.2
! A3.2

```

```

: 444 0575
: 445 0576
: 446 0577
: 447 0578
: 448 0579
: 449 0580
: 450 0581
: 451 0582
: 452 0583
: 453 0584
: 454 0585
: 455 0586
: 456 0587
: 457 0588
: 458 0589
: 459 0590
: 460 0591
: 461 0592
: 462 0593
: 463 0594
: 464 0595
: 465 0596
: 466 0597
: 467 0598
: 468 0599
: 469 0600
: 470 0601
: 471 0602
: 472 0603
: 473 0604
: 474 0605
: 475 0606
: 476 0607
: 477 0608
: 478 0609
: 479 0610
: 480 0611
: 481 0612
: 482 0613
: 483 0614
: 484 0615
: 485 0616
: 486 0617
: 487 0618
: 488 0619
: 489 0620
: 490 0621
: 491 0622
: 492 0623
: 493 0624
: 494 0625
: 495 0626
: 496 0627
: 497 0628
: 498 0629
: 499 0630
: 500 0631

```

```

[dbg$nmatch (.input_desc, dbg$cs_ascid, 5),
 dbg$nmatch (.input_desc, dbg$cs_ad, 2)] :
  BEGIN
    type = dbg$sk_dtype_ad;
  END;

[OTHERWISE] :      ! Some flavor of user error
  BEGIN
    .message_vect =
    ( IF dbg$nmatch (.input_desc, dbg$cs_cr, 1)
      THEN
        dbg$nmake_arg_vect (dbg$_needmore)
      ELSE
        dbg$nsyntax_error (dbg$next_word (.input_desc)));
    RETURN sts$sk_severe;
  END;

  TES;

END;

! Obtain the target of the DEPOSIT in the form of an address expression descriptor
STATUS = DBG$NPARSE_ADDRESS (.INPUT_DESC, NOUN_NODE [DBG$NOUN_VALUE],
                             .INP_RADIX, TOKEN$K_TERM_EQUAL, .MESSAGE_VECT);

! The Address Expression Interpreter MUST give us a warning return to
! indicate the '=' that separates the target from the source.
IF .status NEQ sts$sk_warning
THEN
  BEGIN
    IF .status EQL sts$sk_success ! Missing '='
    THEN
      .message_vect = dbg$nmake_arg_vect (dbg$_needmore);
    RETURN sts$sk_severe;
  END;

! Gobble the '='
IF NOT dbg$nmatch (.input_desc, dbg$cs_equal_sign, 1)
THEN
  BEGIN
    .message_vect =
    ( IF dbg$nmatch (.input_desc, dbg$cs_cr, 1)
      THEN
        dbg$nmake_arg_vect (dbg$_needmore)
      ELSE
        dbg$nsyntax_error (dbg$next_word (.input_desc)));
    RETURN sts$sk_severe;
  END;

! Recover the target address expression descriptor

```

```

501 0632 !
502 0633 addr_exp_desc = .noun_node [dbg$l_noun_value];
503 0634
504 0635
505 0636 ! Create an adverb node which we may use to type the target, but don't link
506 0637 ! it to the verb node until we need to.
507 0638
508 0639 adverb_node = dbg$get_tempmem (dbg$k_adverb_node_size);
509 0640
510 0641
511 0642 ! Type the target based on command overrides, user set overrides, current
512 0643 ! location typing, address expression descriptor contents, or user set defaults.
513 0644 ! Check for command override.
514 0645
515 0646 IF .type NEQ -1
516 0647 THEN
517 0648 BEGIN
518 0649 verb_node [dbg$l_verb_adverb_ptr] = .adverb_node;
519 0650 adverb_node [dbg$b_adverb_literal] = .type;
520 0651 adverb_node [dbg$l_adverb_value] = .length;
521 0652 END
522 0653 ELSE
523 0654 BEGIN
524 0655
525 0656 ! There was no command override. Check for a type/override.
526 0657 !
527 0658 IF dbg$ngget_override_type (type, length)
528 0659 THEN
529 0660 BEGIN
530 0661 verb_node [dbg$l_verb_adverb_ptr] = .adverb_node;
531 0662 adverb_node [dbg$b_adverb_literal] = .type;
532 0663 adverb_node [dbg$l_adverb_value] = .length;
533 0664 END;
534 0665 END;
535 0666
536 0667
537 0668 ! The adverb node has been set up, and the command has been classified as
538 0669 ! either DEPOSIT or DEPOSIT_INSTRUCTION. The target has been placed in the
539 0670 ! first noun node. Now the source must be obtained. We allocate another
540 0671 ! noun node and link it to the first one for this purpose. This second
541 0672 ! noun node will point either to a value descriptor or to a buffer containing
542 0673 ! a potential instruction counted string.
543 0674
544 0675 link = noun_node [dbg$l_noun_link];
545 0676
546 0677 noun_node = dbg$get_tempmem (dbg$k_noun_node_size);
547 0678 .link = .noun_node; ! This links the new noun node
548 0679
549 0680
550 0681 ! For implementation level 3, we always just want to call the
551 0682 ! expression parser.
552 0683
553 0684 IF NOT (STATUS = DBG$NPARSE_EXPRESSION (.INPUT_DESC, .INP_RADIX,
554 0685 NOUN_NODE [DBG$L_NOUN_VALUE],
555 0686 TOKEN$K_TERM_NONE, MESSAGE_VECT,
556 0687 DBG$K_DEPOSIT_VERB))
557 0688 THEN

```

```

0689 BEGIN
0690
0691 ! We do not expect more input after the expression
0692
0693 IF .status EQL sts$warning
0694 THEN
0695     .message_vect = dbg$nsyntax_error (dbg$next_word (.input_desc));
0696 RETURN sts$severe;
0697 END;
0698
0699 ! This assures that the DEPOSIT/INSTR has a quoted string.
0700
0701 IF .type EQL dsc$k_dtype_zi
0702 THEN
0703 BEGIN
0704     valptr = .noun_node [dbg$l_noun_value];
0705     IF .valptr [dbg$b_value_dtype] NEQ dsc$k_dtype_t
0706     THEN
0707         SIGNAL(dbg$_opnotallow, 1, uplit byte(%ascic 'DEPOSIT'));
0708     END;
0709
0710 RETURN sts$success;
0711 END;

```

										.TITLE	DBGEXADep			
										.IDENT	\V04-000\			
										.PSECT	DBG\$PLIT,NOWRT, SHR, PIC,0			
										0B	00000	P.AAA:	.BYTE	11
4E	4F	49	54	43	55	52	54	53	4E	49	00001		.ASCII	\INSTRUCTION\
										06	0000C	P.AAB:	.BYTE	6
										50	0000D		.ASCII	\PACKED\
										09	00013	P.AAC:	.BYTE	9
										44	00014		.ASCII	\DATE_TIME\
45	4D	49	54	5F	45	54	41			05	0001D	P.AAD:	.BYTE	5
										41	0001E		.ASCII	\ASCII\
										05	00023	P.AAE:	.BYTE	5
										41	00024		.ASCII	\ASCIC\
										02	00029	P.AAF:	.BYTE	2
										43	0002A		.ASCII	\AC\
										05	0002C	P.AAG:	.BYTE	5
										57	0002D		.ASCII	\ASCIIW\
										02	00032	P.AAH:	.BYTE	2
										57	00033		.ASCII	\AW\
										05	00035	P.AAI:	.BYTE	5
										5A	00036		.ASCII	\ASCIZ\
										02	0003B	P.AAJ:	.BYTE	2
										5A	0003C		.ASCII	\AZ\
										05	0003E	P.AAK:	.BYTE	5
										44	0003F		.ASCII	\ASCID\
										02	00044	P.AAL:	.BYTE	2
										44	00045		.ASCII	\AD\
										04	00047	P.AAM:	.BYTE	4
										45	00048		.ASCII	\BYTE\
										04	0004C	P.AAN:	.BYTE	4

```

          44 52 4F 57 0004D .ASCII \WORD\
          08 00051 P.AAO: .BYTE 8
44 52 4F 57 47 4E 4F 4C 00052 .ASCII \LONGWORD\
          08 0005A P.AAP: .BYTE 8
44 52 4F 57 44 41 55 51 0005B .ASCII \QUADWORD\
          08 00063 P.AAQ: .BYTE 8
44 52 4F 57 41 54 43 4F 00064 .ASCII \OCTAWORD\
          05 0006C P.AAR: .BYTE 5
          54 41 4F 4C 46 0006D .ASCII \FLOAT\
          07 00072 P.AAS: .BYTE 7
          54 41 4F 4C 46 5F 46 00073 .ASCII \F_FLOAT\
          07 0007A P.AAT: .BYTE 7
          54 41 4F 4C 46 5F 44 0007B .ASCII \D_FLOAT\
          07 00082 P.AAU: .BYTE 7
          54 41 4F 4C 46 5F 47 00083 .ASCII \G_FLOAT\
          07 0008A P.AAV: .BYTE 7
          54 41 4F 4C 46 5F 48 0008B .ASCII \H_FLOAT\
          3A 01 00092 P.AAW: .BYTE 1, 58
          2F 01 00094 P.AAX: .BYTE 1, 47
          3D 01 00096 P.AAY: .BYTE 1, 61
          0D 01 00098 P.AAZ: .BYTE 1, 13
          0009A .BLKB 2
54 49 53 4F 00 00 00 30 0009C P.ABA: .ASCII \0\<0><0><0>
          50 45 44 07 000A0 P.ABB: .ASCII <7>\DEPOSIT\

```

.PSECT DBG\$OWN,NOEXE, PIC,2

00000 NUM_DESC: .BLKB 12

```

DBG$CS_INSTRUCTION= P.AAA
DBG$CS_PACKED= P.AAB
DBG$CS_DATE_TIME= P.AAC
DBG$CS_ASCII= P.AAD
DBG$CS_ASCIC= P.AAE
DBG$CS_AC= P.AAF
DBG$CS_ASCIIW= P.AAG
DBG$CS_AW= P.AAH
DBG$CS_ASCIZ= P.AAI
DBG$CS_AZ= P.AAJ
DBG$CS_ASCID= P.AAK
DBG$CS_AD= P.AAL
DBG$CS_BYTE= P.AAM
DBG$CS_WORD= P.AAN
DBG$CS_LONGWORD= P.AAO
DBG$CS_QUADWORD= P.AAP
DBG$CS_OCTAWORD= P.AAQ
DBG$CS_FLOAT= P.AAR
DBG$CS_F_FLOAT= P.AAS
DBG$CS_D_FLOAT= P.AAT
DBG$CS_G_FLOAT= P.AAU
DBG$CS_H_FLOAT= P.AAV
DBG$CS_COLON= P.AAW
DBG$CS_SLASH= P.AAX
DBG$CS_EQUAL_SIGN= P.AAY
DBG$CS_CR= P.AAZ
.EXTRN DBG$CVT_DX_DX, DBG$DEF_SYM_FIND

```

```
.EXTRN DBG$FLUSHBUF, DBG$GET_TEMP_MEM
.EXTRN DBG$INS_DECODE, DBG$INS_ENCODE
.EXTRN DBG$N_COPY_DESC, DBG$NEWLINE
.EXTRN DBG$N_GET_DEFAULT_TYPE
.EXTRN DBG$N_GET_LENGTH
.EXTRN DBG$N_GET_LVAL, DBG$N_GET_MODE
.EXTRN DBG$N_GET_OVERRIDE_TYPE
.EXTRN DBG$N_GET_PAGES, DBG$N_GET_POTENTIAL_TYPE
.EXTRN DBG$N_GET_TYPE, DBG$N_MAKE_ARG_VECT
.EXTRN DBG$N_MAKE_VAL_DESC
.EXTRN DBG$N_MATCH, DBG$N_NEXT_WORD
.EXTRN DBG$N_OUT_INFO, DBG$N_PARSE_ADDRESS
.EXTRN DBG$N_PARSE_EXPRESSION
.EXTRN DBG$N_SAVE_INTEGER
.EXTRN DBG$N_SAVE_LAST_LOC
.EXTRN DBG$N_SAVE_LAST_VAL
.EXTRN DBG$N_SET_CAST_TYPLEN
.EXTRN DBG$N_SYMBOLIZE, DBG$N_SYNTAX_ERROR
.EXTRN DBG$N_TYPE_CONV, DBG$N_OUTPUT_PSL
.EXTRN DBG$PRINT, DBG$READ_ACCESS
.EXTRN DBG$SET_MOD_LVL
.EXTRN DBG$SRC_TYPE_PC_SOURCE
.EXTRN DBG$STA_SETREGISTERS
.EXTRN DBG$TRANS_TO_REGNAME
.EXTRN SYS$GETMSG, DBG$GB_MOD_PTR
.EXTRN DBG$GB_RADIX, DBG$GL_CMD_RADIX
.EXTRN DBG$RUNFRAME
```

.PSECT DBG\$CODE, NOWRT, SHR, PIC, 0

```
.ENTRY DBG$N_PARSE_DEPOSIT, Save R2,R3,R4,R5,R6,R7,-; 0290
R8,R9,R10,R11
MOVAB NUM_DESC, R11
MOVAB DBG$N_MATCH, R10
MOVAB DBG$CS_COLON, R9
SUBL2 #16, SP
PUSHL #4 0402
CALLS #1, DBG$GET_TEMP_MEM
MOVL R0, NOUN_NODE
MOVL VERB_NODE, R5 0403
MOVL NOUN_NODE, 8(R5)
MNEGL #1, TYPE 0408
CLRL LENGTH 0409
MOVB #1, 1(R5) 0416
MOVZBL DBG$GB_RADIX, INP_RADIX 0417
MOVL INPUT_DESC, R3 0419
PUSHL #1
PUSHAB DBG$CS_SLASH
PUSHL R3
CALLS #3, DBG$N_MATCH
BLBS R0, 2$
BRW 42$
PUSHL #1 0426
PUSHAB DBG$CS_INSTRUCTION
PUSHL R3
CALLS #3, DBG$N_MATCH
CMPL R0, #1
```

OFFC 00000

