

[illegible]

```
DDDDDDDD  BBBB BBBB  GGGGGGGG  SSSSSSSS  YY  YY  MM  MM  BBBB BBBB  LL  ZZZZZZZZZZ
DDDDDDDD  BBBB BBBB  GGGGGGGG  SSSSSSSS  YY  YY  MM  MM  BBBB BBBB  LL  ZZZZZZZZZZ
DD  DD  BB  BB  GG  SS  YY  YY  MMMM  MMMM  BB  BB  LL  ZZ
DD  DD  BB  BB  GG  SS  YY  YY  MMMM  MMMM  BB  BB  LL  ZZ
DD  DD  BB  BB  GG  SS  YY  YY  MM  MM  BB  BB  LL  ZZ
DD  DD  BBBB BBBB  GG  SSSSSS  YY  YY  MM  MM  BBBB BBBB  LL  ZZ
DD  DD  BBBB BBBB  GG  SSSSSS  YY  YY  MM  MM  BBBB BBBB  LL  ZZ
DD  DD  BB  BB  GG  GG  YY  YY  MM  MM  BB  BB  LL  ZZ
DD  DD  BB  BB  GG  GG  YY  YY  MM  MM  BB  BB  LL  ZZ
DD  DD  BB  BB  GG  GG  YY  YY  MM  MM  BB  BB  LL  ZZ
DDDDDDDD  BBBB BBBB  GGGGGG  SSSSSSSS  YY  MM  MM  BBBB BBBB  LL  ZZ
DDDDDDDD  BBBB BBBB  GGGGGG  SSSSSSSS  YY  MM  MM  BBBB BBBB  LL  ZZ
LL  LL  SSSSSSSS
LL  LL  SSSSSSSS
LL  LL  SS
LL  LL  SS
LL  LL  SS
LL  LL  SS
LL  LL  SSSSSS
LL  LL  SSSSSS
LL  LL  SS
LL  LL  SS
LL  LL  SS
LL  LL  SS
LLLLLLLLLL  IIIIIII  SSSSSSSS
LLLLLLLLLL  IIIIIII  SSSSSSSS
```



```

1 0001 0 MODULE DBGSYMBLZ (IDENT = 'V04-000') =
2 0002 0
3 0003 1 BEGIN
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 WRITTEN BY
29 0029 1 Victoria Holt July, 1982
30 0030 1
31 0031 1 MODIFIED BY:
32 0032 1 R. Title
33 0033 1 Nov, 1982 \ > Added DBG$SYMD_TO_PRIMARY routine
34 0034 1 P. Sager /
35 0035 1
36 0036 1 P. Sager Dec, 1982 Rework parts of this module to work better
37 0037 1 for symbolization.
38 0038 1 R. Title Feb 1983 Added DUMP SAT and DUMP GLOBAL routines
39 0039 1 to aid in performance studies.
40 0040 1 R. Title May 1983 Redesign SAT as tree instead of linked
41 0041 1 list. Wrote SEARCH_BIN_SAT and SEARCH_PROG_SAT
42 0042 1 to utilize new data structure.
43 0043 1 W. Carrell Sep 1983 Update DBG$CURRENT_PRIMARY for self_referential
44 0044 1 records
45 0045 1
46 0046 1 MODULE FUNCTION
47 0047 1 This module contains all the routines necessary for parsing and
48 0048 1 executing the SYMBOLIZE command.
49 0049 1
50 0050 1
51 0051 1 REQUIRE 'SRC$:DBGPROLOG.REQ';
52 0185 1
53 0186 1 FORWARD ROUTINE
54 0187 1 DBG$NEXECUTE SYMBOLIZE, ! Performs action of SYMBOLIZE xxx.
55 0188 1 DBG$NPARSE SYMBOLIZE, ! Establishes parse network.
56 0189 1 DBG$SEARCH_BIN_SAT, ! Search module-level SAT
57 0190 1 DBG$SEARCH_PROG_SAT, ! Search program-level SAT