```
5B 00000000' EF 9E 00002
5A 00000000G 00 9E 00009
59 00000000' EF 9E 00010
5E 10 C2 00017
00000000G 00 04 DD 0001A
57 01 FB 0001C
55 08 AC D0 00026
08 A5 57 D0 0002A
04 AE 01 CE 0002E
01 A5 6E D4 00032
58 00000000G 01 90 00034
53 04 AC D0 00038
01 DD 00043 1$:
02 A9 9F 00045
6A 53 DD 00048
03 03 FB 0004A
50 E8 0004D
027D 31 00050
FF6E 01 DD 00053 2$:
53 C9 9F 00055
53 DD 00059
6A 03 FB 0005B
01 50 D1 0005E
```

04	AE	06	12	00061	BNEQ	3\$			
		16	D0	00063	MOVL	#22, TYPE			0428
		DA	11	00067	BRB	1\$			0422
		01	DD	00069	3\$: PUSHL	#1			0431
		B5	A9	9F	0006B	PUSHAB	DBG\$CS_BYTE		
		53	DD	0006E	PUSHL	R3			
	6A	03	FB	00070	CALLS	#3, DBG\$NMATCH			
	01	50	D1	00073	CMPL	R0, #1			
		09	12	00076	BNEQ	4\$			
04	AE	06	D0	00078	MOVL	#6, TYPE			0433
	6E	01	D0	0007C	MOVL	#1, LENGTH			0434
		C2	11	0007F	BRB	1\$			0422
		01	DD	00081	4\$: PUSHL	#1			0437
		BA	A9	9F	00083	PUSHAB	DBG\$CS_WORD		
		53	DD	00086	PUSHL	R3			
	6A	03	FB	00088	CALLS	#3, DBG\$NMATCH			
	01	50	D1	0008B	CMPL	R0, #1			
		09	12	0008E	BNEQ	6\$			
04	AE	07	D0	00090	MOVL	#7, TYPE			0439
	6E	02	D0	00094	MOVL	#2, LENGTH			0440
		AA	11	00097	BRB	1\$			0422
		01	DD	00099	5\$: PUSHL	#1			0443
		B5	A9	9F	0009B	PUSHAB	DBG\$CS_LONGWORD		
		53	DD	0009E	PUSHL	R3			
	6A	03	FB	000A0	CALLS	#3, DBG\$NMATCH			
	01	50	D1	000A3	CMPL	R0, #1			
		06	12	000A6	BNEQ	7\$			
04	AE	08	D0	000A8	MOVL	#8, TYPE			0445
		64	11	000AC	BRB	12\$			0446
		01	DD	000AE	7\$: PUSHL	#1			0449
		C8	A9	9F	000B0	PUSHAB	DBG\$CS_QUADWORD		
		53	DD	000B3	PUSHL	R3			
	6A	03	FB	000B5	CALLS	#3, DBG\$NMATCH			
	01	50	D1	000B8	CMPL	R0, #1			
		06	12	000BB	BNEQ	8\$			
04	AE	09	D0	000BD	MOVL	#9, TYPE			0451
		7C	11	000C1	BRB	15\$			0452
		02	DD	000C3	8\$: PUSHL	#2			0455
		B1	A9	9F	000C5	PUSHAB	DBG\$CS_DATE_TIME		
		53	DD	000C8	PUSHL	R3			
	6A	03	FB	000CA	CALLS	#3, DBG\$NMATCH			
	01	50	D1	000CD	CMPL	R0, #1			
		06	12	000D0	BNEQ	9\$			
04	AE	23	D0	000D2	MOVL	#35, TYPE			0457
		67	11	000D6	BRB	15\$			0458
		01	DD	000D8	9\$: PUSHL	#1			0461
		D1	A9	9F	000DA	PUSHAB	DBG\$CS_OCTAWORD		
		53	DD	000DD	PUSHL	R3			
	6A	03	FB	000DF	CALLS	#3, DBG\$NMATCH			
	01	50	D1	000E2	CMPL	R0, #1			
		06	12	000E5	BNEQ	10\$			
04	AE	1A	D0	000E7	MOVL	#26, TYPE			0463
		6A	11	000EB	BRB	17\$			0464
		01	DD	000ED	10\$: PUSHL	#1			0467
		DA	A9	9F	000EF	PUSHAB	DBG\$CS_FLOAT		
		53	DD	000F2	PUSHL	R3			
	6A	03	FB	000F4	CALLS	#3, DBG\$NMATCH			

	52	50	D0	000F7	MOVL	R0, R2		
	01	52	D1	000FA	CMPL	R2, #1		
		0F	13	000FD	BEQL	11\$		
		01	DD	000FF	PUSHL	#1		0468
		A9	9F	00101	PUSHAB	DBG\$CS_F_FLOAT		
	6A	53	DD	00104	PUSHL	R3		
	01	03	FB	00106	CALLS	#3, DBG\$NMATCH		
		50	D1	00109	CMPL	R0, #1		
04	AE	09	12	0010C	BNEQ	13\$		
	6E	0A	D0	0010E	11\$: MOVL	#10, TYPE		0470
		04	D0	00112	12\$: MOVL	#4, LENGTH		0471
		80	11	00115	BRB	5\$		0422
		02	DD	00117	13\$: PUSHL	#2		0474
		A9	9F	00119	PUSHAB	DBG\$CS_D_FLOAT		
	6A	53	DD	0011C	PUSHL	R3		
	01	03	FB	0011E	CALLS	#3, DBG\$NMATCH		
		50	D1	00121	CMPL	R0, #1		
04	AE	06	12	00124	BNEQ	14\$		
		0B	D0	00126	MOVL	#11, TYPE		0476
		13	11	0012A	BRB	15\$		0477
		01	DD	0012C	14\$: PUSHL	#1		0480
		A9	9F	0012E	PUSHAB	DBG\$CS_G_FLOAT		
	6A	53	DD	00131	PUSHL	R3		
	01	03	FB	00133	CALLS	#3, DBG\$NMATCH		
		50	D1	00136	CMPL	R0, #1		
04	AE	09	12	00139	BNEQ	16\$		
	6E	1B	D0	0013B	MOVL	#27, TYPE		0482
		08	D0	0013F	15\$: MOVL	#8, LENGTH		0483
		66	11	00142	BRB	21\$		0422
		02	DD	00144	16\$: PUSHL	#2		0486
		A9	9F	00146	PUSHAB	DBG\$CS_H_FLOAT		
	6A	53	DD	00149	PUSHL	R3		
	01	03	FB	0014B	CALLS	#3, DBG\$NMATCH		
		50	D1	0014E	CMPL	R0, #1		
04	AE	09	12	00151	BNEQ	18\$		
	6E	1C	D0	00153	MOVL	#28, TYPE		0488
		10	D0	00157	17\$: MOVL	#16, LENGTH		0489
		4E	11	0015A	BRB	21\$		0422
		01	DD	0015C	18\$: PUSHL	#1		0492
		C9	9F	0015E	PUSHAB	DBG\$CS_PACKED		
	6A	53	DD	00162	PUSHL	R3		
	01	03	FB	00164	CALLS	#3, DBG\$NMATCH		
		50	D1	00167	CMPL	R0, #1		
04	AE	40	12	0016A	BNEQ	22\$		
		15	D0	0016C	MOVL	#21, TYPE		0494
		01	DD	00170	PUSHL	#1		0499
		8F	BB	00172	PUSHR	#*M<R3,R9>		
	6A	03	FB	00176	CALLS	#3, DBG\$NMATCH		
	05	50	E8	00179	BLBS	R0, 19\$		
	6E	0A	D0	0017C	MOVL	#10, LENGTH		0502
		7B	11	0017F	BRB	26\$		0499
		8F	BB	00181	19\$: PUSHR	#*M<R3,SP>		0509
00000000G	00	02	FB	00185	CALLS	#2, DBG\$NSAVE_INTEGER		
	43	50	E9	0018C	BLBC	R0, 23\$		
	50	6E	D0	0018F	MOVL	LENGTH, R0		0515
		05	19	00192	BLSS	20\$		
	1F	50	D1	00194	CMPL	R0, #31		

00000000G	00	00028828	63	15	00197		BLEQ	26\$		
			50	DD	00199	20\$:	PUSHL	R0		0517
			01	DD	0019B		PUSHL	#1		
			8F	DD	0019D		PUSHL	#165928		
			03	FB	001A3		CALLS	#3, LIB\$SIGNAL		
			77	11	001AA	21\$:	BRB	29\$		0422
			01	DD	001AC	22\$:	PUSHL	#1		0521
		8B	A9	9F	001AE		PUSHAB	DBG\$CS_ASCII		
			53	DD	001B1		PUSHL	R3		
	6A		03	FB	001B3		CALLS	#3, DBG\$NMATCH		
	01		50	D1	001B6		CMPL	R0, #1		
			43	12	001B9		BNEQ	27\$		
			01	DD	001BB		PUSHL	#1		0527
		0208	8F	BB	001BD		PUSHR	#^M<R3,R9>		
	6A		03	FB	001C1		CALLS	#3, DBG\$NMATCH		
	31		50	E9	001C4		BLBC	R0, 25\$		
		4008	8F	BB	001C7		PUSHR	#^M<R3,SP>		0533
00000000G	00		02	FB	001CB		CALLS	#2, DBG\$NSAVE_INTEGER		
	03		50	E8	001D2	23\$:	BLBS	R0, 24\$		
			01B7	31	001D5		BRW	50\$		
			6E	D5	001D8	24\$:	TSTL	LENGTH		0539
			1C	14	001DA		BGTR	25\$		
	6B		01	B0	001DC		MOVW	#1, NUM_DESC		0544
04	AB	0A	A9	9E	001DF		MOVAB	P.ABA, NUM_DESC+4		0545
			5B	DD	001E4		PUSHL	R11		0546
			01	DD	001E6		PUSHL	#1		
		000281B0	8F	DD	001E8		PUSHL	#164272		
00000000G	00		03	FB	001EE		CALLS	#3, DBG\$NMAKE_ARG_VECT		
			00D1	31	001F5		BRW	40\$		
	04	AE	0E	D0	001F8	25\$:	MOVL	#14, TYPE		0552
			73	11	001FC	26\$:	BRB	34\$		0422
			05	DD	001FE	27\$:	PUSHL	#5		0557
		91	A9	9F	00200		PUSHAB	DBG\$CS_ASCIC		
			53	DD	00203		PUSHL	R3		
	6A		03	FB	00205		CALLS	#3, DBG\$NMATCH		
	52		50	D0	00208		MOVL	R0, R2		
	01		52	D1	0020B		CMPL	R2, #1		
			0F	13	0020E		BEQL	28\$		
			02	DD	00210		PUSHL	#2		0558
		97	A9	9F	00212		PUSHAB	DBG\$CS_AC		
			53	DD	00215		PUSHL	R3		
	6A		03	FB	00217		CALLS	#3, DBG\$NMATCH		
	01		50	D1	0021A		CMPL	R0, #1		
			06	12	0021D		BNEQ	30\$		
	04	AE	26	D0	0021F	28\$:	MOVL	#38, TYPE		0560
			73	11	00223	29\$:	BRB	37\$		0422
			05	DD	00225	30\$:	PUSHL	#5		0563
		9A	A9	9F	00227		PUSHAB	DBG\$CS_ASCII		
			53	DD	0022A		PUSHL	R3		
	6A		03	FB	0022C		CALLS	#3, DBG\$NMATCH		
	52		50	D0	0022F		MOVL	R0, R2		
	01		52	D1	00232		CMPL	R2, #1		
			0F	13	00235		BEQL	31\$		
			02	DD	00237		PUSHL	#2		0564
		A0	A9	9F	00239		PUSHAB	DBG\$CS_AW		
			53	DD	0023C		PUSHL	R3		
	6A		03	FB	0023E		CALLS	#3, DBG\$NMATCH		

	01		50	D1	00241		CPL	R0	#1		
			06	12	00244		BNEQ	32\$			
04	AE		25	D0	00246	31\$:	MOVL	#37,	TYPE		0566
			4C	11	0024A		BRB	37\$			0422
			05	DD	0024C	32\$:	PUSHL	#5			0569
		A3	A9	9F	0024E		PUSHAB	DBG\$CS_ASCIZ			
			53	DD	00251		PUSHL	R3			
	6A		03	FB	00253		CALLS	#3,	DBG\$NMATCH		
	52		50	D0	00256		MOVL	R0,	R2		
	01		52	D1	00259		CPL	R2,	#1		
			0F	13	0025C		BEQL	33\$			
			02	DD	0025E		PUSHL	#2			0570
		A9	A9	9F	00260		PUSHAB	DBG\$CS_AZ			
			53	DD	00263		PUSHL	R3			
	6A		03	FB	00265		CALLS	#3,	DBG\$NMATCH		
	01		50	D1	00268		CPL	R0,	#1		
			06	12	0026B		BNEQ	35\$			
04	AE		27	D0	0026D	33\$:	MOVL	#39,	TYPE		0572
			25	11	00271	34\$:	BRB	37\$			0422
			05	DD	00273	35\$:	PUSHL	#5			0575
		AC	A9	9F	00275		PUSHAB	DBG\$CS_ASCID			
			53	DD	00278		PUSHL	R3			
	6A		03	FB	0027A		CALLS	#3,	DBG\$NMATCH		
	52		50	D0	0027D		MOVL	R0,	R2		
	01		52	D1	00280		CPL	R2,	#1		
			0F	13	00283		BEQL	36\$			
			02	DD	00285		PUSHL	#2			0576
		B2	A9	9F	00287		PUSHAB	DBG\$CS_AD			
			53	DD	0028A		PUSHL	R3			
	6A		03	FB	0028C		CALLS	#3,	DBG\$NMATCH		
	01		50	D1	0028F		CPL	R0,	#1		
			07	12	00292		BNEQ	38\$			
04	AE		38	D0	00294	36\$:	MOVL	#56,	TYPE		0578
		FDA8	31	00298	37\$:	BRW	1\$				0422
			01	DD	0029B	38\$:	PUSHL	#1			0584
		06	A9	9F	0029D		PUSHAB	DBG\$CS_CR			
			53	DD	002A0		PUSHL	R3			
	6A		03	FB	002A2		CALLS	#3,	DBG\$NMATCH		
	0F		50	E9	002A5		BLBC	R0,	39\$		
00000000G	00	000280D0	8F	DD	002A8		PUSHL	#164048			0586
			01	FB	002AE		CALLS	#1,	DBG\$NMAKE_ARG_VECT		
			12	11	002B5		BRB	40\$			
			53	DD	002B7	39\$:	PUSHL	R3			0588
00000000G	00		01	FB	002B9		CALLS	#1,	DBG\$NNEXT_WORD		
			50	DD	002C0		PUSHL	R0			
00000000G	00		01	FB	002C2		CALLS	#1,	DBG\$NSYNTAX_ERROR		
	OC		50	D0	002C9	40\$:	MOVL	R0,	@MESSAGE_VECT		0584
			00BF	31	002CD	41\$:	BRW	50\$			0589
			AC	D0	002D0	42\$:	MOVL	MESSAGE_VECT,	R4		0600
			54	DD	002D4		PUSHL	R4			
			04	DD	002D6		PUSHL	#4			0599
		0188	8F	BB	002D8		PUSHR	#^M<R3,R7,R8>			
00000000G	00		05	FB	002DC		CALLS	#5,	DBG\$NPARSE_ADDRESS		
	56		50	D0	002E3		MOVL	R0,	STATUS		
			07	13	002E6		BEQL	43\$			0606
			56	D1	002E8		CPL	STATUS,	#1		0609
	01		E0	12	002EB		BNEQ	41\$			

			1A	11	002ED		BRB	44\$				0611
			01	DD	002EF	43\$:	PUSHL	#1				0618
		04	A9	9F	002F1		PUSHAB	DBG\$CS_EQUAL_SIGN				
			53	DD	002F4		PUSHL	R3				
	6A		03	FB	002F6		CALLS	#3, DBG\$NMATCH				
	1C		50	E8	002F9		BLBS	R0, 45\$				
			01	DD	002FC		PUSHL	#1				0622
		06	A9	9F	002FE		PUSHAB	DBG\$CS_CR				
			53	DD	00301		PUSHL	R3				
	6A		03	FB	00303		CALLS	#3, DBG\$NMATCH				
	71		50	E9	00306		BLBC	R0, 48\$				
00000000G	00	000280D0	8F	DD	00309	44\$:	PUSHL	#164048				0624
			01	FB	0030F		CALLS	#1, DBG\$NMAKE_ARG_VECT				
			74	11	00316		BRB	49\$				0622
	50		67	D0	00318	45\$:	MOVL	(NOUN_NODE), ADDR_EXP_DESC				0633
			03	DD	0031B		PUSHL	#3				0639
00000000G	00		01	FB	0031D		CALLS	#1, DBG\$GET_TEMPMEM				
			50	D0	00324		MOVL	R0, ADVERB_NODE				
FFFFFFFFF	8F		04	AE	D1 00327		CMPL	TYPE, #-1				0646
			0F	12	0032F		BNEQ	46\$				
			5E	DD	00331		PUSHL	SP				0658
		08	AE	9F	00333		PUSHAB	TYPE				
00000000G	00		02	FB	00336		CALLS	#2, DBG\$NGET_OVERRIDE_TYPE				
	0C		50	E9	0033D		BLBC	R0, 47\$				
	04	A5	52	D0	00340	46\$:	MOVL	ADVERB_NODE, 4(R5)				0661
		62	04	AE	90 00344		MOVB	TYPE, (ADVERB_NODE)				0662
	04	A2	6E	D0	00348		MOVL	LENGTH, 4(ADVERB_NODE)				0663
		52	08	A7	9E 0034C	47\$:	MOVAB	8(R7), LINK				0675
			04	DD	00350		PUSHL	#4				0677
00000000G	00		01	FB	00352		CALLS	#1, DBG\$GET_TEMPMEM				
	57		50	D0	00359		MOVL	R0, NOUN_NODE				
	62		57	D0	0035C		MOVL	NOUN_NODE, (LINK)				0678
			05	DD	0035F		PUSHL	#5				0685
			54	DD	00361		PUSHL	R4				0686
			7E	D4	00363		CLRL	-(SP)				0685
			57	DD	00365		PUSHL	NOUN_NODE				
		0108	8F	BB	00367		PUSHR	#^M<R3,R8>				
00000000G	00		06	FB	0036B		CALLS	#6, DBG\$NPARSE_EXPRESSION				
	56		50	D0	00372		MOVL	R0, STATUS				
	1B		56	E8	00375		BLBS	STATUS, 51\$				
			15	12	00378		BNEQ	50\$				0693
			53	DD	0037A	48\$:	PUSHL	R3				0695
00000000G	00		01	FB	0037C		CALLS	#1, DBG\$NNEXT_WORD				
			50	DD	00383		PUSHL	R0				
00000000G	00		01	FB	00385		CALLS	#1, DBG\$NSYNTAX_ERROR				
	64		50	D0	0038C	49\$:	MOVL	R0, (R4)				
	50		04	D0	0038F	50\$:	MOVL	#4, R0				0696
			04	04	00392		RET					
	16	04	AE	D1	00393	51\$:	CMPL	TYPE, #22				0701
			1B	12	00397		BNEQ	52\$				
	50		67	D0	00399		MOVL	(NOUN_NODE), VALPTR				0704
	0E	16	A0	91	0039C		CMPB	22(VACPTR), #14				0705
			12	13	003A0		BEQL	52\$				
		0E	A9	9F	003A2		PUSHAB	P.ABB				0707
			01	DD	003A5		PUSHL	#1				
			8F	DD	003A7		PUSHL	#166346				
00000000G	00	000289CA	03	FB	003AD		CALLS	#3, LIB\$SIGNAL				

DBGEXADep
V04-000

D 9
16-Sep-1984 01:13:03
14-Sep-1984 12:16:54

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEXADep.B32;1

Page 20
(3)

50

01 D0 003B4 52\$: MOVL #1, R0
04 003B7 RET

: 0710
: 0711

; Routine Size: 952 bytes, Routine Base: DBG\$CODE + 0000

```

582 0712 1 GLOBAL ROUTINE DBG$SET_PAGE_PROT (PAGE_LIST, RESET_FLAG, MESSAGE_VECT) =
583 0713 1
584 0714 1 ++
585 0715 1 FUNCTIONAL DESCRIPTION:
586 0716 1
587 0717 1 This routine accepts the page list data structure produced by dbg$ngget_pages
588 0718 1 or create_page_list and changes the protection of each page so specified
589 0719 1 to provide write access for DEPOSIT's. As this is done, a new data structure
590 0720 1 is constructed in which each node in the tree is replace with a node that
591 0721 1 has one more field. This extra field contains the old protection of the
592 0722 1 page so that it may be reset after the deposit.
593 0723 1
594 0724 1 When called upon to reset the page protections, this routine uses the data
595 0725 1 structure it produced in the first step, and changes all page protections
596 0726 1 back to their original values.
597 0727 1
598 0728 1 FORMAL PARAMETERS:
599 0729 1
600 0730 1 PAGE_LIST - The address of a longword containing the address
601 0731 1 of the first node in the page list.
602 0732 1
603 0733 1 RESET_FLAG - A longword containing true or false. If false,
604 0734 1 protection is changed to allow write access. If
605 0735 1 true, protection for each page is set back to
606 0736 1 the original value
607 0737 1
608 0738 1 MESSAGE_VECT - The address of a longword to contain the address
609 0739 1 of a message argument vector for errors
610 0740 1
611 0741 1 IMPLICIT INPUTS:
612 0742 1
613 0743 1 NONE
614 0744 1
615 0745 1 IMPLICIT OUTPUTS:
616 0746 1
617 0747 1 When NOT .reset_flag, a new page list is constructed.
618 0748 1 When .reset_flag, page protections are reset to original values.
619 0749 1
620 0750 1 On failure, a message argument vector is constructed.
621 0751 1
622 0752 1 ROUTINE VALUE:
623 0753 1
624 0754 1 An unsigned integer longword completion code
625 0755 1
626 0756 1 COMPLETION CODES:
627 0757 1
628 0758 1 sts$k_success (1) - Success. Page protection set or reset.
629 0759 1
630 0760 1 sts$k_severe (4) - Failure. Message argument vector produced.
631 0761 1
632 0762 1 SIDE EFFECTS:
633 0763 1
634 0764 1 NONE
635 0765 1
636 0766 1 --
637 0767 2 BEGIN
638 0768 2

```

```

639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695

```

```

FIELD PROT_NODE_FIELDS =
  SET
  NODE_LINK      = [0, 0, 32, 0],      ! Link field
  PAGE_NUM       = [1, 0, 32, 0],      ! Page number field
  OLD_PROT       = [2, 0, 8, 0]       ! Old protection field
  TES;

LITERAL
  PROT_NODE_SIZE = 3;                  ! Length of node in longwords

MACRO
  PROT_NODE      = BLOCK [prot_node_size] FIELD (prot_node_fields) %;

LOCAL
  PAGE_VECT      : VECTOR [2];        ! Vector to hold page numbers

! Check to see if we are setting or resetting protections
IF .reset_flag
THEN
  BEGIN
    ! We are restoring protection. Therefore, page_list contains the address
    ! of a longword that contains the address of the modified page list.
    LOCAL
      NODE : REF prot_node;          ! Modified page list node

    ! Obtain the address of the list.
    node = ..page_list;

    ! Loop and reset the page protections
    WHILE .node NEQA 0
    DO
      BEGIN
        ! Reset the old protection
        !
        page_vect [0] = .node [page_num];
        page_vect [1] = .node [page_num]; ! We don't use a range
        $SETPRT (inadr = page_vect [0], prot = .node [old_prot]);

        ! Obtain the next node
        !
        node = .node [node_link];
      END;          ! End of loop
    END

  ELSE
  BEGIN

```

```

: 696      0826      W
: 697      0827      W
: 698      0828      W
: 699      0829      W
: 700      0830      W
: 701      0831      W
: 702      0832      W
: 703      0833      W
: 704      0834      W
: 705      0835      W
: 706      0836      W
: 707      0837      W
: 708      0838      W
: 709      0839      W
: 710      0840      W
: 711      0841      W
: 712      0842      W
: 713      0843      W
: 714      0844      W
: 715      0845      W
: 716      0846      W
: 717      0847      W
: 718      0848      W
: 719      0849      W
: 720      0850      W
: 721      0851      W
: 722      0852      W
: 723      0853      W
: 724      0854      W
: 725      0855      W
: 726      0856      W
: 727      0857      W
: 728      0858      W
: 729      0859      W
: 730      0860      W
: 731      0861      W
: 732      0862      W
: 733      0863      W
: 734      0864      W
: 735      0865      W
: 736      0866      W
: 737      0867      W
: 738      P 0868      W
: 739      P 0869      W
: 740      0870      6
: 741      0871      5
: 742      0872      6
: 743      0873      6
: 744      0874      6
: 745      0875      5
: 746      0876      5
: 747      0877      4
: 748      0878      4
: 749      0879      4
: 750      0880      4
: 751      0881      4
: 752      0882      4

```

```

BUILTIN PROBEW;
: We are changing protections. Therefore, we must construct the modified
: page list.
LOCAL
LINK,           ! Temporary pointer
NEW_NODE : REF prot_node,      ! Modified list node
OLD_NODE : REF dbg$l_link_node; ! Old list node
: Recover the head node of the unmodified page list
old_node = ..page_list;
(link = .page_list) = 0;
: Loop changing the protection and constructing the new list
: Check to see if this page is already writeable, before we
: attempt to set the protection! This fixes a bug in trying
: to deposit into a global section (mapped for write access)
: when changing page protections is not allowed.
WHILE .old_node NEQA 0 DO
BEGIN
IF NOT PROBEW(%REF(0),%REF(1),.old_node[dbg$l_link_node_value]) THEN
BEGIN
! Allocate a modified node and link it
:
new_node = dbg$get_tempmem (prot_node_size);
link = .new_node;
link = new_node [node_link];
: Copy over relevant info
:
new_node [page_num] = .old_node [dbg$l_link_node_value];
: Change the protection, if possible
:
page_vect [0] = .new_node [page_num];
page_vect [1] = .new_node [page_num];
IF NOT $$SETPRT (inadr = page_vect [0],
                 prot = prt$c_uw,
                 prvppt = new_node [old_prot])
THEN
BEGIN
message_vect = dbg$nmake_arg_vect (dbg$_badtarget);
RETURN sfs$k_severe;
END;
END;
: Pick up the next node
old_node = .old_node [dbg$l_link_node_link];

```

: 753 0883
: 754 0884
: 755 0885
: 756 0886
: 757 0887
: 758 0888

END; ! End of loop
END;
RETURN sts\$k success;
END; ? End of dbg\$set_page_prot

1
1
1
1
1
1

.EXTRN SYSS\$SETPRT

			003C 00000	.ENTRY	DBG\$SET PAGE PROT, Save R2,R3,R4,R5	: 0712
	55	00000000G	00 9E 00002	MOVAB	SYSS\$SETPRT, R5	
	5E		08 C2 00009	SUBL2	#8, SP	
	22	08	AC E9 0000C	BLBC	RESET_FLAG, 2\$: 0787
	52	04	BC D0 00010	MOVL	@PAGE_LIST, NODE	: 0799
			7B 13 00014	BEQL	5\$: 0804
	6E	04	A2 D0 00016	MOVL	4(NODE), PAGE_VECT	: 0810
04	AE	04	A2 D0 0001A	MOVL	4(NODE), PAGE_VECT+4	: 0811
			7E D4 0001F	CLRL	-(SP)	: 0813
	7E	08	A2 9A 00021	MOVZBL	8(NODE), -(SP)	
			7E 7C 00025	CLRQ	-(SP)	
		10	AE 9F 00027	PUSHAB	PAGE_VECT	
	65		05 FB 0002A	CALLS	#5, SYSS\$SETPRT	
	52		62 D0 0002D	MOVL	(NODE), NODE	: 0818
			E2 11 00030	BRB	1\$: 0804
	53	04	BC D0 00032	MOVL	@PAGE_LIST, OLD_NODE	: 0838
	54	04	AC D0 00036	MOVL	PAGE_LIST, LINK	: 0840
			64 D4 0003A	CLRL	(LINK)	
			53 D5 0003C	TSTL	OLD_NODE	: 0848
			51 13 0003E	BEQL	5\$	
04	B3	01	00 0D 00040	PROBEW	#0, #1, @4(OLD_NODE)	: 0850
			45 12 00045	BNEQ	4\$	
			03 DD 00047	PUSHL	#3	: 0855
	00000000G	00	01 FB 00049	CALLS	#1, DBG\$GET TEMP MEM	
		52	50 D0 00050	MOVL	R0, NEW_NODE	
		64	52 D0 00053	MOVL	NEW_NODE, (LINK)	: 0856
		54	52 D0 00056	MOVL	NEW_NODE, LINK	: 0857
04	A2	04	A3 D0 00059	MOVL	4(OLD_NODE), 4(NEW_NODE)	: 0861
	6E	04	A2 D0 0005E	MOVL	4(NEW_NODE), PAGE_VECT	: 0865
04	AE	04	A2 D0 00062	MOVL	4(NEW_NODE), PAGE_VECT+4	: 0866
		08	A2 9F 00067	PUSHAB	8(NEW_NODE)	: 0870
			04 DD 0006A	PUSHL	#4	
			7E 7C 0006C	CLRQ	-(SP)	
		10	AE 9F 0006E	PUSHAB	PAGE_VECT	
	65		05 FB 00071	CALLS	#5, SYSS\$SETPRT	
	15		50 E8 00074	BLBS	R0, 4\$	
		00028820	8F DD 00077	PUSHL	#165920	: 0873
00000000G	00		01 FB 0007D	CALLS	#1, DBG\$NMAKE ARG_VECT	
0C	BC		50 D0 00084	MOVL	R0, @MESSAGE_VECT	
	50		04 D0 00088	MOVL	#4, R0	: 0874
			04 0008B	RET		
	53		63 D0 0008C	MOVL	(OLD_NODE), OLD_NODE	: 0881
			AB 11 0008F	BRB	3\$: 0848
	50		01 D0 00091	MOVL	#1, R0	: 0887
			04 00094	RET		: 0888

DBGEXADP
V04-000

¹₉
16-Sep-1984 01:13:03
14-Sep-1984 12:16:54

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGEXADP.B32;1

Page 25
(4)

; Routine Size: 149 bytes, Routine Base: DBG\$CODE + 03B8