```


:	58	0191	1	DBG\$SEARCH_BIN SAT_REC,	:	Recursive search-sat routine
:	59	0192	1	DBG\$SEARCH_PROG SAT_REC,	:	Recursive search-sat routine
:	60	0193	1	DBG\$SEARCH_GLOBAL,	:	Searches Global Symbol Chain
:	61	0194	1	DBG\$DUMP_GLOBAL: NOVALUE,	:	Counts length of Global Symbol Chain
:	62	0195	1	DBG\$SEARCH SAT,	:	Searches SAT to symbolize address.
:	63	0196	1	DBG\$DUMP SAT: NOVALUE,	:	Dump information about SAT
:	64	0197	1	DBG\$SEARCH_VAX_CALL_STACK,	:	Searches VAX call stack.
:	65	0198	1	DBG\$SYMBOLIZE REG,	:	Attempts to symbolize register addr.
:	66	0199	1	DBG\$SYMID TO PRIMARY,	:	Turns SYMID,OFFSET into Primary
:	67	0200	1	DBG\$NEW_SYMBOLIZE,	:	New symbolization routine
:	68	0201	1	DBG\$PC_TO_SYMID,	:	Given address, search Module SAT or
:	69	0202	1		:	Global symbol chain for RST
:	70	0203	1	FIND_ROUTINE,	:	Finds routine corresponding to PC.
:	71	0204	1	GET_ADDRESS_DESC,	:	Return Address Descriptor from given
:	72	0205	1		:	address representation
:	73	0206	1	GSTADDRESS,	:	Get GST's address
:	74	0207	1	IS_SYMBOL_BOUND_TO_REG,	:	Determines if symbol is bound to reg.
:	75	0208	1	IS_SYMBOL_PSECT,	:	Determines if symbol is PSECT
:	76	0209	1	PRINT_REGNAME: NOVALUE,	:	Encodes register into output buffer.
:	77	0210	1	SEARCH_MODULE_SAT: NOVALUE,	:	Searches module level SAT for addr.
:	78	0211	1	BUILD_PRIMARY_SUBNODE,	:	Build primary subnode
:	79	0212	1	GET_RECORD_COMPONENT,	:	Get the record component from the list
:	80	0213	1	SYMBOLIZE_ARRAY_ELEMENT,	:	Symbolize to array element
:	81	0214	1	SYMBOLIZE_RECORD_COMPONENT;	:	Symbolize to record component