```

: 760 0889 1 GLOBAL ROUTINE DBG$NPARSE_EVALUATE (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
: 761 0890 1
: 762 0891 1  !++
: 763 0892 1  FUNCTIONAL DESCRIPTION:
: 764 0893 1
: 765 0894 1      Parse network for the EVALUATE command. The parsing method used is that
: 766 0895 1      of ATN's. This network constructs a command execution tree to be executed
: 767 0896 1      by dbg$nextecute_evaluate.
: 768 0897 1
: 769 0898 1  FORMAL PARAMETERS:
: 770 0899 1
: 771 0900 1      INPUT_DESC          - A longword containing the address of a standard
: 772 0901 1      string descriptor which reflects the input string
: 773 0902 1
: 774 0903 1      VERB_NODE            - A longword containing the address of the verb
: 775 0904 1      (head) node of the command execution tree
: 776 0905 1
: 777 0906 1      MESSAGE_VECT        - The address of a longword to contain the
: 778 0907 1      address of a standard message argument vector on errors
: 779 0908 1
: 780 0909 1
: 781 0910 1  IMPLICIT INPUTS:
: 782 0911 1
: 783 0912 1      NONE
: 784 0913 1
: 785 0914 1  IMPLICIT OUTPUTS:
: 786 0915 1
: 787 0916 1      On success, a command execution tree is constructed representing the parsed
: 788 0917 1      input command.
: 789 0918 1
: 790 0919 1      On failure, a message argument vector is constructed.
: 791 0920 1
: 792 0921 1  ROUTINE VALUE:
: 793 0922 1
: 794 0923 1      An unsigned integer longword completion code
: 795 0924 1
: 796 0925 1  COMPLETION CODES:
: 797 0926 1
: 798 0927 1      sts$k_success (1)      - Success. Input parsed and execution tree constructed.
: 799 0928 1
: 800 0929 1      sts$k_severe  (4)      - Failure. Tree not constructed. Message vector
: 801 0930 1      constructed.
: 802 0931 1
: 803 0932 1  SIDE EFFECTS:
: 804 0933 1
: 805 0934 1      NONE
: 806 0935 1
: 807 0936 1  !--
: 808 0937 1
: 809 0938 2  BEGIN
: 810 0939 2
: 811 0940 2  MAP
: 812 0941 2      INPUT_DESC : REF BLOCK [, BYTE],
: 813 0942 2      VERB_NODE  : REF dbg$verb_node;
: 814 0943 2
: 815 0944 2  ! Define counted strings used at this level
: 816 0945 2

```

```

: 817
: 818
: 819
: 820
: 821
: 822
: 823
: 824
: 825
: 826
: 827
: 828
: 829
: 830
: 831
: 832
: 833
: 834
: 835
: 836
: 837
: 838
: 839
: 840
: 841
: 842
: 843
: 844
: 845
: 846
: 847
: 848
: 849
: 850
: 851
: 852
: 853
: 854
: 855
: 856
: 857
: 858
: 859
: 860
: 861
: 862
: 863
: 864
: 865
: 866
: 867
: 868
: 869
: 870
: 871
: 872
: 873

```

```

BIND
DBG$CS_ADDRESS      = UPLIT BYTE (7, 'ADDRESS'),
DBG$CS_CONDITION    = UPLIT BYTE (15, 'CONDITION_VALUE'),
DBG$CS_BINARY       = UPLIT BYTE (6, 'BINARY'),
DBG$CS_HEXADecimal  = UPLIT BYTE (11, 'HEXADECIMAL'),
DBG$CS_OCTAL        = UPLIT BYTE (5, 'OCTAL'),
DBG$CS_DECIMAL      = UPLIT BYTE (7, 'DECIMAL'),
DBG$CS_COMMA        = UPLIT BYTE (1, dbg$k_comma),
DBG$CS_CR           = UPLIT BYTE (1, dbg$k_car_return),
DBG$CS_SLASH        = UPLIT BYTE (1, dbg$k_slash);

LOCAL
STATUS,              ! Holds return status
NOUN_NODE            : REF dbg$noun_node,      ! Object
NEXT_NOUN_NODE      : REF dbg$noun_node,      ! Next object
ADVERB_NODE         : REF dbg$adverb_node,    ! Modifier
RADIX;              ! Radix

! Create and link a noun node
noun_node = dbg$get_tempmem (dbg$k_noun_node_size);
verb_node [dbg$l_verb_object_ptr] = .noun_node;

! Start with default radix
radix = .dbg$gb_radix[dbg$b_radix_output_over];

! Accept any / switches. Start with plain jane evaluate - this may get changed.
verb_node [dbg$b_verb_composite] = evaluate;

WHILE dbg$nmatch (.input_desc, dbg$cs_slash, 1)
DO
  BEGIN
    ! Try to match legal switches
    SELECTONE true
    OF
    SET
      [dbg$nmatch (.input_desc, dbg$cs_address, 1)] : ! address switch
      BEGIN
        verb_node [dbg$b_verb_composite] = evaluate_addr;
      END;
      [dbg$nmatch (.input_desc, dbg$cs_condition, 1)] : ! condition switch
      BEGIN
        verb_node [dbg$b_verb_composite] = evaluate_cond;
      END;
      [dbg$nmatch (.input_desc, dbg$cs_binary, 2)] :
      BEGIN
        radix = dbg$k_binary;

```

```

874      1003      END;
875      1004
876      1005      [dbg$nmact (.input_desc, dbg$cs_hexadecimal, 1)] : ! Change radix to hex
877      1006      BEGIN
878      1007      radix = dbg$k_hex;
879      1008      END;
880      1009
881      1010      [dbg$nmact (.input_desc, dbg$cs_octal, 1)] : ! Change radix to octal
882      1011      BEGIN
883      1012      radix = dbg$k_octal;
884      1013      END;
885      1014
886      1015      [dbg$nmact (.input_desc, dbg$cs_decimal, 1)] : ! Change radix to decimal
887      1016      BEGIN
888      1017      radix = dbg$k_decimal;
889      1018      END;
890      1019
891      1020      [OTHERWISE] : ! Syntax error
892      1021      BEGIN
893      1022      .message_vect = dbg$nsyntax_error (dbg$nnext_word (.input_desc));
894      1023      RETURN sts$k_severe;
895      1024      END;
896      1025
897      1026      TES
898      1027
899      1028      END;
900      1029
901      1030
902      1031      ! Be sure the user has supplied us with some sort of input after the verb.
903      1032      !
904      1033      IF dbg$nmact (.input_desc, dbg$cs_cr, 1)
905      1034      THEN
906      1035      BEGIN
907      1036      .message_vect = dbg$nmact_arg_vect (dbg$_needmore);
908      1037      RETURN sts$k_severe;
909      1038      END;
910      1039
911      1040
912      1041      ! Loop until input is exhausted.
913      1042      !
914      1043      WHILE TRUE DO
915      1044      BEGIN
916      1045
917      1046
918      1047      ! Obtain the object of the evaluate based on the verb composite
919      1048      !
920      1049      CASE .verb_node [dbg$b_verb_composite] FROM evaluate TO evaluate_addr
921      1050      OF
922      1051      SET
923      1052
924      1053
925      1054      ! Call the Expression Interpreter to obtain expression value.
926      1055      !
927      1056      [evaluate] :
928      1057      STATUS = DBG$NPARSE_EXPRESSION (.INPUT_DESC,
929      1058      .DBG$GB_RADIX[DBG$B_RADIX_INPUT],
930      1059      NOUN_NODE [DBG$L_NOON_VALUE],

```

```

: 931
: 932
: 933
: 934
: 935
: 936
: 937
: 938
: 939
: 940
: 941
: 942
: 943
: 944
: 945
: 946
: 947
: 948
: 949
: 950
: 951
: 952
: 953
: 954
: 955
: 956
: 957
: 958
: 959
: 960
: 961
: 962
: 963
: 964
: 965
: 966
: 967
: 968
: 969
: 970
: 971
: 972
: 973
: 974
: 975
: 976
: 977
: 978
: 979
: 980
: 981
: 982
: 983
: 984
: 985
: 986
: 987

```

```

TOKEN$K_TERM_COMMA, .MESSAGE_VECT);
! Call Address Expression Interpreter.
[evaluate_addr] :
STATUS = DBG$NPARSE_ADDRESS (.INPUT_DESC,
NOUN_NODE [DBG$L_NOUN_VALUE],
DBG$GB_RADIX[DBG$B_RADIX_INPUT],
TOKEN$K_TERM_COMMA, .MESSAGE_VECT);

TES;

! Check the return status.
! SELECTONE .status OF
SET

! If successful, then continue.
[sts$k_success]:
BEGIN
0;
END;

! If a warning is returned, a comma may have been discovered in the
! input stream. Check for this. If there is no comma, a syntax error
! has occurred. If a comma is present, there must be more input or else
! error. If all is OK, then link in another noun node.
[sts$k_warning]:
BEGIN
IF NOT dbg$nmatch (.input_desc, dbg$cs_comma, 1)
THEN
BEGIN
.message_vect = dbg$nsyntax_error (dbg$next_word (.input_desc));
RETURN sts$k_severe;
END;
IF dbg$nmatch (.input_desc, dbg$cs_cr, 1)
THEN
BEGIN
.message_vect = dbg$make_arg_vect (dbg$_needmore);
RETURN sts$k_severe;
END;
next_noun_node = dbg$get_tempmem (dbg$k_noun_node_size);
noun_node[dbg$l_noun_link] = .next_noun_node;
noun_node = .next_noun_node;
END;

! Anything else, return severe error.
[OTHERWISE]:
BEGIN
RETURN sts$k_severe;

```

```

: 988      1117
: 989      1118
: 990      1119
: 991      1120
: 992      1121
: 993      1122
: 994      1123
: 995      1124
: 996      1125
: 997      1126
: 998      1127
: 999      1128
1000      1129
1001      1130
1002      1131
1003      1132
1004      1133
1005      1134
1006      1135
1007      1136
1008      1137
1009      1138
1010      1139
1011      1140
1012      1141
1013      1142
1014      1143
1015      1144
1016      1145
1017      1146
1018      1147
1019      1148
: 1020     1149

```

```

END;

TES;

! Check for exhausted input.
! IF dbg$match (.input_desc, dbg$cs_cr, 1) OR
!   .input_desc[dsc$w_length] EQL 0
THEN
  EXITLOOP;
END; ! End while loop.

! Check for a modifier.
! IF .radix NEQ dbg$k_default
THEN
  BEGIN
    ! Construct a radix modifier node and link to verb node
    ! adverb_node = dbg$get_tempmem (dbg$k_adverb_node_size);
    adverb_node [dbg$b_adverb_literal] = .radix;
    verb_node [dbg$l_verb_adverb_ptr] = .adverb_node;
    adverb_node [dbg$l_adverb_link] = 0;
  END;
RETURN sts$k_success;
END; ! End of dbg$parse_evaluate

```

													.PSECT			DBG\$PLIT, NOWRT, SHR, PIC, 0																		
													07	000A8	P.ABC:	.BYTE	7																	
													41	000A9		.ASCII	\ADDRESS\																	
45	55	4C	41	56	5F	4E	4F	49	54	49	44	4E	4F	0F	000B0	P.ABD:	.BYTE	15																
														43	000B1		.ASCII	\CONDITION_VALUE\																
														06	000C0	P.ABE:	.BYTE	6																
														59	52	41	4E	49	42	000C1		.ASCII	\BINARY\											
														0B	000C7	P.ABF:	.BYTE	11																
														4C	41	4D	49	43	45	44	41	58	45	48	000C8		.ASCII	\HEXADECIMAL\						
														05	000D3	P.ABG:	.BYTE	5																
														4C	41	54	43	4F	000D4		.ASCII	\OCTAL\												
														07	000D9	P.ABH:	.BYTE	7																
														4C	41	4D	49	43	45	44	000DA		.ASCII	\DECIMAL\										
														2C	01	000E1	P.ABI:	.BYTE	1, 44															
														0D	01	000E3	P.ABJ:	.BYTE	1, 13															
														2F	01	000E5	P.ABK:	.BYTE	1, 47															

```

DBG$CS_ADDRESS= P.ABC
DBG$CS_CONDITION= P.ABD
DBG$CS_BINARY= P.ABE

```

DBG\$CS_HEXADecimal= P.ABF
DBG\$CS_OCTAL= P.ABG
DBG\$CS_DECIMAL= P.ABH
DBG\$CS_COMMA= P.ABI
DBG\$CS_CR= P.ABJ
DBG\$CS_SLASH= P.ABK

```

.PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                07FC 00000
.ENTRY DBG$NPARSE EVALUATE, Save R2,R3,R4,R5,R6,- : 0889
                                R7,R8,R9,R10
5A 00000000G 00 9E 00002 MOVAB DBG$GET_TEMPmem, R10
59 00000000G 00 9E 00009 MOVAB DBG$NMATCH, R9
58 00000000' EF 9E 00010 MOVAB DBG$CS_CR, R8
                                04 DD 00017 PUSHL #4 : 0966
6A 01 FB 00019 CALLS #1, DBG$GET_TEMPmem
53 50 D0 0001C MOVL R0, NOUN_NODE
54 08 AC D0 0001F MOVL VERB_NODE, R4 : 0968
08 A4 53 D0 00023 MOVL NOUN_NODE, 8(R4)
01 55 00000000G 00 9A 00027 MOVZBL DBG$GB_RADIX+2, RADIX : 0973
A4 01 90 0002E MOVB #1, 1(R4) : 0978
52 04 AC D0 00032 MOVL INPUT_DESC, R2 : 0980
                                01 DD 00036 1$:
                                02 A8 9F 00038 PUSHAB DBG$CS_SLASH
                                52 DD 0003B PUSHL R2
69 03 FB 0003D CALLS #3, DBG$NMATCH
03 50 E8 00040 BLBS R0, 2$
                                008E 31 0C043 BRW 10$
                                01 DD 00046 2$:
                                C5 A8 9F 00048 PUSHAB DBG$CS_ADDRESS
                                52 DD 0004B PUSHL R2
69 03 FB 0004D CALLS #3, DBG$NMATCH
01 50 D1 00050 CMPL R0, #1
                                06 12 00053 BNEQ 4$
01 A4 02 90 00055 MOVB #2, 1(R4) : 0992
                                DB 11 00059 3$:
                                01 DD 0005B 4$:
                                CD A8 9F 0005D PUSHAB DBG$CS_CONDITION
                                52 DD 00060 PUSHL R2
69 03 FB 00062 CALLS #3, DBG$NMATCH
01 50 D1 00065 CMPL R0, #1
                                06 12 00068 BNEQ 5$
01 A4 03 90 0006A MOVB #3, 1(R4) : 0997
                                C6 11 0006E BRB 1$ : 0982
                                02 DD 00070 5$:
                                DD A8 9F 00072 PUSHAB DBG$CS_BINARY
                                52 DD 00075 PUSHL R2
69 03 FB 00077 CALLS #3, DBG$NMATCH
01 50 D1 0007A CMPL R0, #1
                                05 12 0007D BNEQ 6$
55 02 D0 0007F MOVL #2, RADIX : 1002
                                B2 11 00082 BRB 1$ : 0982
                                01 DD 00084 6$:
                                E4 A8 9F 00086 PUSHAB DBG$CS_HEXADecimal
                                52 DD 00089 PUSHL R2
69 03 FB 0008B CALLS #3, DBG$NMATCH : 1005

```

	01		50	D1	0008E		CMPL	R0, #1		
			05	12	00091		BNEQ	7\$		
	55		10	DD	00093		MOVL	#16, RADIX		1007
			9E	11	00096		BRB	1\$		0982
			01	DD	00098	7\$:	PUSHL	#1		1010
		F0	A8	9F	0009A		PUSHAB	DBG\$CS_OCTAL		
			52	DD	0009D		PUSHL	R2		
	69		03	FB	0009F		CALLS	#3, DBG\$NMATCH		
	01		50	D1	000A2		CMPL	R0, #1		
			05	12	000A5		BNEQ	8\$		
	55		08	DD	000A7		MOVL	#8, RADIX		1012
			8A	11	000AA		BRB	1\$		0982
			01	DD	000AC	8\$:	PUSHL	#1		1015
		F6	A8	9F	000AE		PUSHAB	DBG\$CS_DECIMAL		
			52	DD	000B1		PUSHL	R2		
	69		03	FB	000B3		CALLS	#3, DBG\$NMATCH		
	01		50	D1	000B6		CMPL	R0, #1		
			05	12	000B9		BNEQ	9\$		
	55		0A	DD	000BB		MOVL	#10, RADIX		1017
			99	11	000BE		BRB	3\$		0982
			52	DD	000C0	9\$:	PUSHL	R2		1022
00000000G	00		01	FB	000C2		CALLS	#1, DBG\$NNEXT_WORD		
00000000G	00		50	DD	000C9		PUSHL	R0		
			01	FB	000CB		CALLS	#1, DBG\$NSYNTAX_ERROR		
			70	11	000D2		BRB	17\$		
			01	DD	000D4	10\$:	PUSHL	#1		1033
		0104	8F	BB	000D6		PUSHR	#^M<R2,R8>		
	69		03	FB	000DA		CALLS	#3, DBG\$NMATCH		
	57		50	E8	000DD		BLBS	R0, 16\$		
01	50	00000000G	00	9A	000E0	11\$:	MOVZBL	DBG\$GB_RADIX, R0		1058
	01		A4	8F	000E7		CASEB	1(R4), #1, #1		1049
	0016		0004		000EC	12\$:	.WORD	13\$-12\$,- 14\$-12\$		
			0C	AC	000F0	13\$:	PUSHL	MESSAGE_VECT		1060
			01	DD	000F3		PUSHL	#1		1059
			09	BB	000F5		PUSHR	#^M<R0,R3>		
			52	DD	000F7		PUSHL	R2		
00000000G	00		05	FB	000F9		CALLS	#5, DBG\$NPARSE_EXPRESSION		
			10	11	00100		BRB	15\$		
			0C	AC	00102	14\$:	PUSHL	MESSAGE_VECT		1068
			01	DD	00105		PUSHL	#1		1066
			50	DD	00107		PUSHL	R0		1067
			0C	BB	00109		PUSHR	#^M<R2,R3>		1066
00000000G	00		05	FB	0010B		CALLS	#5, DBG\$NPARSE_ADDRESS		
	57		50	DD	00112	15\$:	MOVL	R0, STATUS		
	01		57	D1	00115		CMPL	STATUS, #1		1081
			45	13	00118		BEQL	20\$		
			57	D5	0011A		TSTL	STATUS		1092
			3D	12	0011C		BNEQ	19\$		
			01	DD	0011E		PUSHL	#1		1094
		FE	A8	9F	00120		PUSHAB	DBG\$CS_COMMA		
			52	DD	00123		PUSHL	R2		
	69		03	FB	00125		CALLS	#3, DBG\$NMATCH		
	95		50	E9	00128		BLBC	R0, 9\$		
			01	DD	0012B		PUSHL	#1		1100
		0104	8F	BB	0012D		PUSHR	#^M<R2,R8>		
	69		03	FB	00131		CALLS	#3, DBG\$NMATCH		

	13		50	E9	00134		BLBC	R0, 18\$		
		000280D0	8F	DD	00137	16\$:	PUSHL	#164048		1103
00000000G	00		01	FB	0013D		CALLS	#1, DBG\$NMAKE_ARG_VECT		
	0C		50	D0	00144	17\$:	MOVL	R0, @MESSAGE_VECT		
			11	11	00148		BRB	19\$		1104
			04	DD	0014A	18\$:	PUSHL	#4		1106
	6A		01	FB	0014C		CALLS	#1, DBG\$GET_TEMP MEM		
	56		50	D0	0014F		MOVL	R0, NEXT_NOUN_NODE		
08	A3		56	D0	00152		MOVL	NEXT_NOUN_NODE, 8(NOUN_NODE)		1107
	53		56	D0	00156		MOVL	NEXT_NOUN_NODE, NOUN_NODE		1108
			04	11	00159		BRB	20\$		1075
	50		04	D0	0015B	19\$:	MOVL	#4, R0		1116
				04	0015E		RET			
			01	DD	0015F	20\$:	PUSHL	#1		1124
		0104	8F	BB	00161		PUSHR	#*M<R2,R8>		
	69		03	FB	00165		CALLS	#3, DBG\$NMATCH		
	07		50	E8	00168		BLBS	R0, 21\$		
			62	B5	0016B		TSTW	(R2)		1125
			03	13	0016D		BEQL	21\$		
			FF6E	31	0016F		BRW	11\$		
	01		55	D1	00172	21\$:	CMPL	RADIX, #1		1134
			0F	13	00175		BEQL	22\$		
			03	DD	00177		PUSHL	#3		1140
	6A		01	FB	00179		CALLS	#1, DBG\$GET_TEMP MEM		
	60		55	90	0017C		MOVB	RADIX, (ADVERB_NODE)		1142
04	A4		50	D0	0017F		MOVL	ADVERB_NODE, 4(R4)		1143
		08	A0	D4	00183		CLRL	8(ADVERB_NODE)		1144
	50		01	D0	00186	22\$:	MOVL	#1, R0		1147
			04	00189			RET			1149

; Routine Size: 394 bytes, Routine Base: DBG\$CODE + 044D

```

: 1022 1150 1 GLOBAL ROUTINE DBG$NPARSE_EXAMINE (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
: 1023 1151 1
: 1024 1152 1 ++
: 1025 1153 1 FUNCTIONAL DESCRIPTION:
: 1026 1154 1
: 1027 1155 1     ATN parse network for the EXAMINE verb. The network constructs a command
: 1028 1156 1     execution tree consisting of a linked list of command operands which the
: 1029 1157 1     execution network accepts as input to perform the command.
: 1030 1158 1
: 1031 1159 1     Three adverb nodes are ALWAYS created and linked. They are in the order:
: 1032 1160 1
: 1033 1161 1     1) Type node - This node contains override or default type info. It may
: 1034 1162 1     have the value of dummy_type when the source of the
: 1035 1163 1     EXAMINE is described by a primary descriptor and
: 1036 1164 1     no override type has been specified.
: 1037 1165 1
: 1038 1166 1     2) Radix node - The radix to be used in outputting the value.
: 1039 1167 1     For implementation level 2, DBG$K_DEFAULT is
: 1040 1168 1     translated to the real radix.
: 1041 1169 1
: 1042 1170 1
: 1043 1171 1     3) Mode node - Either true or false. True means symbolic output. False
: 1044 1172 1     means no symbolic output.
: 1045 1173 1
: 1046 1174 1 FORMAL PARAMETERS:
: 1047 1175 1
: 1048 1176 1     INPUT_DESC - A longword containing the address of the command
: 1049 1177 1     input descriptor
: 1050 1178 1
: 1051 1179 1     VERB_NODE - A longword containing the address of the command
: 1052 1180 1     execution tree verb (head) node
: 1053 1181 1
: 1054 1182 1     MESSAGE_VECT - The address of a longword to contain the address
: 1055 1183 1     of a standard message argument vector on errors
: 1056 1184 1
: 1057 1185 1 IMPLICIT INPUTS:
: 1058 1186 1
: 1059 1187 1     NONE
: 1060 1188 1
: 1061 1189 1 IMPLICIT OUTPUTS:
: 1062 1190 1
: 1063 1191 1     On success, the command execution tree is constructed.
: 1064 1192 1
: 1065 1193 1     On failure, a message argument vector is constructed or obtained.
: 1066 1194 1
: 1067 1195 1 ROUTINE VALUE:
: 1068 1196 1
: 1069 1197 1     An unsigned integer longword completion code
: 1070 1198 1
: 1071 1199 1 COMPLETION CODES:
: 1072 1200 1
: 1073 1201 1     sts$k_success (1) - Success. Command execution tree constructed.
: 1074 1202 1
: 1075 1203 1     sts$k_severe (4) - Failure. Error encountered. Message argument
: 1076 1204 1     constructed and returned.
: 1077 1205 1
: 1078 1206 1 SIDE EFFECTS:

```

```

: 1079      1207      1 |
: 1080      1208      1 |
: 1081      1209      1 |
: 1082      1210      1 |
: 1083      1211      1 |
: 1084      1212      1 |
: 1085      1213      1 |
: 1086      1214      1 |
: 1087      1215      1 |
: 1088      1216      1 |
: 1089      1217      1 |
: 1090      1218      1 |
: 1091      1219      1 |
: 1092      1220      1 |
: 1093      1221      1 |
: 1094      1222      1 |
: 1095      1223      1 |
: 1096      1224      1 |
: 1097      1225      1 |
: 1098      1226      1 |
: 1099      1227      1 |
: 1100      1228      1 |
: 1101      1229      1 |
: 1102      1230      1 |
: 1103      1231      1 |
: 1104      1232      1 |
: 1105      1233      1 |
: 1106      1234      1 |
: 1107      1235      1 |
: 1108      1236      1 |
: 1109      1237      1 |
: 1110      1238      1 |
: 1111      1239      1 |
: 1112      1240      1 |
: 1113      1241      1 |
: 1114      1242      1 |
: 1115      1243      1 |
: 1116      1244      1 |
: 1117      1245      1 |
: 1118      1246      1 |
: 1119      1247      1 |
: 1120      1248      1 |
: 1121      1249      1 |
: 1122      1250      1 |
: 1123      1251      1 |
: 1124      1252      1 |
: 1125      1253      1 |
: 1126      1254      1 |
: 1127      1255      1 |
: 1128      1256      1 |
: 1129      1257      1 |
: 1130      1258      1 |
: 1131      1259      1 |
: 1132      1260      1 |
: 1133      1261      1 |
: 1134      1262      1 |
: 1135      1263      1 |

NONE

BEGIN

MAP
  INPUT_DESC      : REF BLOCK [ ,BYTE],
  VERB_NODE       : REF dbg$verb_node;

BIND

  ! Strings used at this level of parsing
  DBG$CS_INSTRUCTION = UPLIT BYTE (11, 'INSTRUCTION'),
  DBG$CS_SYMBOLS     = UPLIT BYTE (8, 'SYMBOLIC'),
  DBG$CS_NOSYMBOLS  = UPLIT BYTE (10, 'NOSYMBOLIC'),
  DBG$CS_BINARY     = UPLIT BYTE (6, 'BINARY'),
  DBG$CS_OCTAL      = UPLIT BYTE (5, 'OCTAL'),
  DBG$CS_DECIMAL    = UPLIT BYTE (7, 'DECIMAL'),
  DBG$CS_HEXADecimal = UPLIT BYTE (11, 'HEXADecimal'),
  DBG$CS_DEFAULT    = UPLIT BYTE (7, 'DEFAULT'),
  DBG$CS_AC         = UPLIT BYTE (2, 'AC'),
  DBG$CS_ASCII     = UPLIT BYTE (5, 'ASCII'),
  DBG$CS_AD         = UPLIT BYTE (2, 'AD'),
  DBG$CS_ASCID     = UPLIT BYTE (5, 'ASCID'),
  DBG$CS_AW        = UPLIT BYTE (2, 'AW'),
  DBG$CS_ASCIIW    = UPLIT BYTE (5, 'ASCIIW'),
  DBG$CS_AZ        = UPLIT BYTE (2, 'AZ'),
  DBG$CS_ASCIIz    = UPLIT BYTE (5, 'ASCIIz'),
  DBG$CS_ASCII     = UPLIT BYTE (5, 'ASCII'),
  DBG$CS_CONDITION_VALUE = UPLIT BYTE (15, 'CONDITION_VALUE'),
  DBG$CS_PSL       = UPLIT BYTE (3, 'PSL'),
  DBG$CS_PSW       = UPLIT BYTE (3, 'PSW'),
  DBG$CS_BYTE      = UPLIT BYTE (4, 'BYTE'),
  DBG$CS_WORD      = UPLIT BYTE (4, 'WORD'),
  DBG$CS_LONGWORD  = UPLIT BYTE (8, 'LONG'),
  DBG$CS_QUADWORD  = UPLIT BYTE (8, 'QUADWORD'),
  DBG$CS_OCTAWORD  = UPLIT BYTE (8, 'OCTAWORD'),
  DBG$CS_FLOAT     = UPLIT BYTE (5, 'FLOAT'),
  DBG$CS_F_FLOAT   = UPLIT BYTE (7, 'F_FLOAT'),
  DBG$CS_D_FLOAT   = UPLIT BYTE (7, 'D_FLOAT'),
  DBG$CS_G_FLOAT   = UPLIT BYTE (7, 'G_FLOAT'),
  DBG$CS_H_FLOAT   = UPLIT BYTE (7, 'H_FLOAT'),
  DBG$CS_SOURCE    = UPLIT BYTE (6, 'SOURCE'),
  DBG$CS_PACKED    = UPLIT BYTE (6, 'PACKED'),
  DBG$CS_DATE_TIME = UPLIT BYTE (9, 'DATE_TIME'),
  DBG$CS_COMMA     = UPLIT BYTE (1, dbg$k_comma),
  DBG$CS_COLON     = UPLIT BYTE (1, dbg$k_colon),
  DBG$CS_SLASH     = UPLIT BYTE (1, dbg$k_slash),
  DBG$CS_CR        = UPLIT BYTE (1, dbg$k_car_return);

LOCAL
  ADDR_EXP_DESC   : REF dbg$aed,      ! Address expression descriptor
  NOUN_NODE       : REF dbg$noun_node, ! Object
  ADVERB_NODE     : REF dbg$adverb_node, ! Adverb

```

```

: 1136      1264      LINK,                               ! Pointer
: 1137      1265      SYM_OVERRIDE_PRESENT,
: 1138      1266      INP_RADIX,                       ! Input radix
: 1139      1267      RADIX,                           ! Radix for numeric literals
: 1140      1268      TYPE,                             ! Type of source
: 1141      1269      SYMBOL_FLAG,                     ! IF true then SYMBOLIC EXAMINE
: 1142      1270      LOWER_PC_FLAG,                   ! TRUE if we are parsing
: 1143      1271                                           ! the lower pc in the
: 1144      1272                                           ! pc range.
: 1145      1273      SOURCE_FLAG,                       ! TRUE for EX/SOURCE
: 1146      1274      MSG_TEXT,                          ! TRUE for EX/CONDITION_VALUE
: 1147      1275      LENGTH,                            ! Length of source
: 1148      1276      ADDRESS : VECTOR [2],             ! Address of source
: 1149      1277      STATUS;                            ! Return status
: 1150      1278
: 1151      1279      ! Create and link a noun node to contain the target
: 1152      1280      !
: 1153      1281      noun_node = dbg$get_tempmem (dbg$k_noun_node_size);
: 1154      1282
: 1155      1283      verb_node [dbg$l_verb_object_ptr] = .noun_node;
: 1156      1284
: 1157      1285
: 1158      1286      ! Initialize target type and length
: 1159      1287      !
: 1160      1288      type = -1;                          ! -1 indicates that we have not found an override type
: 1161      1289      length = 0;
: 1162      1290
: 1163      1291
: 1164      1292      ! Initialize the symbolic - nosymbolic flag and source flag
: 1165      1293      !
: 1166      1294      symbol_flag = dbg$nget_mode ();
: 1167      1295      sym_override_present = FALSE;
: 1168      1296      source_flag = FALSE;
: 1169      1297      msg_text = FALSE;
: 1170      1298
: 1171      1299      ! Initialize the lower_pc flag
: 1172      1300      !
: 1173      1301      lower_pc_flag = TRUE;
: 1174      1302
: 1175      1303      ! Initialize the radix.
: 1176      1304      !
: 1177      1305      inp_radix = .dbg$gb_radix[dbg$b_radix_input];
: 1178      1306      radix = .dbg$gb_radix[dbg$b_radix_output_over];
: 1179      1307
: 1180      1308      ! Accept any command switches. We start out with a verb composite representing
: 1181      1309      ! a plain EXAMINE. This may change as a result of command or user set overrides.
: 1182      1310      ! We changes the composite to EXAMINE_INSTRUCTION if we find /INSTRUCTION.
: 1183      1311      ! The composite is changed to EXAMINE_CONDITION_VALUE if /CONDITION_VALUE
: 1184      1312      ! is found.
: 1185      1313      !
: 1186      1314      verb_node [dbg$b_verb_composite] = examine;
: 1187      1315
: 1188      1316      WHILE dbg$nmatch (.input_desc, dbg$cs_slash, 1)
: 1189      1317      DO
: 1190      1318          BEGIN
: 1191      1319          SELECTONE true
: 1192      1320          OF

```

: 1193 1321 W
: 1194 1322 W
: 1195 1323 W
: 1196 1324 W
: 1197 1325 W
: 1198 1326 W
: 1199 1327 W
: 1200 1328 W
: 1201 1329 W
: 1202 1330 W
: 1203 1331 W
: 1204 1332 W
: 1205 1333 W
: 1206 1334 W
: 1207 1335 W
: 1208 1336 W
: 1209 1337 W
: 1210 1338 W
: 1211 1339 W
: 1212 1340 W
: 1213 1341 W
: 1214 1342 W
: 1215 1343 W
: 1216 1344 W
: 1217 1345 W
: 1218 1346 W
: 1219 1347 W
: 1220 1348 W
: 1221 1349 W
: 1222 1350 W
: 1223 1351 W
: 1224 1352 W
: 1225 1353 W
: 1226 1354 W
: 1227 1355 W
: 1228 1356 W
: 1229 1357 W
: 1230 1358 W
: 1231 1359 W
: 1232 1360 W
: 1233 1361 W
: 1234 1362 W
: 1235 1363 W
: 1236 1364 W
: 1237 1365 W
: 1238 1366 W
: 1239 1367 W
: 1240 1368 W
: 1241 1369 W
: 1242 1370 W
: 1243 1371 W
: 1244 1372 W
: 1245 1373 W
: 1246 1374 W
: 1247 1375 W
: 1248 1376 W
: 1249 1377 W

```
SET
[dbg$match (.input_desc, dbg$cs_instruction, 1)] :
BEGIN
type = dsc$k_dtype_zi;
length = 0;
radix = dbg$k_default;
END;

[dbg$match (.input_desc, dbg$cs_byte, 1)] :
BEGIN
type = dsc$k_dtype_b;
length = byte_length;
END;

[dbg$match (.input_desc, dbg$cs_word, 1)] :
BEGIN
type = dsc$k_dtype_w;
length = word_length;
END;

[dbg$match (.input_desc, dbg$cs_condition_value, 1)]:
BEGIN
type = dsc$k_dtype_l;
length = long_length;
msg_text = TRUE;
END;

[dbg$match (.input_desc, dbg$cs_date_time, 2)]:
BEGIN
type = dsc$k_dtype_adt;
length = 8;
END;

[dbg$match (.input_desc, dbg$cs_psl, 3)]:
BEGIN
type = dsc$k_dtype_lu;
length = long_length;
verb_node [dbg$b_verb_composite] = examine_psl;
END;

[dbg$match (.input_desc, dbg$cs_psw, 3)]:
BEGIN
type = dsc$k_dtype_wu;
length = word_length;
verb_node [dbg$b_verb_composite] = examine_psw;
END;

[dbg$match (.input_desc, dbg$cs_longword, 1)] :
BEGIN
type = dsc$k_dtype_l;
length = long_length;
END;

[dbg$match (.input_desc, dbg$cs_ac, 2),
dbg$match (.input_desc, dbg$cs_ascic, 5)]:
BEGIN
```

: A3.1
: A3.1
: A3.1 M3.3
: A3.1
: A3.1

```

: 1250      1378      4      type = dsc$K_dtype ac;
: 1251      1379      4      length = byte_length;
: 1252      1380      4      radix = dbg$K_default;
: 1253      1381      4      END;
: 1254      1382      4
: 1255      1383      4      [dbg$Nmatch (.input_desc, dbg$cs_3w, 2),
: 1256      1384      4      dbg$Nmatch (.input_desc, dbg$cs_asciw, 5)]:
: 1257      1385      4      BEGIN
: 1258      1386      4      type = dsc$K_dtype vt;
: 1259      1387      4      length = word_length;
: 1260      1388      4      radix = dbg$K_default;
: 1261      1389      4      END;
: 1262      1390      4
: 1263      1391      4      [dbg$Nmatch (.input_desc, dbg$cs_az, 2),
: 1264      1392      4      dbg$Nmatch (.input_desc, dbg$cs_asciz, 5)]:
: 1265      1393      4      BEGIN
: 1266      1394      4      type = dsc$K_dtype az;
: 1267      1395      4      length = byte_length;
: 1268      1396      4      radix = dbg$K_default;
: 1269      1397      4      END;
: 1270      1398      4
: 1271      1399      4      [dbg$Nmatch (.input_desc, dbg$cs_ad, 2),
: 1272      1400      4      dbg$Nmatch (.input_desc, dbg$cs_ascid, 5)]:
: 1273      1401      4      BEGIN
: 1274      1402      4      type = dbg$K_dtype ad;
: 1275      1403      4      length = 2 * long_length;
: 1276      1404      4      radix = dbg$K_default;
: 1277      1405      4      END;
: 1278      1406      4
: 1279      1407      4      [dbg$Nmatch (.input_desc, dbg$cs_ascii, 1)] :
: 1280      1408      4      BEGIN
: 1281      1409      4
: 1282      1410      4      ! Accept the ':' and integer. If there is no colon, we assume
: 1283      1411      4      ! a length of 4
: 1284      1412      4      !
: 1285      1413      4      IF NOT dbg$Nmatch (.input_desc, dbg$cs_colon, 1)
: 1286      1414      4      THEN
: 1287      1415      5      BEGIN
: 1288      1416      5      length = 0;
: 1289      1417      5      END
: 1290      1418      4      ELSE
: 1291      1419      5      BEGIN
: 1292      1420      5
: 1293      1421      5      ! We found the colon. Accept the integer
: 1294      1422      5      !
: 1295      1423      5      IF NOT dbg$Nsave_integer (.input_desc, length)
: 1296      1424      5      THEN
: 1297      1425      5      RETURN sts$K_severe;
: 1298      1426      5
: 1299      1427      5      ! Check to see that the length of the ascii examine is not 0
: 1300      1428      5      !
: 1301      1429      5      IF .length GTRU 2048
: 1302      1430      5      THEN
: 1303      1431      6      BEGIN
: 1304      1432      6      OWN
: 1305      1433      6      num_desc : dbg$stg_desc;
: 1306      1434      6

```

```

: 1307      1435      6      num_desc [dsc$w_length] = 1;
: 1308      1436      6      num_desc [dsc$a_pointer] = UPLIT TYPE ('0');
: 1309      1437      6      .message_vect = dbg$make_arg_vect (dbg$_invnumber, 1, num_desc);
: 1310      1438      6      RETURN sfs$k_severe;
: 1311      1439      5      END;
: 1312      1440      4      END;
: 1313      1441      4
: 1314      1442      4      type = dsc$k_dtype_t;
: 1315      1443      4      radix = dbg$k_default;
: 1316      1444      4      END;
: 1317      1445      4
: 1318      1446      4      [dbg$match (.input_desc, dbg$cs_default, 3)] :
: 1319      1447      4      BEGIN
: 1320      1448      4      radix = dbg$k_default;
: 1321      1449      4      END;
: 1322      1450      4
: 1323      1451      4      [dbg$match (.input_desc, dbg$cs_binary, 2)] :
: 1324      1452      4      BEGIN
: 1325      1453      4      radix = dbg$k_binary;
: 1326      1454      4      END;
: 1327      1455      4
: 1328      1456      4      [dbg$match (.input_desc, dbg$cs_octal, 1)] :
: 1329      1457      4      BEGIN
: 1330      1458      4      radix = dbg$k_octal;
: 1331      1459      4      END;
: 1332      1460      4
: 1333      1461      4      [dbg$match (.input_desc, dbg$cs_d_float, 2)] :
: 1334      1462      4      BEGIN
: 1335      1463      4      type = dsc$k_dtype_d;
: 1336      1464      4      length = 8;
: 1337      1465      4      radix = dbg$k_default;
: 1338      1466      4      END;
: 1339      1467      4
: 1340      1468      4      [dbg$match (.input_desc, dbg$cs_decimal, 1)] :
: 1341      1469      4      BEGIN
: 1342      1470      4      radix = dbg$k_decimal;
: 1343      1471      4      END;
: 1344      1472      4
: 1345      1473      4      [dbg$match (.input_desc, dbg$cs_hexadecimal, 1)] :
: 1346      1474      4      BEGIN
: 1347      1475      4      radix = dbg$k_hex;
: 1348      1476      4      END;
: 1349      1477      4
: 1350      1478      4      [dbg$match (.input_desc, dbg$cs_symbols, 1)] :
: 1351      1479      4      BEGIN
: 1352      1480      4      symbol_flag = true;
: 1353      1481      4      sym_override_present = TRUE;
: 1354      1482      4      END;
: 1355      1483      4
: 1356      1484      4      [dbg$match (.input_desc, dbg$cs_nosymbols, 1)] :
: 1357      1485      4      BEGIN
: 1358      1486      4      symbol_flag = false;
: 1359      1487      4      sym_override_present = TRUE;
: 1360      1488      4      END;
: 1361      1489      4
: 1362      1490      4      [dbg$match (.input_desc, dbg$cs_source, 2)] : ! EX/SOURCE
: 1363      1491      4      BEGIN

```

```

: 1364      1492      4      source_flag = TRUE;
: 1365      1493      4      END;
: 1366      1494      3
: 1367      1495      3      [dbg$nmatch (.input_desc, dbg$cs_quadword, 1)] :
: 1368      1496      4      BEGIN
: 1369      1497      4      type = dsc$k_dtype_q;
: 1370      1498      4      length = 8;
: 1371      1499      4      END;
: 1372      1500      3
: 1373      1501      4      [dbg$nmatch (.input_desc, dbg$cs_octaword, 5)] :
: 1374      1502      4      BEGIN
: 1375      1503      4      type = dsc$k_dtype_o;
: 1376      1504      4      length = 16;
: 1377      1505      4      END;
: 1378      1506      3
: 1379      1507      3      [dbg$nmatch (.input_desc, dbg$cs_float, 1)
: 1380      1508      3      dbg$nmatch (.input_desc, dbg$cs_f_float, 1)] :
: 1381      1509      4      BEGIN
: 1382      1510      4      type = dsc$k_dtype_f;
: 1383      1511      4      length = 4;
: 1384      1512      4      radix = dbg$k_default;
: 1385      1513      4      END;
: 1386      1514      3
: 1387      1515      4      [dbg$nmatch (.input_desc, dbg$cs_g_float, 1)] :
: 1388      1516      4      BEGIN
: 1389      1517      4      type = dsc$k_dtype_g;
: 1390      1518      4      length = 8;
: 1391      1519      4      radix = dbg$k_default;
: 1392      1520      4      END;
: 1393      1521      3
: 1394      1522      4      [dbg$nmatch (.input_desc, dbg$cs_h_float, 2)] :
: 1395      1523      4      BEGIN
: 1396      1524      4      type = dsc$k_dtype_h;
: 1397      1525      4      length = 16;
: 1398      1526      4      radix = dbg$k_default;
: 1399      1527      4      END;
: 1400      1528      3
: 1401      1529      4      [dbg$nmatch (.input_desc, dbg$cs_packed, 2)] :
: 1402      1530      4      BEGIN
: 1403      1531      4      type = dsc$k_dtype_p;
: 1404      1532      4      radix = dbg$k_default;
: 1405      1533      4
: 1406      1534      4      ! Accept the ':' and integer. If there is no colon, use
: 1407      1535      4      ! a length or -1 to signal that we will have to find the length
: 1408      1536      4      ! after its been converted from text.
: 1409      1537      4      !
: 1410      1538      4      IF NOT dbg$nmatch (.input_desc, dbg$cs_colon, 1)
: 1411      1539      4      THEN
: 1412      1540      5      BEGIN
: 1413      1541      5      length = -1;
: 1414      1542      5      END
: 1415      1543      4      ELSE
: 1416      1544      5      BEGIN
: 1417      1545      5
: 1418      1546      5      ! We found the colon. Accept the integer
: 1419      1547      5      !
: 1420      1548      5      IF NOT dbg$nsave_integer (.input_desc, length)

```

```

: 1421          1549          S
: 1422          1550          S
: 1423          1551          S
: 1424          1552          S
: 1425          1553          S
: 1426          1554          S
: 1427          1555          S
: 1428          1556          S
: 1429          1557          S
: 1430          1558          S
: 1431          1559          S
: 1432          1560          S
: 1433          1561          S
: 1434          1562          S
: 1435          1563          S
: 1436          1564          S
: 1437          1565          S
: 1438          1566          S
: 1439          1567          S
: 1440          1568          S
: 1441          1569          S
: 1442          1570          S
: 1443          1571          S
: 1444          1572          S
: 1445          1573          S
: 1446          1574          S
: 1447          1575          S
: 1448          1576          S
: 1449          1577          S
: 1450          1578          S
: 1451          1579          S
: 1452          1580          S
: 1453          1581          S
: 1454          1582          S
: 1455          1583          S
: 1456          1584          S
: 1457          1585          S
: 1458          1586          S
: 1459          1587          S
: 1460          1588          S
: 1461          1589          S
: 1462          1590          S
: 1463          1591          S
: 1464          1592          S
: 1465          1593          S
: 1466          1594          S
: 1467          1595          S
: 1468          1596          S
: 1469          1597          S
: 1470          1598          S
: 1471          1599          S
: 1472          1600          S
: 1473          1601          S
: 1474          1602          S
: 1475          1603          S
: 1476          1604          S
: 1477          1605          S

      THEN
      RETURN sts$sk_severe;

      ! Check to see that the length is within 0 and 31
      ! IF (.length LSS 0) OR (.length GTR 31)
      THEN
      SIGNAL(dbg$_illpacsiz, 1, .length);
      END;
    END;

    [OTHERWISE] :      ! Some flavor of user error
    BEGIN
      .message_vect =
      ( IF dbg$match (.input_desc, dbg$cs_cr, 1)
      THEN
      dbg$make_arg_vect (dbg$_needmore)
      ELSE
      dbg$syntax_error (dbg$next_word (.input_desc)));
    RETURN sts$sk_severe;
    END;

  TES;
END;

IF .sym_override_present
THEN
BEGIN
  dbg$set_mod_lvl (override_mode);
  dbg$gb_mod_ptr[mode_symbol[s]] = .symbol_flag;
END;

! Check for EXAMINE <CR>. If this is the case, we call the pseudo symbol
! interpreter to find the meaning of successor ourselves.
! IF dbg$match (.input_desc, dbg$cs_cr, 1)
! THEN
! BEGIN
! LOCAL
!   AE_DESC      : REF dbg$aed, ! Address expression descriptor
!   SYMBOL_TYPE,
!   SYMBOL_VALUE,
!   TYPE,
!   LENGTH;
! MAP
!   SYMBOL_VALUE: REF DBG$VALDESC;
! LOCAL
!   DUMMY,
!   KIND;
! ! Look up the symbol in the DEFINE symbol table.
! ! IF DBG$DEF_SYM_FIND (UPLIT BYTE (%ASCIC '%NEXTLOC')),

```

```

! A3.1
! A3.1
! A3.1
! A3.1
! A3.1
! A3.1
! A3.1
! A3.1
! A3.1
! A3.1

```

```

: 1478 1606 3
: 1479 1607 3
: 1480 1608 3
: 1481 1609 3
: 1482 1610 3
: 1483 1611 3
: 1484 1612 4
: 1485 1613 4
: 1486 1614 4
: 1487 1615 4
: 1488 1616 4
: 1489 1617 4
: 1490 1618 5
: 1491 1619 5
: 1492 1620 5
: 1493 1621 5
: 1494 1622 5
: 1495 1623 5
: 1496 1624 5
: 1497 1625 5
: 1498 1626 5
: 1499 1627 5
: 1500 1628 5
: 1501 1629 4
: 1502 1630 4
: 1503 1631 4
: 1504 1632 4
: 1505 1633 4
: 1506 1634 4
: 1507 1635 4
: 1508 1636 4
: 1509 1637 4
: 1510 1638 4
: 1511 1639 3
: 1512 1640 3
: 1513 1641 3
: 1514 1642 3
: 1515 1643 3
: 1516 1644 3
: 1517 1645 3
: 1518 1646 3
: 1519 1647 3
: 1520 1648 3
: 1521 1649 3
: 1522 1650 3
: 1523 1651 3
: 1524 1652 3
: 1525 1653 3
: 1526 1654 3
: 1527 1655 3
: 1528 1656 3
: 1529 1657 3
: 1530 1658 3
: 1531 1659 3
: 1532 1660 3
: 1533 1661 3
: 1534 1662 3

```

```

                                KIND, SYMBOL VALUE,
                                DUMMY, DUMMY)
THEN
  IF .KIND EQL DEFINE_ADDRESS
  OR .KIND EQL DEFINE_VALUE
  THEN
    BEGIN
      ! Turn value descriptors into volatile value descriptors.
      !
      IF .SYMBOL_VALUE [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
      THEN
        BEGIN
          SYMBOL_VALUE [DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
          SYMBOL_VALUE [DBG$L_VALUE_POINTER] =
            .SYMBOL_VALUE [DBG$L_VALUE_VALUE0];
          IF .SYMBOL_VALUE [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_V
          OR .SYMBOL_VALUE [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_VU
          OR .SYMBOL_VALUE [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SV
          OR .SYMBOL_VALUE [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SVU
          THEN
            SYMBOL_VALUE [DBG$L_VALUE_POS] =
              .SYMBOL_VALUE [DBG$L_VALUE_VALUE1];
          END;
          ! We have found a matching DEFINEd symbol.
          ! Copy the descriptor into temporary memory.
          ! (Fourth parameter FALSE <-> copy into tempmem).
          !
          DBG$N_COPY_DESC (.SYMBOL_VALUE, SYMBOL_VALUE,
            DUMMY, FALSE);
        END
      ELSE
        SIGNAL (DBG$_NOCURLOC)
      ELSE
        SIGNAL (DBG$_NOCURLOC);
      noun_node [dbg$l_noun_value] = .symbol_value;
      noun_node [dbg$l_noun_value2] = .symbol_value;
    END
  ELSE
    ! Loop through the examine list
    WHILE TRUE DO
      BEGIN
        ! Check for exhausted input
        IF dbg$match (.input_desc, dbg$cs cr, 1)
        OR .input_desc[ds$c$w_length] EQL 0
        THEN
          EXITLOOP;
        ! Input left. Let the AEI deal with it.

```

```

: 1535
: 1536
: 1537
: 1538
: 1539
: 1540
: 1541
: 1542
: 1543
: 1544
: 1545
: 1546
: 1547
: 1548
: 1549
: 1550
: 1551
: 1552
: 1553
: 1554
: 1555
: 1556
: 1557
: 1558
: 1559
: 1560
: 1561
: 1562
: 1563
: 1564
: 1565
: 1566
: 1567
: 1568
: 1569
: 1570
: 1571
: 1572
: 1573
: 1574
: 1575
: 1576
: 1577
: 1578
: 1579
: 1580
: 1581
: 1582
: 1583
: 1584
: 1585
: 1586
: 1587
: 1588
: 1589
: 1590
: 1591

```

```

!
IF .LOWER_PC_FLAG
THEN
BEGIN
STATUS = DBG$NPARSE ADDRESS(.INPUT_DESC,
NOUN_NODE [DBG$L_NOUN_VALUE], .INP_RADIX,
TOKEN$K_TERM_COMCOL, .MESSAGE_VECT);
NOUN_NODE[DBG$L_NOUN_VALUE2] = .NOUN_NODE[DBG$L_NOUN_VALUE];
END
ELSE
STATUS = DBG$NPARSE ADDRESS(.INPUT_DESC,
NOUN_NODE [DBG$L_NOUN_VALUE2], .INP_RADIX,
TOKEN$K_TERM_COMMA, .MESSAGE_VECT);

IF .status NEQ sts$k_success
THEN
IF .status EQL sts$k_warning
THEN
BEGIN
! Check for comma.
IF dbg$nmatch (.input_desc, dbg$cs_comma, 1)
THEN
BEGIN
! Check for end of line after the comma.
IF dbg$nmatch (.input_desc, dbg$cs_cr, 1)
THEN
SIGNAL(DBG$_NEEDMORE);

! Link in another noun node
!
noun_node[dbg$l_noun_link] = dbg$get_tempmem (dbg$k_noun_node_size);
noun_node = .noun_node[dbg$l_noun_link];

! Set LOWER_PC_FLAG back to TRUE for the next
! PC range in the list.
!
lower_pc_flag = TRUE;
END
ELSE
IF .lower_pc_flag
THEN
IF dbg$nmatch (.input_desc, dbg$cs_colon, 1)
THEN
lower_pc_flag = FALSE
ELSE
BEGIN
.message_vect = dbg$nsyntax_error(
dbg$next_word (.input_desc));
RETURN sts$k_severe;

```

```

: 1592      1720      4
: 1593      1721      4
: 1594      1722      4
: 1595      1723      4
: 1596      1724      4
: 1597      1725      4
: 1598      1726      4
: 1599      1727      4
: 1600      1728      4
: 1601      1729      4
: 1602      1730      4
: 1603      1731      4
: 1604      1732      4
: 1605      1733      4
: 1606      1734      4
: 1607      1735      4
: 1608      1736      4
: 1609      1737      4
: 1610      1738      4
: 1611      1739      4
: 1612      1740      4
: 1613      1741      4
: 1614      1742      4
: 1615      1743      4
: 1616      1744      4
: 1617      1745      4
: 1618      1746      4
: 1619      1747      4
: 1620      1748      4
: 1621      1749      4
: 1622      1750      4
: 1623      1751      4
: 1624      1752      4
: 1625      1753      4
: 1626      1754      4
: 1627      1755      4
: 1628      1756      4
: 1629      1757      4
: 1630      1758      4
: 1631      1759      4
: 1632      1760      4
: 1633      1761      4
: 1634      1762      4
: 1635      1763      4
: 1636      1764      4
: 1637      1765      4
: 1638      1766      4
: 1639      1767      4
: 1640      1768      4
: 1641      1769      4
: 1642      1770      4
: 1643      1771      4
: 1644      1772      4
: 1645      1773      4
: 1646      1774      4
: 1647      1775      4
: 1648      1776      4

```

```

                                END;
                                END ! check for comma
ELSE
    RETURN sts$k_severe;

! Set up for next time around loop
END; ! While loop

! The first adverb node we set up is the type node. We start by setting up a
! dummy type. If we find a real type, we change the adverb value.
! We check to see if type is instruction. If so, we change the verb
! composite to an EXAMINE_INSTRUCTION.
! Override types and default types apply, as well as command override
! types.
adverb_node = dbg$get_tempmem (dbg$k_adverb_node_size);
verb_node [dbg$l_verb_adverb_ptr] = .adverb_node;
adverb_node [dbg$b_adverb_literal] = dummy_type;

! Obtain the address expression descriptor
addr_exp_desc = .noun_node [dbg$l_noun_value];

! Check for command override
IF .type NEQ -1
THEN
    BEGIN
        adverb_node [dbg$b_adverb_literal] = .type;
        adverb_node [dbg$l_adverb_value] = .length;

        ! Check for EX/CONDITION_VALUE.
        IF .msg_text
        THEN
            verb_node [dbg$b_verb_composite] = examine_condition_value;
        END
    END
ELSE
    BEGIN
        ! There was no command override. Check for a type/override.
        IF dbg$nget_override_type (type, length)
        THEN
            BEGIN
                adverb_node [dbg$b_adverb_literal] = .type;
            END
        END
    END

```

```

: 1649      1777  4      adverb_node [dbg$l_adverb_value] = .length;
: 1650      1778  4      END
: 1651      1779  3      ELSE
: 1652      1780  4      BEGIN
: 1653      1781  4
: 1654      1782  4      ! There was no type override. We supply a type based on the contents
: 1655      1783  4      ! of the address expression descriptor, current location type,
: 1656      1784  4      ! and/or default type. We may also change the verb composite value.
: 1657      1785  4      !
: 1658      1786  4      ! Check for a current loc type. If this is other than 0 or notype,
: 1659      1787  4      ! then we use this type for display.
: 1660      1788  4      !
: 1661      1789  4      dbg$nget_potential_type (type, length);
: 1662      1790  4
: 1663      1791  4      IF .type NEQ dbg$k_notype AND .type NEQ 0
: 1664      1792  4      THEN
: 1665      1793  5      BEGIN
: 1666      1794  5      adverb_node [dbg$b_adverb_literal] = .type;
: 1667      1795  5      adverb_node [dbg$l_adverb_value] = .length;
: 1668      1796  4      END;
: 1669      1797  3      END;
: 1670      1798  2      END;
: 1671      1799  2
: 1672      1800  2
: 1673      1801  2      ! The adverb type node has been set up, and the command has been classified
: 1674      1802  2      ! as either EXAMINE, EXAMINE_INSTRUCTION, EXAMINE_CONDITION_VALUE, or
: 1675      1803  2      ! EXAMINE_REGISTER. The source has been placed in the first noun node.
: 1676      1804  2      ! Now we construct adverb nodes for radix and mode.
: 1677      1805  2      !
: 1678      1806  2      link = adverb_node [dbg$l_adverb link];
: 1679      1807  2      adverb_node = dbg$get_tempmem (dbg$k_adverb_node_size);
: 1680      1808  2
: 1681      1809  2      .link = .adverb_node;
: 1682      1810  2      adverb_node [dbg$b_adverb_literal] = .radix;
: 1683      1811  2      dbg$gl_cmd_radix = .radix;
: 1684      1812  2
: 1685      1813  2      ! Create as adverb node for the mode (symbolic or nosymbolic).
: 1686      1814  2      !
: 1687      1815  2      link = adverb_node [dbg$l_adverb link];
: 1688      1816  2      adverb_node = dbg$get_tempmem (dbg$k_adverb_node_size);
: 1689      1817  2
: 1690      1818  2      .link = .adverb_node;
: 1691      1819  2      adverb_node [dbg$b_adverb_literal] = .symbol_flag;
: 1692      1820  2
: 1693      1821  2      IF .source_flag
: 1694      1822  2      THEN
: 1695      1823  2      verb_node[dbg$b_verb_composite] = examine_source;
: 1696      1824  2
: 1697      1825  2      RETURN sts$k_success;
: 1698      1826  1      END;

```

.PSECT DBG\$PLIT, NOWRT, SHR, PIC, 0

4E 4F 49 54 43 55 52 54 53 4E 0B 000E7 P.ABL: .BYTE 11
49 000E8 .ASCII \INSTRUCTION\ :

									08	000F3	P.ABM:	.BYTE	8														
		43	49	4C	4F	42	4D	59	53	000F4		.ASCII	\SYMBOLIC\														
									0A	000FC	P.ABN:	.BYTE	10														
	43	49	4C	4F	42	4D	59	53	4F	4E	000FD		.ASCII	\NOSYMBOLIC\													
									06	00107	P.ABO:	.BYTE	6														
					59	52	41	4E	49	42	00108		.ASCII	\BINARY\													
									05	0010E	P.ABP:	.BYTE	5														
					4C	41	54	43	4F	0010F		.ASCII	\OCTAL\														
									07	00114	P.ABQ:	.BYTE	7														
					4C	41	4D	49	43	45	44	00115		.ASCII	\DECIMAL\												
									0B	0011C	P.ABR:	.BYTE	11														
	4C	41	4D	49	43	45	44	41	58	45	48	0011D		.ASCII	\HEXADECIMAL\												
									07	00128	P.ABS:	.BYTE	7														
					54	4C	55	41	46	45	44	00129		.ASCII	\DEFAULT\												
									02	00130	P.ABT:	.BYTE	2														
									43	41	00131		.ASCII	\AC\													
									05	0C133	P.ABU:	.BYTE	5														
						43	49	43	53	41	00134		.ASCII	\ASCIC\													
									02	00139	P.ABV:	.BYTE	2														
									44	41	0013A		.ASCII	\AD\													
									05	0013C	P.ABW:	.BYTE	5														
						44	49	43	53	41	0013D		.ASCII	\ASCID\													
									02	00142	P.ABX:	.BYTE	2														
									57	41	00143		.ASCII	\AW\													
									05	00145	P.ABY:	.BYTE	5														
						57	49	43	53	41	00146		.ASCII	\ASCIW\													
									02	0014B	P.ABZ:	.BYTE	2														
									5A	41	0014C		.ASCII	\AZ\													
									05	0014E	P.ACA:	.BYTE	5														
						5A	49	43	53	41	0014F		.ASCII	\ASCIZ\													
									05	00154	P.ACB:	.BYTE	5														
						49	49	43	53	41	00155		.ASCII	\ASCI\													
									0F	0015A	P.ACC:	.BYTE	15														
45	55	4C	41	56	5F	4E	4F	49	54	49	44	4E	4F	43	0015B		.ASCII	\CONDITION_VALUE\									
														03	0016A	P.ACD:	.BYTE	3									
														4C	53	50	0016B		.ASCII	\PSL\							
																03	0016E	P.ACE:	.BYTE	3							
														57	53	50	0016F		.ASCII	\PSW\							
																04	00172	P.ACF:	.BYTE	4							
														45	54	59	42	00173		.ASCII	\BYTE\						
																04	00177	P.ACG:	.BYTE	4							
														44	52	4F	57	00178		.ASCII	\WORD\						
																08	0017C	P.ACH:	.BYTE	8							
														47	4E	4F	4C	0017D		.ASCII	\LONG\						
																08	00181	P.ACI:	.BYTE	8							
														44	52	4F	57	44	41	55	51	00182		.ASCII	\QUADWORD\		
																08	0018A	P.ACJ:	.BYTE	8							
														44	52	4F	57	41	54	43	4F	0018B		.ASCII	\OCTAWORD\		
																05	00193	P.ACK:	.BYTE	5							
																54	41	4F	4C	46	46	00194		.ASCII	\FLOAT\		
																07	00199	P.ACL:	.BYTE	7							
																54	41	4F	4C	46	5F	46	0019A		.ASCII	\F_FLOAT\	
																						07	001A1	P.ACM:	.BYTE	7	
																54	41	4F	4C	46	5F	44	001A2		.ASCII	\D_FLOAT\	
																						07	001A9	P.ACN:	.BYTE	7	
																54	41	4F	4C	46	5F	47	001AA		.ASCII	\G_FLOAT\	
																						07	001B1	P.ACO:	.BYTE	7	

.....

	54	41	4F	4C	46	5F	48	001B2		.ASCII	\H_FLOAT\	
							06	001B9	P.ACP:	.BYTE	6	
		45	43	52	55	4F	53	001BA		.ASCII	\SOURCE\	
							06	001C0	P.ACQ:	.BYTE	6	
		44	45	4B	43	41	50	001C1		.ASCII	\PACKED\	
							09	001C7	P.ACR:	.BYTE	9	
45	4D	49	54	5F	45	54	41	44	001C8		.ASCII	\DATE_TIME\
							2C	01	001D1	P.ACS:	.BYTE	1, 44
							3A	01	001D3	P.ACT:	.BYTE	1, 58
							2F	01	001D5	P.ACU:	.BYTE	1, 47
							0D	01	001D7	P.ACV:	.BYTE	1, 13
					00	00	00	30	001D9		.BLKB	3
					00	00	00	30	001DC	P.ACW:	.ASCII	\0\<0><0><0>
43	4F	4C	54	58	45	4E	25	08	001E0	P.ACX:	.ASCII	<8>\%NEXTLOC\

.PSECT DBGSOWN,NOEXE, PIC,2

0000C NUM_DESC:
.BLKB 12

DBGSCS_INSTRUCTION= P.ABL
 DBGSCS_SYMBOLS= P.ABM
 DBGSCS_NOSYMBOLS= P.ABN
 DBGSCS_BINARY= P.ABO
 DBGSCS_OCTAL= P.ABP
 DBGSCS_DECIMAL= P.ABQ
 DBGSCS_HEXADecimal= P.ABR
 DBGSCS_DEFAULT= P.ABS
 DBGSCS_AC= P.ABT
 DBGSCS_ASCIC= P.ABU
 DBGSCS_AD= P.ABV
 DBGSCS_ASCID= P.ABW
 DBGSCS_AW= P.ABX
 DBGSCS_ASCIW= P.ABY
 DBGSCS_AZ= P.ABZ
 DBGSCS_ASCIZ= P.ACA
 DBGSCS_ASCII= P.ACB
 DBGSCS_CONDITION_VALUE=
 P.ACC
 DBGSCS_PSL= P.ACD
 DBGSCS_PSW= P.ACE
 DBGSCS_BYTE= P.ACF
 DBGSCS_WORD= P.ACG
 DBGSCS_LONGWORD= P.ACH
 DBGSCS_QUADWORD= P.ACI
 DBGSCS_OCTAWORD= P.ACJ
 DBGSCS_FLOAT= P.ACK
 DBGSCS_F_FLOAT= P.ACL
 DBGSCS_D_FLOAT= P.ACM
 DBGSCS_G_FLOAT= P.ACN
 DBGSCS_H_FLOAT= P.ACO
 DBGSCS_SOURCE= P.ACP
 DBGSCS_PACKED= P.ACQ
 DBGSCS_DATE_TIME= P.ACR
 DBGSCS_COMMA= P.ACS
 DBGSCS_COLON= P.ACT
 DBGSCS_SLASH= P.ACU

DBG\$CS_CR= P.ACv

				.PSECT	DBG\$CODE,NOWRT, SHR, PIC,0	
				.ENTRY	DBG\$NPARSE EXAMINE, Save R2,R3,R4,R5,R6,R7,-;	1150
					R8,R9,R10,R11	
				SUBL2	#32, \$P	
00000000G	00		04 DD 00005	PUSHL	#4	1281
	53		01 FB 00007	CALLS	#1, DBG\$GET TEMPMEM	
	55	08	50 DO 0000E	MOVL	R0, NOUN_NODE	1283
08	A5		53 DO 00015	MOVL	VERB_NODE, R5	
14	AE		01 CE 00019	MNEGL	#1, TYPE	1288
		10	AE D4 0001D	CLRL	LENGTH	1289
00000000G	00		00 FB 00020	CALLS	#0, DBG\$NGET MODE	1294
	59		50 DO 00027	MOVL	R0, SYMBOL_FLAG	
			56 D4 0002A	CLRL	SYM_OVERRIDE_PRESENT	1295
			5A 7C 0002C	CLRQ	MSG_TEXT	1297
	58		01 DO 0002E	MOVL	#1, LOWER_PC_FLAG	1301
	6E	00000000G	00 9A 00031	MOVZBL	DBG\$GB_RADIX, INP_RADIX	1305
	54	00000000G	00 9A 00038	MOVZBL	DBG\$GB_RADIX+2, RADIX	1306
01	A5		01 90 0003F	MOVB	#1, 1(R5)	1314
	52	04	AC DO 00043	MOVL	INPUT_DESC, R2	1316
			01 DD 00047	PUSHL	#1	
		00000000'	EF 9F 00049	PUSHAB	DBG\$CS_SLASH	
			52 DD 0004F	PUSHL	R2	
00000000G	00		03 FB 00051	CALLS	#3, DBG\$NMATCH	
	03		50 E8 00058	BLBS	R0, 2\$	
		049F	31 0005B	BRW	59\$	
			01 DD 0005E	PUSHL	#1	1323
		00000000'	EF 9F 00060	PUSHAB	DBG\$CS_INSTRUCTION	
			52 DD 00066	PUSHL	R2	
00000000G	00		03 FB 00068	CALLS	#3, DBG\$NMATCH	
	01		50 D1 0006F	CPL	R0, #1	
			0A 12 00072	BNEQ	3\$	
14	AE		16 DO 00074	MOVL	#22, TYPE	1325
		10	AE D4 00078	CLRL	LENGTH	1326
		03EB	31 0007B	BRW	50\$	1327
			01 DD 0007E	PUSHL	#1	1330
		00000000'	EF 9F 00080	PUSHAB	DBG\$CS_BYTE	
			52 DD 00086	PUSHL	R2	
00000000G	00		03 FB 00088	CALLS	#3, DBG\$NMATCH	
	01		50 D1 0008F	CPL	R0, #1	
			0A 12 00092	BNEQ	4\$	
14	AE		06 DO 00094	MOVL	#6, TYPE	1332
10	AE		01 DO 00098	MOVL	#1, LENGTH	1333
			A9 11 0009C	BRB	1\$	1319
			01 DD 0009E	PUSHL	#1	1336
		00000000'	EF 9F 000A0	PUSHAB	DBG\$CS_WORD	
			52 DD 000A6	PUSHL	R2	
00000000G	00		03 FB 000A8	CALLS	#3, DBG\$NMATCH	
	01		50 D1 000AF	CPL	R0, #1	
			0A 12 000B2	BNEQ	6\$	
14	AE		07 DO 000B4	MOVL	#7, TYPE	1338
10	AE		02 DO 000B8	MOVL	#2, LENGTH	1339
			89 11 000BC	BRB	1\$	1319

		00000000'	01 DD 000BE 6\$:	PUSHL #1		1342
			FF 9F 000C0	PUSHAB DBG\$CS_CONDITION_VALUE		
00000000G	00		52 DD 000C6	PUSHL R2		
	01		03 FB 000C8	CALLS #3, DBG\$NMATCH		
			50 D1 000CF	CMPL R0, #1		
			0D 12 000D2	BNEQ 7\$		
14	AE		08 D0 000D4	MOVL #8, TYPE	1344	
10	AE		04 D0 000D8	MOVL #4, LENGTH	1345	
	5A		01 D0 000DC	MOVL #1, MSG_TEXT	1346	
			63 11 000DF	BRB 10\$	1319	
		00000000'	02 DD 000E1 7\$:	PUSHL #2	1349	
			FF 9F 000E3	PUSHAB DBG\$CS_DATE_TIME		
00000000G	00		52 DD 000E9	PUSHL R2		
	01		03 FB 000EB	CALLS #3, DBG\$NMATCH		
			50 D1 000F2	CMPL R0, #1		
			07 12 000F5	BNEQ 8\$		
14	AE		23 D0 000F7	MOVL #35, TYPE	1351	
			02D1 31 000FB	BRW 42\$	1352	
		00000000'	03 DD 000FE 8\$:	PUSHL #3	1355	
			EF 9F 00100	PUSHAB DBG\$CS_PSL		
00000000G	00		52 DD 00106	PUSHL R2		
	01		03 FB 00108	CALLS #3, DBG\$NMATCH		
			50 D1 0010F	CMPL R0, #1		
			0E 12 00112	BNEQ 9\$		
14	AE		04 D0 00114	MOVL #4, TYPE	1357	
10	AE		04 D0 00118	MOVL #4, LENGTH	1358	
01	A5		06 90 0011C	MOVB #6, 1(R5)	1359	
			9A 11 00120	BRB 5\$	1319	
		00000000'	03 DD 00122 9\$:	PUSHL #3	1362	
			EF 9F 00124	PUSHAB DBG\$CS_PSW		
00000000G	00		52 DD 0012A	PUSHL R2		
	01		03 FB 0012C	CALLS #3, DBG\$NMATCH		
			50 D1 00133	CMPL R0, #1		
			0E 12 00136	BNEQ 11\$		
14	AE		03 D0 00138	MOVL #3, TYPE	1364	
10	AE		02 D0 0013C	MOVL #2, LENGTH	1365	
01	A5		07 90 00140	MOVB #7, 1(R5)	1366	
			1E 11 00144	BRB 12\$	1319	
		00000000'	01 DD 00146 10\$:	PUSHL #1	1369	
			EF 9F 00148	PUSHAB DBG\$CS_LONGWORD		
00000000G	00		52 DD 0014E	PUSHL R2		
	01		03 FB 00150	CALLS #3, DBG\$NMATCH		
			50 D1 00157	CMPL R0, #1		
			0B 12 0015A	BNEQ 13\$		
14	AE		08 D0 0015C	MOVL #8, TYPE	1371	
10	AE		04 D0 00160	MOVL #4, LENGTH	1372	
			FEE0 31 00164	BRW 1\$	1319	
			02 DD 00167 12\$:	PUSHL #2	1375	
		00000000'	EF 9F 00169	PUSHAB DBG\$CS_AC		
00000000G	00		52 DD 0016F	PUSHL R2		
	57		03 FB 00171	CALLS #3, DBG\$NMATCH		
	01		50 D0 00178	MOVL R0, R7		
			57 D1 0017B	CMPL R7, #1		
			16 13 0017E	BEQL 14\$		
			05 DD 00180	PUSHL #5		
		00000000'	EF 9F 00182	PUSHAB DBG\$CS_ASCII		
			52 DD 00188	PUSHL R2	1376	

00000000G	00	03	FB	0018A	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00191	CMPL	R0, #1		
		0A	12	00194	BNEQ	16\$		
	14	26	D0	00196	14\$:	MOVL	#38, TYPE	1378
	10	01	D0	0019A	15\$:	MOVL	#1, LENGTH	1379
		37	11	0019E		BRB	18\$	1380
		02	DD	001A0	16\$:	PUSHL	#2	1383
		EF	9F	001A2		PUSHAB	DBG\$CS_AW	
		52	DD	001A8		PUSHL	R2	
00000000G	00	03	FB	001AA	CALLS	#3, DBG\$NMATCH		
	57	50	D0	001B1	MOVL	R0, R7		
	01	57	D1	001B4	CMPL	R7, #1		
		16	13	001B7	BEQL	17\$		
		05	DD	001B9	PUSHL	#5		1384
		EF	9F	001BB	PUSHAB	DBG\$CS_ASCIW		
		52	DD	001C1	PUSHL	R2		
00000000G	00	03	FB	001C3	CALLS	#3, DBG\$NMATCH		
	01	50	D1	001CA	CMPL	R0, #1		
		0A	12	001CD	BNEQ	19\$		
	14	25	D0	001CF	17\$:	MOVL	#37, TYPE	1386
	10	02	D0	001D3		MOVL	#2, LENGTH	1387
		6C	11	001D7	18\$:	BRB	24\$	1388
		02	DD	001D9	19\$:	PUSHL	#2	1391
		EF	9F	001DB		PUSHAB	DBG\$CS_AZ	
		52	DD	001E1	PUSHL	R2		
00000000G	00	03	FB	001E3	CALLS	#3, DBG\$NMATCH		
	57	50	D0	001EA	MOVL	R0, R7		
	01	57	D1	001ED	CMPL	R7, #1		
		16	13	001F0	BEQL	20\$		
		05	DD	001F2	PUSHL	#5		1392
		EF	9F	001F4	PUSHAB	DBG\$CS_ASCIZ		
		52	DD	001FA	PUSHL	R2		
00000000G	00	03	FB	001FC	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00203	CMPL	R0, #1		
		06	12	00206	BNEQ	21\$		
	14	27	D0	00208	20\$:	MOVL	#39, TYPE	1394
		8C	11	0020C		BRB	15\$	1395
		02	DD	0020E	21\$:	PUSHL	#2	1399
		EF	9F	00210		PUSHAB	DBG\$CS_AD	
		52	DD	00216	PUSHL	R2		
00000000G	00	03	FB	00218	CALLS	#3, DBG\$NMATCH		
	57	50	D0	0021F	MOVL	R0, R7		
	01	57	D1	00222	CMPL	R7, #1		
		16	13	00225	BEQL	22\$		
		05	DD	00227	PUSHL	#5		1400
		EF	9F	00229	PUSHAB	DBG\$CS_ASCID		
		52	DD	0022F	PUSHL	R2		
00000000G	00	03	FB	00231	CALLS	#3, DBG\$NMATCH		
	01	50	D1	00238	CMPL	R0, #1		
		0A	12	0023B	BNEQ	25\$		
	14	38	D0	0023D	22\$:	MOVL	#56, TYPE	1402
	10	08	D0	00241	23\$:	MOVL	#8, LENGTH	1403
		79	11	00245	24\$:	BRB	29\$	1404
		01	DD	00247	25\$:	PUSHL	#1	1407
		EF	9F	00249		PUSHAB	DBG\$CS_ASCII	
		52	DD	0024F	PUSHL	R2		
00000000G	00	03	FB	00251	CALLS	#3, DBG\$NMATCH		

	01		50	D1	00258	CMPL	R0, #1		
			66	12	00258	BNEQ	30\$		
			01	DD	0025D	PUSHL	#1		1413
		00000000'	EF	9F	0025F	PUSHAB	DBG\$CS_COLON		
00000000G	00		52	DD	00265	PUSHL	R2		
	05		03	FB	00267	CALLS	#3, DBG\$NMATCH		
			50	EB	0026E	BLBS	R0, 26\$		
		10	AE	D4	00271	CLRL	LENGTH		1416
			46	11	00274	BRB	28\$		1413
		10	AE	9F	00276	PUSHAB	LENGTH		1423
			52	DD	00279	PUSHL	R2		
00000000G	00		02	FB	0027B	CALLS	#2, DBG\$NSAVE_INTEGER		
	03		50	EB	00282	BLBS	R0, 27\$		
			03FB	31	00285	BRW	76\$		
00000800	8F	10	AE	D1	00288	CMPL	LENGTH, #2048		1429
			2A	1B	00290	BLEQU	28\$		
00000000'	EF		01	B0	00292	MOVW	#1, NUM_DESC		1435
00000000'	EF	0C000000'	EF	9E	00299	MOVAB	P.ACW, NUM_DESC+4		1436
		00000000'	EF	9F	002A4	PUSHAB	NUM_DESC		1437
			01	DD	002AA	PUSHL	#1		
		000281B0	8F	DD	002AC	PUSHL	#164272		
00000000G	00		03	FB	002B2	CALLS	#3, DBG\$NMAKE_ARG_VECT		
			03C3	31	002B9	BRW	75\$		
	14	AE	0E	D0	002BC	MOVL	#14, TYPE		1442
			01A6	31	002C0	BRW	50\$		1443
			03	DD	002C3	PUSHL	#3		1446
		00000000'	EF	9F	002C5	PUSHAB	DBG\$CS_DEFAULT		
			52	DD	002CB	PUSHL	R2		
00000000G	00		03	FB	002CD	CALLS	#3, DBG\$NMATCH		
	01		50	D1	002D4	CMPL	R0, #1		
			E7	13	002D7	BEQL	29\$		
		00000000'	02	DD	002D9	PUSHL	#2		1451
			EF	9F	002DB	PUSHAB	DBG\$CS_BINARY		
			52	DD	002E1	PUSHL	R2		
00000000G	00		03	FB	002E3	CALLS	#3, DBG\$NMATCH		
	01		50	D1	002EA	CMPL	R0, #1		
			05	12	002ED	BNEQ	31\$		
		54	02	D0	002EF	MOVL	#2, RADIX		1453
			6C	11	002F2	BRB	35\$		1319
		00000000'	01	DD	002F4	PUSHL	#1		1456
			EF	9F	002F6	PUSHAB	DBG\$CS_OCTAL		
			52	DD	002FC	PUSHL	R2		
00000000G	00		03	FB	002FE	CALLS	#3, DBG\$NMATCH		
	01		50	D1	00305	CMPL	R0, #1		
			05	12	00308	BNEQ	32\$		
		54	08	D0	0030A	MOVL	#8, RADIX		1458
			51	11	0030D	BRB	35\$		1319
		00000000'	02	DD	0030F	PUSHL	#2		1461
			EF	9F	00311	PUSHAB	DBG\$CS_D_FLOAT		
			52	DD	00317	PUSHL	R2		
00000000G	00		03	FB	00319	CALLS	#3, DBG\$NMATCH		
	01		50	D1	00320	CMPL	R0, #1		
			07	12	00323	BNEQ	33\$		
		14	AE	D0	00325	MOVL	#11, TYPE		1463
			FF15	31	00329	BRW	23\$		1464
		00000000'	01	DD	0032C	PUSHL	#1		1468
			EF	9F	0032E	PUSHAB	DBG\$CS_DECIMAL		

00000000G	00	52	DD	00334	PUSHL	R2			
	01	03	FB	00336	CALLS	#3,	DBG\$NMATCH		
		50	D1	0033D	CMPL	R0,	#1		
	54	05	12	00340	BNEQ	34\$			1470
		0A	D0	00342	MOVL	#10,	RADIX		1319
		6C	11	00345	BRB	40\$			1473
		01	DD	00347	PUSHL	#1			
	00000000'	EF	9F	00349	PUSHAB	DBG\$CS_HEXADecimal			
		52	DD	0034F	PUSHL	R2			
00000000G	00	03	FB	00351	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	00358	CMPL	R0,	#1		
	54	05	12	0035B	BNEQ	36\$			1475
		10	D0	0035D	MOVL	#16,	RADIX		1319
		71	11	00360	BRB	43\$			1478
		01	DD	00362	PUSHL	#1			
	00000000'	EF	9F	00364	PUSHAB	DBG\$CS_SYMBOLS			
		52	DD	0036A	PUSHL	R2			
00000000G	00	03	FB	0036C	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	00373	CMPL	R0,	#1		
	59	05	12	00376	BNEQ	37\$			1480
		01	D0	00378	MOVL	#1,	SYMBOL_FLAG		1481
		18	11	0037B	BRB	38\$			1484
		01	DD	0037D	PUSHL	#1			
	00000000'	EF	9F	0037F	PUSHAB	DBG\$CS_NOSYMBOLS			
		52	DD	00385	PUSHL	R2			
00000000G	00	03	FB	00387	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	0038E	CMPL	R0,	#1		
		07	12	00391	BNEQ	39\$			
	56	59	D4	00393	CLRL	SYMBOL_FLAG			1486
		01	D0	00395	MOVL	#1,	SYM_OVERRIDE_PRESENT		1487
		59	11	00398	BRB	45\$			1319
		02	DD	0039A	PUSHL	#2			1490
	00000000'	EF	9F	0039C	PUSHAB	DBG\$CS_SOURCE			
		52	DD	003A2	PUSHL	R2			
00000000G	00	03	FB	003A4	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	003AB	CMPL	R0,	#1		
	5B	05	12	003AE	BNEQ	41\$			1492
		01	D0	003B0	MOVL	#1,	SOURCE_FLAG		1319
		3E	11	003B3	BRB	45\$			1495
		01	DD	003B5	PUSHL	#1			
	00000000'	EF	9F	003B7	PUSHAB	DBG\$CS_QUADWORD			
		52	DD	003BD	PUSHL	R2			
00000000G	00	03	FB	003BF	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	003C6	CMPL	R0,	#1		
		0A	12	003C9	BNEQ	44\$			
	14 AE	09	D0	003CB	MOVL	#9,	TYPE		1497
	10 AE	08	D0	003CF	MOVL	#8,	LENGTH		1498
		1E	11	003D3	BRB	45\$			1319
		05	DD	003D5	PUSHL	#5			1501
	00000000'	EF	9F	003D7	PUSHAB	DBG\$CS_OCTAWORD			
		52	DD	003DD	PUSHL	R2			
00000000G	00	03	FB	003DF	CALLS	#3,	DBG\$NMATCH		
	01	50	D1	003E6	CMPL	R0,	#1		
		0A	12	003E9	BNEQ	46\$			
	14 AE	1A	D0	003EB	MOVL	#26,	TYPE		1503
	10 AE	10	D0	003EF	MOVL	#16,	LENGTH		1504
		77	11	003F3	BRB	51\$			1319

		00000000'	01 DD 003F5	46\$:	PUSHL #1	1507
			EF 9F 003F7		PUSHAB DBG\$CS_FLOAT	
00000000G	00		52 DD 003FD		PUSHL R2	
	57		03 FB 003FF		CALLS #3, DBG\$NMATCH	
	01		50 DO 00406		MOVL R0, R7	
			57 D1 00409		CMPL R7, #1	
			16 13 0040C		BEQL 47\$	
		00000000'	01 DD 0040E		PUSHL #1	1508
			EF 9F 00410		PUSHAB DBG\$CS_F_FLOAT	
00000000G	00		52 DD 00416		PUSHL R2	
	01		03 FB 00418		CALLS #3, DBG\$NMATCH	
			50 D1 0041F		CMPL R0, #1	
			0A 12 00422		BNEQ 48\$	
14	AE		0A DO 00424	47\$:	MOVL #10, TYPE	1510
10	AE		04 DO 00428		MOVL #4, LENGTH	1511
			3B 11 0042C		BRB 50\$	1512
		00000000'	01 DD 0042E	48\$:	PUSHL #1	1515
			EF 9F 00430		PUSHAB DBG\$CS_G_FLOAT	
00000000G	00		52 DD 00436		PUSHL R2	
	01		03 FB 00438		CALLS #3, DBG\$NMATCH	
			50 D1 0043F		CMPL R0, #1	
			07 12 00442		BNEQ 49\$	
14	AE		1B DO 00444		MOVL #27, TYPE	1517
			FD 31 00448		BRW 23\$	1518
		00000000'	02 DD 0044B	49\$:	PUSHL #2	1522
			EF 9F 0044D		PUSHAB DBG\$CS_H_FLOAT	
00000000G	00		52 DD 00453		PUSHL R2	
	01		03 FB 00455		CALLS #3, DBG\$NMATCH	
			50 D1 0045C		CMPL R0, #1	
			0D 12 0045F		BNEQ 52\$	
14	AE		1C DO 00461		MOVL #28, TYPE	1524
10	AE		10 DO 00465		MOVL #16, LENGTH	1525
	54		01 DO 00469	50\$:	MOVL #1, RADIX	1526
			65 11 0046C	51\$:	BRB 56\$	1319
		00000000'	02 DD 0046E	52\$:	PUSHL #2	1529
			EF 9F 00470		PUSHAB DBG\$CS_PACKED	
00000000G	00		52 DD 00476		PUSHL R2	
	01		03 FB 00478		CALLS #3, DBG\$NMATCH	
			50 D1 0047F		CMPL R0, #1	
			52 12 00482		BNEQ 57\$	
14	AE		15 DO 00484		MOVL #21, TYPE	1531
54			01 DO 00488		MOVL #1, RADIX	1532
		00000000'	01 DD 0048B		PUSHL #1	1538
			EF 9F 0048D		PUSHAB DBG\$CS_COLON	
00000000G	00		52 DD 00493		PUSHL R2	
	06		03 FB 00495		CALLS #3, DBG\$NMATCH	
			50 EB 0049C		BLBS R0, 53\$	
10	AE		01 CE 0049F		MNEGL #1, LENGTH	1541
			2E 11 004A3		BRB 56\$	1538
		10	AE 9F 004A5	53\$:	PUSHAB LENGTH	1548
			52 DD 004A8		PUSHL R2	
00000000G	00		02 FB 004AA		CALLS #2, DBG\$NSAVE_INTEGER	
	03		50 EB 004B1		BLBS R0, 54\$	
			01 CC 31 004B4		BRW 76\$	
	50	10	AE DO 004B7	54\$:	MOVL LENGTH, R0	1554
			05 19 004BB		BLSS 55\$	
	1F		50 D1 004BD		CMPL R0, #31	

			11	15	004C0		BLEQ	56\$		
			50	DD	004C2	55\$:	PUSHL	R0		1556
			01	DD	004C4		PUSHL	#1		
00000000G	00	00028828	8F	DD	004C6		PUSHL	#165928		
			03	FB	004CC		CALLS	#3, LIBSSIGNAL		
			FB71	31	004D3	56\$:	BRW	1\$		1319
			01	DD	004D6	57\$:	PUSHL	#1		1563
		00000000'	EF	9F	004D8		PUSHAB	DBG\$CS_CR		
			52	DD	004DE		PUSHL	R2		
00000000G	00		03	FB	004E0		CALLS	#3, DBG\$NMATCH		
	03		50	EB	004E7		BLBS	R0, 58\$		
			0180	31	004EA		BRW	74\$		
		000280DC	8F	DD	004ED	58\$:	PUSHL	#164048		1565
00000000G	00		01	FB	004F3		CALLS	#1, DBG\$NMAKE_ARG_VECT		
			0182	31	004FA		BRW	75\$		1563
	14		56	E9	004FD	59\$:	BLBC	SYM_OVERRIDE_PRESENT, 60\$		1576
			02	DD	00500		PUSHL	#2		1579
00000000G	00		01	FB	00502		CALLS	#1, DBG\$SET_MOD_LVL		
	50	00000000G	00	DD	00509		MOVL	DBG\$GB_MOD_PTR, R0		1580
	02	A0	59	90	00510		MOVB	SYMBOL_FLAG, 2(R0)		
			01	DD	00514	60\$:	PUSHL	#1		1586
		00000000'	EF	9F	00516		PUSHAB	DBG\$CS_CR		
			52	DD	0051C		PUSHL	R2		
00000000G	00		03	FB	0051E		CALLS	#3, DBG\$NMATCH		
	03		50	EB	00525		BLBS	R0, 61\$		
			0086	31	00528		BRW	68\$		
			08	AE	9F	0052B	61\$:	PUSHAB	DUMMY	1605
			0C	AE	9F	0052E		PUSHAB	DUMMY	
			14	AE	9F	00531		PUSHAB	SYMBOL_VALUE	
			10	AE	9F	00534		PUSHAB	KIND	
		00000000'	EF	9F	00537		PUSHAB	P.ACX		
00000000G	00		05	FB	0053D		CALLS	#5, DBG\$DEF_SYM_FIND		
	51		50	E9	00544		BLBC	R0, 65\$		
	01	04	AE	D1	00547		CMPL	KIND, #1		1609
			06	13	0054B		BEQL	62\$		
	05	04	AE	D1	0054D		CMPL	KIND, #5		1610
			45	12	00551		BNEQ	65\$		
	50	0C	AE	D0	00553	62\$:	MOVL	SYMBOL_VALUE, R0		1616
7A	8F	02	A0	91	00557		CMPB	2(R0), #122		
			27	12	0055C		BNEQ	64\$		
	02	A0	83	8F	90	0055E		MOVB	#-125, 2(R0)	1619
18	A0	20	A0	D0	00563		MOVL	32(R0), 24(R0)		1621
	01	16	A0	91	00568		CMPB	22(R0), #1		1622
			12	13	0056C		BEQL	63\$		
	22	16	A0	91	0056E		CMPB	22(R0), #34		1623
			0C	13	00572		BEQL	63\$		
	29	16	A0	91	00574		CMPB	22(R0), #41		1624
			06	13	00578		BEQL	63\$		
	2A	16	A0	91	0057A		CMPB	22(R0), #42		1625
			05	12	0057E		BNEQ	64\$		
	1C	A0	24	A0	D0	00580	63\$:	MOVL	36(R0), 28(R0)	1628
			7E	D4	00585	64\$:	CLRL	-(SP)		1635
			0C	AE	9F	00587		PUSHAB	DUMMY	
			14	AE	9F	0058A		PUSHAB	SYMBOL_VALUE	
			50	DD	0058D		PUSHL	R0		
00000000G	00		04	FB	0058F		CALLS	#4, DBG\$Ncopy_DESC		
			0D	11	00596		BRB	66\$		1609

00000000G	00	00028808	8F	DD	00598	65\$:	PUSHL	#165896	1642
	63		01	FB	0059E		CALLS	#1, LIBSSIGNAL	1644
	OC	OC	AE	DO	005A5	66\$:	MOVL	SYMBOL_VALUE, (NOUN_NODE)	1645
	A3	OC	AE	DO	005A9		MOVL	SYMBOL_VALUE, 12(NOUN_NODE)	1586
			00D6	31	005AE	67\$:	BRW	77\$	1657
			01	DD	005B1	68\$:	PUSHL	#1	
		00000000'	EF	9F	005B3		PUSHAB	DBG\$CS_CR	
00000000G	00		52	DD	005B9		PUSHL	R2	
	E9		03	FB	005BB		CALLS	#3, DBG\$NMATCH	
			50	E8	005C2		BLBS	R0, 67\$	
			62	B5	005C5		TSTW	(R2)	1658
			E5	13	005C7		BEQL	67\$	
	1A		58	E9	005C9		BLBC	LOWER_PC_FLAG, 69\$	1664
		OC	AC	DD	005CC		PUSHL	MESSAGE_VECT	1669
			08	DD	005CF		PUSHL	#8	1668
			OC	AE	DD	005D1	PUSHL	INP_RADIX	
			05	FB	005D4		PUSHR	#*MZR2,R3>	
00000000G	00		50	DD	005D6		CALLS	#5, DBG\$NPARSE_ADDRESS	
	57		50	DO	005DD		MOVL	R0, STATUS	
	OC		63	DO	005E0		MOVL	(NOUN_NODE), 12(NOUN_NODE)	1670
			17	11	005E4		BRB	70\$	1664
			OC	AC	DD	005E6	69\$:	PUSHL	MESSAGE_VECT
			01	DD	005E9		PUSHL	#1	1675
			08	AE	DD	005EB		PUSHL	INP_RADIX
		OC	A3	9F	005EE		PUSHAB	12(NOUN_NODE)	1674
			52	DD	005F1		PUSHL	R2	
00000000G	00		05	FB	005F3		CALLS	#5, DBG\$NPARSE_ADDRESS	
	57		50	DO	005FA		MOVL	R0, STATUS	
	01		57	D1	005FD	70\$:	CMPL	STATUS, #1	1677
			AF	13	00600		BEQL	68\$	
			57	D5	00602		TSTL	STATUS	1679
			7D	12	00604		BNEQ	76\$	
			01	DD	00606		PUSHL	#1	1685
		00000000'	EF	9F	00608		PUSHAB	DBG\$CS_COMMA	
			52	DD	0060E		PUSHL	R2	
00000000G	00		03	FB	00610		CALLS	#3, DBG\$NMATCH	
	37		50	E9	00617		BLBC	R0, 72\$	
			01	DD	0061A		PUSHL	#1	1692
		00000000'	EF	9F	0061C		PUSHAB	DBG\$CS_CR	
			52	DD	00622		PUSHL	R2	
00000000G	00		03	FB	00624		CALLS	#3, DBG\$NMATCH	
	0D		50	E9	0062B		BLBC	R0, 71\$	
		000280D0	8F	DD	0062E		PUSHL	#164048	1694
00000000G	00		01	FB	00634		CALLS	#1, LIBSSIGNAL	
			04	DD	0063B	71\$:	PUSHL	#4	1700
00000000G	00		01	FB	0063D		CALLS	#1, DBG\$GET_TEMPMEM	
	08		50	DO	00644		MOVL	R0, 8(NOUN_NODE)	
			53	DO	00648		MOVL	8(NOUN_NODE), NOUN_NODE	1701
		08	01	DO	0064C		MOVL	#1, LOWER_PC_FLAG	1706
			19	11	0064F		BRB	73\$	1685
			58	E9	00651	72\$:	BLBC	LOWER_PC_FLAG, 73\$	1710
	16		01	DD	00654		PUSHL	#1	1712
		00000000'	EF	9F	00656		PUSHAB	DBG\$CS_COLON	
			52	DD	0065C		PUSHL	R2	
00000000G	00		03	FB	0065E		CALLS	#3, DBG\$NMATCH	
	05		50	E9	00665		BLBC	RC, 74\$	
			58	D4	00668		CLRL	LOWER_PC_FLAG	1714

		FF44	31	0066A	73\$:	BRW	68\$		
		52	DD	0066D	74\$:	PUSHL	R2		1718
00000000G	00	01	FB	0066F		CALLS	#1, DBGS\$NEXT_WORD		
		50	DD	00676		PUSHL	R0		
00000000G	00	01	FB	00678		CALLS	#1, DBGS\$SYNTAX_ERROR		
	0C	50	DD	0067F	75\$:	MOVL	R0, @MESSAGE_VECT		
		04	DD	00683	76\$:	MOVL	#4, R0		1725
		04	DD	00686		RET			
		03	DD	00687	77\$:	PUSHL	#3		1742
00000000G	00	01	FB	00689		CALLS	#1, DBGS\$GET_TEMPMEM		
	57	50	DD	00690		MOVL	R0, ADVERB_NODE		
	04	57	DD	00693		MOVL	ADVERB_NODE, 4(R5)		1744
	67	80	8F	90	00697	MOVB	#-128, (ADVERB_NODE)		1745
	50	63	DD	0069B		MOVL	(NOUN_NODE), ADDR_EXP_DESC		1750
FFFFFFFFF	8F	14	AE	D1	0069E	CMPL	TYPE, #-1		1755
		12	13	006A6		BEQL	78\$		
	67	14	AE	90	006A8	MOVB	TYPE, (ADVERB_NODE)		1758
	04	10	AE	D0	006AC	MOVL	LENGTH, 4(ADVERB_NODE)		1759
	3B	5A	E9	006B1		BLBC	MSG_TEXT, 80\$		1763
	01	05	90	006B4		MOVB	#5, -1(R5)		1765
		35	11	006B8		BRB	80\$		1755
		10	AE	9F	006BA	78\$:	PUSHAB	LENGTH	1773
		18	AE	9F	006BD	PUSHAB	TYPE		
00000000G	00	02	FB	006C0		CALLS	#2, DBGS\$NGET_OVERRIDE_TYPE		
	1C	50	EB	006C7		BLBS	R0, 79\$		
		10	AE	9F	006CA	PUSHAB	LENGTH		1789
		18	AE	9F	006CD	PUSHAB	TYPE		
00000000G	00	02	FB	006D0		CALLS	#2, DBGS\$NGET_POTENTIAL_TYPE		
00000080	8F	14	AE	D1	006D7	CMPL	TYPE, #128		1791
		0E	13	006DF		BEQL	80\$		
		14	AE	D5	006E1	TSTL	TYPE		
	67	09	13	006E4		BEQL	80\$		
	04	14	AE	90	006E6	79\$:	MOVB	TYPE, (ADVERB_NODE)	1794
	A7	10	AE	D0	006EA	MOVL	LENGTH, 4(ADVERB_NODE)		1795
	52	08	A7	9E	006EF	80\$:	MOVAB	8(R7), LINK	1806
		03	DD	006F3		PUSHL	#3		1807
00000000G	00	01	FB	006F5		CALLS	#1, DBGS\$GET_TEMPMEM		
	57	50	DD	006FC		MOVL	R0, ADVERB_NODE		
	62	57	DD	006FF		MOVL	ADVERB_NODE, (LINK)		1809
	67	54	90	00702		MOVB	RADIX, (ADVERB_NODE)		1810
00000000G	00	54	DD	00705		MOVL	RADIX, DBGS\$GL_CMND_RADIX		1811
	52	08	A7	9E	0070C	MOVAB	8(R7), LINK		1815
		03	DD	00710		PUSHL	#3		1816
00000000G	00	01	FB	00712		CALLS	#1, DBGS\$GET_TEMPMEM		
	57	50	DD	00719		MOVL	R0, ADVERB_NODE		
	62	57	DD	0071C		MOVL	ADVERB_NODE, (LINK)		1818
	67	59	90	0071F		MOVB	SYMBOL_FLAG, (ADVERB_NODE)		1819
	04	5B	E9	00722		BLBC	SOURCE_FLAG, 81\$		1821
	01	04	90	00725		MOVB	#4, 1(R5)		1823
	50	01	DD	00729	81\$:	MOVL	#1, R0		1825
		04	0072C			RET			1826

; Routine Size: 1837 bytes, Routine Base: DBG\$CODE + 05D7

```

: 1700 1827 1 GLOBAL ROUTINE DBG$NFORMAT_WITH_RADIX (ADDRESS, OFFSET, LENGTH, RADIX) : NOVALUE =
: 1701 1828 1
: 1702 1829 1
: 1703 1830 1 ++
: 1704 1831 1 FUNCTIONAL DESCRIPTION:
: 1705 1832 1     Formats the contents of a arbitrary number of bits into a printable
: 1706 1833 1     hex or octal number, or a sequence of unsigned decimal longwords.
: 1707 1834 1
: 1708 1835 1 FORMAL PARAMETERS:
: 1709 1836 1
: 1710 1837 1     ADDRESS      -      A longword containing the byte address of the
: 1711 1838 1     location to be printed
: 1712 1839 1
: 1713 1840 1     OFFSET      -      A longword containing the starting bit offset
: 1714 1841 1
: 1715 1842 1     LENGTH      -      A longword containing the number of bits to be
: 1716 1843 1     printed
: 1717 1844 1
: 1718 1845 1     RADIX       -      A longword containing dbg$k_octal, dbg$k_decimal, or
: 1719 1846 1     dbg$k_hex, indicating the radix to be used for output
: 1720 1847 1
: 1721 1848 1 IMPLICIT INPUTS:
: 1722 1849 1
: 1723 1850 1     NONE
: 1724 1851 1
: 1725 1852 1 IMPLICIT OUTPUTS:
: 1726 1853 1
: 1727 1854 1     The byte stream corresponding to the translated number is filled
: 1728 1855 1
: 1729 1856 1 ROUTINE VALUE:
: 1730 1857 1
: 1731 1858 1     NOVALUE
: 1732 1859 1
: 1733 1860 1 COMPLETION CODES:
: 1734 1861 1
: 1735 1862 1     NONE
: 1736 1863 1
: 1737 1864 1 SIDE EFFECTS:
: 1738 1865 1
: 1739 1866 1     NONE
: 1740 1867 1
: 1741 1868 1 --
: 1742 1869 2 BEGIN
: 1743 1870 2
: 1744 1871 2 LOCAL
: 1745 1872 2     CHAR_TABLE : VECTOR [16],      ! Character trans table
: 1746 1873 2     NIBBLE     : BLOCK [1],      ! Nexts bits for char trans
: 1747 1874 2     BIT_VECT  : REF BITVECTOR, ! Bits to be translated
: 1748 1875 2     BITS_PER_NIBBLE, ! Number of bits for 1 char
: 1749 1876 2     FIRST_NIBBLE_BITS, ! Number of bits for first nibble
: 1750 1877 2     FIRST_FLAG,   ! Indicates first nibble
: 1751 1878 2     START,        ! Start bitvect position
: 1752 1879 2     FINISH,       ! End bitvect position
: 1753 1880 2     LONG_WORD   : REF BLOCK,    ! Longword for decimal output
: 1754 1881 2     INDEX;
: 1755 1882 2
: 1756 1883 2 MACRO

```

```

: 1757 M 1884 2
: 1758 M 1885 2
: 1759 M 1886 2
: 1760 M 1887 2
: 1761 M 1888 2
: 1762 M 1889 2
: 1763 M 1890 2
: 1764 M 1891 2
: 1765 M 1892 2
: 1766 M 1893 2
: 1767 M 1894 2
: 1768 M 1895 2
: 1769 M 1896 2
: 1770 M 1897 2
: 1771 M 1898 2
: 1772 M 1899 2
: 1773 M 1900 2
: 1774 M 1901 2
: 1775 M 1902 2
: 1776 M 1903 2
: 1777 M 1904 2
: 1778 M 1905 2
: 1779 M 1906 2
: 1780 M 1907 2
: 1781 M 1908 2
: 1782 M 1909 2
: 1783 M 1910 2
: 1784 M 1911 2
: 1785 M 1912 2
: 1786 M 1913 2
: 1787 M 1914 2
: 1788 M 1915 2
: 1789 M 1916 2
: 1790 M 1917 2
: 1791 M 1918 2
: 1792 M 1919 2
: 1793 M 1920 2
: 1794 M 1921 2
: 1795 M 1922 2
: 1796 M 1923 2
: 1797 M 1924 2
: 1798 M 1925 2
: 1799 M 1926 2
: 1800 M 1927 4
: 1801 M 1928 4
: 1802 M 1929 4
: 1803 M 1930 4
: 1804 M 1931 4
: 1805 M 1932 4
: 1806 M 1933 4
: 1807 M 1934 3
: 1808 M 1935 4
: 1809 M 1936 4
: 1810 M 1937 4
: 1811 M 1938 4
: 1812 M 1939 4
: 1813 M 1940 4

```

```

GET_NIBBLE =
BEGIN
! This macro obtains the next nibble of bits for octal or hex output
!
nibble = 0;

! Obtain the correct number of bits
!
IF .bits_per_nibble EQL 3
THEN
BEGIN
nibble [0, 2, 1, 0] = .bit_vect [.start];
nibble [0, 1, 1, 0] = .bit_vect [.start - 1];
nibble [0, 0, 1, 0] = .bit_vect [.start - 2];
END
ELSE
BEGIN
nibble [0, 3, 1, 0] = .bit_vect [.start];
nibble [0, 2, 1, 0] = .bit_vect [.start - 1];
nibble [0, 1, 1, 0] = .bit_vect [.start - 2];
nibble [0, 0, 1, 0] = .bit_vect [.start - 3];
END;

start = .start - .bits_per_nibble;
END %;

! Check for decimal or other radix
!
IF .radix EQL dbg$k_decimal
THEN
BEGIN
! Line up the longword and take bits corresponding to the
! the first longword and output them
!
long_word = .address;
IF .length + .offset LEQU %BPVAL
THEN
BEGIN
! The entity has less than a longword of bits, counting the offset
!
nibble = .long_word [0, .offset, .length, 0];
length = 0;
END
ELSE
BEGIN
! At least a longword of bits counting the offset
!
nibble = .long_word [0, .offset, %BPVAL - .offset, 0];
length = .length - (%BPVAL - .offset);

```

```

: 1814 1941
: 1815 1942
: 1816 1943
: 1817 1944
: 1818 1945
: 1819 1946
: 1820 1947
: 1821 1948
: 1822 1949
: 1823 1950
: 1824 1951
: 1825 1952
: 1826 1953
: 1827 1954
: 1828 1955
: 1829 1956
: 1830 1957
: 1831 1958
: 1832 1959
: 1833 1960
: 1834 1961
: 1835 1962
: 1836 1963
: 1837 1964
: 1838 1965
: 1839 1966
: 1840 1967
: 1841 1968
: 1842 1969
: 1843 1970
: 1844 1971
: 1845 1972
: 1846 1973
: 1847 1974
: 1848 1975
: 1849 1976
: 1850 1977
: 1851 1978
: 1852 1979
: 1853 1980
: 1854 1981
: 1855 1982
: 1856 1983
: 1857 1984
: 1858 1985
: 1859 1986
: 1860 1987
: 1861 1988
: 1862 1989
: 1863 1990
: 1864 1991
: 1865 1992
: 1866 1993
: 1867 1994
: 1868 1995
: 1869 1996
: 1870 1997

        END;

        ! Output the first longword
        dbg$print (UPLIT BYTE (%ASCIC '!ZL'), .nibble);

        ! Loop, outputting the bits a longword at a time
        index = 1;
        WHILE .length GEQU %BPVAL
        DO
            BEGIN
                ! Output a space
                dbg$print (UPLIT BYTE (%ASCIC '!AC'), UPLIT BYTE (%ASCIC ' '));

                ! Output the next longword
                dbg$print (UPLIT BYTE (%ASCIC '!ZL'),
                    .long_word [.index, 0, 32, 0]);

                ! Increase index and decrease length
                index = .index + 1;
                length = .length - %BPVAL;
            END;

        ! Output remaining bits, if any
        IF .length GTRU 0
        THEN
            BEGIN
                dbg$print (UPLIT BYTE (%ASCIC '!AC'), UPLIT BYTE (%ASCIC ' '));
                nibble = .long_word [.index, 0, .length, 0];
                dbg$print (UPLIT BYTE (%ASCIC '!ZL'), .nibble);
            END;
        END

    ELSE
        BEGIN
            ! Octal or hex radix. Set up the bit vector and the start and finish
            ! indicies
            first_flag = true;
            bit_vect = .address;
            finish = .offset;
            start = .finish + .length - 1;
        END
    END

```

```

: 1871
: 1872
: 1873
: 1874
: 1875
: 1876
: 1877
: 1878
: 1879
: 1880
: 1881
: 1882
: 1883
: 1884
: 1885
: 1886
: 1887
: 1888
: 1889
: 1890
: 1891
: 1892
: 1893
: 1894
: 1895
: 1896
: 1897
: 1898
: 1899
: 1900
: 1901
: 1902
: 1903
: 1904
: 1905
: 1906
: 1907
: 1908
: 1909
: 1910
: 1911
: 1912
: 1913
: 1914
: 1915
: 1916
: 1917
: 1918
: 1919
: 1920
: 1921
: 1922
: 1923
: 1924
: 1925
: 1926
: 1927

```

```

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054

```

```

! Set up bits_per_nibble based on radix
bits_per_nibble = (IF .radix EQL dbg$ok_octal THEN 3 ELSE 4);

```

```

! Load up the character trans table
char_table [0] = UPLIT BYTE (%ASCIC '0');
char_table [1] = UPLIT BYTE (%ASCIC '1');
char_table [2] = UPLIT BYTE (%ASCIC '2');
char_table [3] = UPLIT BYTE (%ASCIC '3');
char_table [4] = UPLIT BYTE (%ASCIC '4');
char_table [5] = UPLIT BYTE (%ASCIC '5');
char_table [6] = UPLIT BYTE (%ASCIC '6');
char_table [7] = UPLIT BYTE (%ASCIC '7');
char_table [8] = UPLIT BYTE (%ASCIC '8');
char_table [9] = UPLIT BYTE (%ASCIC '9');
char_table [10] = UPLIT BYTE (%ASCIC 'A');
char_table [11] = UPLIT BYTE (%ASCIC 'B');
char_table [12] = UPLIT BYTE (%ASCIC 'C');
char_table [13] = UPLIT BYTE (%ASCIC 'D');
char_table [14] = UPLIT BYTE (%ASCIC 'E');
char_table [15] = UPLIT BYTE (%ASCIC 'F');

```

```

! Determine how many bits go in the first nibble and format the first nibble
first_nibble_bits = .length MOD .bits_per_nibble;
nibble = 0;

```

```

CASE .first_nibble_bits FROM 0 TO 3
  OF
  SET
  [0] :
    BEGIN
      0; ! Do nothing since no remainder
    END;
  [1] :
    BEGIN
      nibble [0, 0, 1, 0] = .bit_vect [.start];
    END;
  [2] :
    BEGIN
      nibble [0, 1, 1, 0] = .bit_vect [.start];
      nibble [0, 0, 1, 0] = .bit_vect [.start - 1];
    END;
  [3] :
    BEGIN
      nibble [0, 2, 1, 0] = .bit_vect [.start];
      nibble [0, 1, 1, 0] = .bit_vect [.start - 1];
      nibble [0, 0, 1, 0] = .bit_vect [.start - 2];
    END;

```

```

: 1928
: 1929
: 1930
: 1931
: 1932
: 1933
: 1934
: 1935
: 1936
: 1937
: 1938
: 1939
: 1940
: 1941
: 1942
: 1943
: 1944
: 1945
: 1946
: 1947
: 1948
: 1949
: 1950
: 1951
: 1952
: 1953
: 1954
: 1955
: 1956
: 1957
: 1958
: 1959
: 1960
: 1961

```

```

2055      TES;
2056
2057      start = .start - .first_nibble_bits;
2058
2059      IF .first_nibble_bits NEQ 0
2060      THEN
2061      BEGIN
2062      first_flag = false;
2063      dbg$print (UPLIT BYTE (%ASCIC '!AC'), .char_table [.nibble]);
2064      END;
2065
2066      ! Format the rest of the bits
2067      !
2068      !
2069      WHILE .start GTR .finish
2070      DO
2071      BEGIN
2072      get_nibble;
2073
2074
2075      ! IF the first character emitted is alphabetic, prepend a '0'
2076      !
2077      !
2078      IF .first_flag AND .nibble GTR 9
2079      THEN
2080      dbg$print (UPLIT BYTE (%ASCIC '!AC'), UPLIT BYTE (%ASCIC '0'));
2081
2082      dbg$print (UPLIT BYTE (%ASCIC '!AC'), .char_table [.nibble]);
2083      first_flag = false;
2084      END;
2085
2086      END;
2087
2088      RETURN;
2089      END;
2090
2091      ! End of format_with_radix

```

.PSECT DBG\$PLIT, NOWRT, SHR, PIC, 0

4C	5A	21	03	001E9	P.ACY:	.ASCII	<3>!ZL\
43	41	21	03	001ED	P.ACZ:	.ASCII	<3>!AC\
		20	01	001F1	P.ADA:	.ASCII	<1>\ \
4C	5A	21	03	001F3	P.ADB:	.ASCII	<3>!ZL\
43	41	21	03	001F7	P.ADC:	.ASCII	<3>!AC\
		20	01	001FB	P.ADD:	.ASCII	<1>\ \
4C	5A	21	03	001FD	P.ADE:	.ASCII	<3>!ZL\
		30	01	00201	P.ADF:	.ASCII	<1>!0\
		31	01	00203	P.ADG:	.ASCII	<1>!1\
		32	01	00205	P.ADH:	.ASCII	<1>!2\
		33	01	00207	P.ADI:	.ASCII	<1>!3\
		34	01	00209	P.ADJ:	.ASCII	<1>!4\
		35	01	0020B	P.ADK:	.ASCII	<1>!5\
		36	01	0020D	P.ADL:	.ASCII	<1>!6\
		37	01	0020F	P.ADM:	.ASCII	<1>!7\
		38	01	00211	P.ADN:	.ASCII	<1>!8\
		39	01	00213	P.ADO:	.ASCII	<1>!9\
		41	01	00215	P.ADP:	.ASCII	<1>!A\

.....

		6A	02	FB	0008B	CALLS	#2, DBGSPRINT		
		58		04	0008E	RET			1917
		56	04	AC	0008F	6\$:	MOVL	#1, FIRST_FLAG	1992
		53		50	00092	MOVL	ADDRESS, BIT_VECT		1993
	51	53	0C	AC	00096	MOVL	R0, FINISH		1994
		52	FF	A1	00099	ADDL3	LENGTH, FINISH, R1		1995
		08	10	AC	0009E	MOVAB	-1(R1), START		
				05	000A2	CPL	RADIX, #8		2000
		57		03	000A6	BNEQ	7\$		
				03	000AB	MOVL	#3, BITS_PER_NIBBLE		
		57		04	000AD	BRB	8\$		
		6E	18	A9	000B0	7\$:	MOVL	#4, BITS PER NIBBLE	
		04	1A	A9	000B4	8\$:	MOVAB	P.ADF, CHAR_TABLE	2005
		08	1C	A9	000B9	MOVAB	P.ADG, CHAR_TABLE+4		2006
		0C	1E	A9	000BE	MOVAB	P.ADH, CHAR_TABLE+8		2007
		10	20	A9	000C3	MOVAB	P.ADI, CHAR_TABLE+12		2008
		14	22	A9	000C8	MOVAB	P.ADJ, CHAR_TABLE+16		2009
		18	24	A9	000CD	MOVAB	P.ADK, CHAR_TABLE+20		2010
		1C	26	A9	000D2	MOVAB	P.ADL, CHAR_TABLE+24		2011
		20	28	A9	000D7	MOVAB	P.ADM, CHAR_TABLE+28		2012
		24	2A	A9	000DC	MOVAB	P.ADN, CHAR_TABLE+32		2013
		28	2C	A9	000E1	MOVAB	P.ADO, CHAR_TABLE+36		2014
		2C	2E	A9	000E6	MOVAB	P.ADP, CHAR_TABLE+40		2015
		30	30	A9	000E6	MOVAB	P.ADQ, CHAR_TABLE+44		2016
		34	32	A9	000EB	MOVAB	P.ADR, CHAR_TABLE+48		2017
		38	34	A9	000F0	MOVAB	P.ADS, CHAR_TABLE+52		2018
		3C	36	A9	000F5	MOVAB	P.ADT, CHAR_TABLE+56		2019
7E	00			01	000FA	MOVAB	P.ADU, CHAR_TABLE+60		2020
50	50	8E		57	000FF	EMUL	#1, LENGTH, #0, -(SP)		2025
					00105	EDIV	BITS PER NIBBLE, (SP)+, FIRST_NIBBLE_BITS, -		
				54	0010A	CLRL	FIRST_NIBBLE_BITS		2026
				50	0010C	CASEL	NIBBLE		2028
0026	0016	000A	004C	00110	9\$:	.WORD	FIRST_NIBBLE_BITS, #0, #3		
							14\$-9\$,-		
							10\$-9\$,-		
							11\$-9\$,-		
							12\$-9\$		
				42	00118	BRB	14\$		
51	66	01		52	0011A	10\$:	EXTZV	START, #1, (BIT VECT), R1	2039
54	01	00		51	0011F	INSV	R1, #0, #1, NIBBLE		
				36	00124	BRB	14\$		2028
51	66	01		52	00126	11\$:	EXTZV	START, #1, (BIT VECT), R1	2044
54	01	01		51	0012B	INSV	R1, #1, #1, NIBBLE		
		51	FF	A2	00130	MOVAB	-1(R2), R1		2045
				1C	00134	BRB	13\$		
51	66	01		52	00136	12\$:	EXTZV	START, #1, (BIT VECT), R1	2050
54	01	02		51	0013B	INSV	R1, #2, #1, NIBBLE		
		51	FF	A2	00140	MOVAB	-1(R2), R1		2051
55	66	01		51	00144	EXTZV	R1, #1, (BIT VECT), R5		
54	01	01		55	00149	INSV	R5, #1, #1, NIBBLE		
		51	FE	A2	0014E	MOVAB	-2(R2), R1		2052
55	66	01		51	00152	13\$:	EXTZV	R1, #1, (BIT VECT), R5	
54	01	00		55	00157	INSV	R5, #0, #1, NIBBLE		
		52		50	0015C	14\$:	SUBL2	FIRST_NIBBLE_BITS, START	2057
				50	0015F	TSTL	FIRST_NIBBLE_BITS		2059
				0B	00161	BEQL	15\$		
				58	00163	CLRL	FIRST_FLAG		2062

			38	6E44 DD 00165	PUSHL	CHAR TABLE[NIBBLE]		2063
				A9 9F 00168	PUSHAB	P.ADV		
	6A			02 FB 0016B	CALLS	#2, DBG\$PRINT		
	53			52 D1 0016E	15\$:	CMPL	START, FINISH	2069
				72 15 00171	BLEQ	19\$		
				54 D4 00173	CLRL	NIBBLE		2071
	55		FF	A2 9E 00175	MOVAB	-1(R2), R5		
	51		FE	A2 9E 00179	MOVAB	-2(R2), R1		
	03			57 D1 0017D	CMPL	BITS_PER_NIBBLE, #3		
				16 12 00180	BNEQ	16\$		
50	66	01		52 EF 00182	EXTZV	START, #1, (BIT VECT), R0		
54	01	02		50 FO 00187	INSV	R0, #2, #1, NIBBLE		
50	66	01		55 EF 0018C	EXTZV	R5, #1, (BIT VECT), R0		
54	01	01		50 FO 00191	INSV	R0, #1, #1, NIBBLE		
				22 11 00196	BRB	17\$		
50	66	01		52 EF 00198	16\$:	EXTZV	START, #1, (BIT VECT), R0	
54	01	03		50 FO 0019D	INSV	R0, #3, #1, NIBBLE		
50	66	01		55 EF 001A2	EXTZV	R5, #1, (BIT VECT), R0		
54	01	02		50 FO 001A7	INSV	R0, #2, #1, NIBBLE		
50	66	01		51 EF 001AC	EXTZV	R1, #1, (BIT VECT), R0		
54	01	01		50 FO 001B1	INSV	R0, #1, #1, NIBBLE		
			FD	A2 9E 001B6	MOVAB	-3(R2), R1		
50	66	01		51 EF 001BA	17\$:	EXTZV	R1, #1, (BIT VECT), R0	
54	01	00		50 FO 001BF	INSV	R0, #0, #1, NIBBLE		
		52		57 C2 001C4	SUBL2	BITS_PER_NIBBLE, START		
		0E		58 E9 001C7	BLBC	FIRST_FLAG, 18\$		2077
		09		54 D1 001CA	CMPL	NIBBLE, #9		
				09 15 001CD	BLEQ	18\$		
			40	A9 9F 001CF	PUSHAB	P.ADX		2079
			3C	A9 9F 001D2	PUSHAB	P.ADV		
	6A			02 FB 001D5	CALLS	#2, DBG\$PRINT		
				6E44 DD 001DB	18\$:	PUSHL	CHAR TABLE[NIBBLE]	2081
			42	A9 9F 001DB	PUSHAB	P.ADV		
	6A			02 FB 001DE	CALLS	#2, DBG\$PRINT		
				58 D4 001E1	CLRL	FIRST_FLAG		2082
				89 11 001E3	BRB	15\$		2069
				04 001E5	19\$:	RET		2088

: Routine Size: 486 bytes, Routine Base: DBG\$CODE + 0D04

: 1962 2089 1 END !End of module
: 1963 2090 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$PLIT	559	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$OWN	24	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$CODE	3818	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
-\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	27	0	1000	00:01.8
-\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	0	0	7	00:00.1
-\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	95	6	97	00:01.8
-\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	4	0	31	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	6	1	22	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGGEN.L32;1	150	5	3	12	00:00.3

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:DBGEXADep/OBJ=OBJ\$:DBGEXADep MSRC\$:DBGEXADep/UPDATE=(ENH\$:DBGEXADep)

: Size: 3818 code + 583 data bytes
 : Run Time: 01:00.5
 : Elapsed Time: 03:12.6
 : Lines/CPU Min: 2071
 : Lexemes/CPU-Min: 11613
 : Memory Used: 467 pages
 : Compilation Complete

Terminal 1	Terminal 2	Terminal 3	Terminal 4	Terminal 5	Terminal 6	Terminal 7	Terminal 8	Terminal 9	Terminal 10	Terminal 11	Terminal 12
Terminal 13	Terminal 14	Terminal 15	Terminal 16	Terminal 17	Terminal 18	Terminal 19	Terminal 20	Terminal 21	Terminal 22	Terminal 23	Terminal 24
Terminal 25	Terminal 26	Terminal 27	Terminal 28	Terminal 29	Terminal 30	Terminal 31	Terminal 32	Terminal 33	Terminal 34	Terminal 35	Terminal 36
Terminal 37	Terminal 38	Terminal 39	Terminal 40	Terminal 41	Terminal 42	Terminal 43	Terminal 44	Terminal 45	Terminal 46	Terminal 47	Terminal 48
Terminal 49	Terminal 50	Terminal 51	Terminal 52	Terminal 53	Terminal 54	Terminal 55	Terminal 56	Terminal 57	Terminal 58	Terminal 59	Terminal 60
Terminal 61	Terminal 62	Terminal 63	Terminal 64	Terminal 65	Terminal 66	Terminal 67	Terminal 68	Terminal 69	Terminal 70	Terminal 71	Terminal 72
Terminal 73	Terminal 74	Terminal 75	Terminal 76	Terminal 77	Terminal 78	Terminal 79	Terminal 80	Terminal 81	Terminal 82	Terminal 83	Terminal 84
Terminal 85	Terminal 86	Terminal 87	Terminal 88	Terminal 89	Terminal 90	Terminal 91	Terminal 92	Terminal 93	Terminal 94	Terminal 95	Terminal 96
Terminal 97	Terminal 98	Terminal 99	Terminal 100	Terminal 101	Terminal 102	Terminal 103	Terminal 104	Terminal 105	Terminal 106	Terminal 107	Terminal 108
Terminal 109	Terminal 110	Terminal 111	Terminal 112	Terminal 113	Terminal 114	Terminal 115	Terminal 116	Terminal 117	Terminal 118	Terminal 119	Terminal 120
Terminal 121	Terminal 122	Terminal 123	Terminal 124	Terminal 125	Terminal 126	Terminal 127	Terminal 128	Terminal 129	Terminal 130	Terminal 131	Terminal 132