83	0215	1	EXTERNAL ROUTINE	
84	0216	1	DBG\$BUILD_INVOC RST,	Build RST entry with invocation num
85	0217	1	DBG\$BUILD-PRIMARY_SUBNODE: NOVALUE,	Build a subnode for a Primary
86	0218	1	DBG\$GET_TEMP_MEM,	Dynamic memory allocation.
87	0219	1	DBG\$GET_DST_NAME,	Get the name from DST record
88	0220	1	DBG\$MAKE_SKELETON_DESC,	
89	0221	1	DBG\$NEWLINE: NOVALUE,	Flush current print line.
90	0222	1	DBG\$NGET_ADDRESS,	Obtains an address from an address
91	0223	1		expression descriptor.
92	0224	1	DBG\$NMAKE_ARG_VECT,	Sets up message vector.
93	0225	1	DBG\$NMATCH,	Matches input to counted strings.
94	0226	1	DBG\$NNEXT_WORD,	Obtains next word of input.
95	0227	1	DBG\$NPARSE_ADDRESS,	Parses input and returns an AED.
96	0228	1	DBG\$PC_TO_LINE_LOOKUP,	Looks up line no. for PC addr.
97	0229	1	DBG\$PRIM_TO_VAL: NOVALUE,	Turn Primary into Value
98	0230	1	DBG\$PRINT: NOVALUE,	Print/format ASCII text.
99	0231	1	DBG\$PRINT_CONTROL: NOVALUE,	Sets up tabs for DBG\$PRINT.
100	0232	1	DBG\$PRINT-OFFSET: NOVALUE,	Prints value in correct radix.
101	0233	1	DBG\$PRINT-SYMBOL_PATHNAME: NOVALUE,	Print symbol name with pathname
102	0234	1	DBG\$STA_ADDRESS_TO_REGDESCR,	Return Register Descriptor for register
103	0235	1		address
104	0236	1	DBG\$STA_GETSYM-OFF,	Convert addr. to symbol and offset.
105	0237	1	DBG\$STA_LINE_NUM_RST,	Build RST Entry for line number
106	0238	1	DBG\$STA_REGISTER_NAME,	Convert Register Descriptor into ASCII
107	0239	1		name string for the register
108	0240	1	DBG\$STA_SETCONTEXT: NOVALUE,	Set the context for given SYMID
109	0241	1	DBG\$STA_SYM_KIND: NOVALUE,	Obtains "kind" of a symbol
110	0242	1	DBG\$STA_SYM_NAME: NOVALUE,	Translates symid to unqualified name.
111	0243	1	DBG\$STA_SYM_SIZE: NOVALUE,	Returns size associated with symid.
112	0244	1	DBG\$STA_SYM_TYPE: NOVALUE,	Determine TYPEID from SYMID
113	0245	1	DBG\$STA_SYM_VALUE: NOVALUE,	Get the value kind and value
114	0246	1	DBG\$STA_TYP_RECORD,	Get info about record
115	0247	1	DBG\$STA_TYPEFCODE,	Obtain FCODE from SYMID
116	0248	1	DBG\$STA-VARIANT_SELECT,	Get a pointer to a variant set RST
117	0249	1	SYSSFAO;	Formatter.
118	0250	1		
119	0251	1	EXTERNAL	
120	0252	1	DBG\$GL_CURRENT_PRIMARY,	Pointer to the primary being processed
121	0253	1	DBG\$GB_RADIX: VECTOR[3, BYTE],	Radix settings
122	0254	1	DBG\$GL_ARRSUB_FLAG,	Used for controlling array subscripting
123	0255	1	DBG\$GL_RECCMP_FLAG,	Used for controlling record component
124	0256	1	DBG\$GB_LANGUAGE: BYTE,	Current language setting
125	0257	1	DBG\$FINAL_HANDL,	Call frame exception handler.
126	0258	1	DBG\$REG_VALUES: VECTOR[, LONG],	Vector of user register values in the
127	0259	1		current context.
128	0260	1	DBG\$RUNFRAME: BLOCK[, BYTE],	The current user run frame.
129	0261	1	RST\$START_ADDR: REF RST\$ENTRY,	Ptr to 1st module RST entry.
130	0262	1	SAT\$START_ADDR;	Address of 1st SAT entry on program
131	0263	1		SAT chain.
132	0264	1		
133	0265	1	LITERAL	
134	0266	1	TAB = 4;	Indentation level for DBG\$PRINT_CONTROL.
135	0267	1		
136	0268	1	OWN	
137	0269	1	OWN_MODULE_BEGIN,	Begin address of the module
138	0270	1	OWN_MODULE_END;	End address of the module
139	0271	1		

DBGSYMBLZ
V04-000

N 2
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 4
(3)

```
: 141      0272 1 MACRO
: 142      0273 1 PRINT BIT_OFFSET(BIT_OFFSET) =
: 143      0274 1 IF .BIT_OFFSET NEQ 0
: 144      0275 1 THEN
: 145      0276 1 BEGIN
: 146      0277 1   DBG$PRINT (UPLIT BYTE (%ASCIC '<')));
: 147      0278 1   DBG$PRINT (UPLIT BYTE (%ASCIC '!UL')
: 148      0279 1   DBG$PRINT (UPLIT BYTE (%ASCIC ', 0, 0>');BIT_OFFSET);
: 149      0280 1 END;%
: 150      0281 1
```



```
152 0282 1 GLOBAL ROUTINE DBG$NEXECUTE_SYMBOLIZE (VERB_NODE, MESSAGE_VECT) =
153 0283 1
154 0284 1 FUNCTION
155 0285 1 This routine performs the action associated with the SYMBOLIZE xxx
156 0286 1 command.
157 0287 1
158 0288 1 A virtual address is obtained from the command execution tree. From
159 0289 1 there, the routine determines what kind of address it is: a register
160 0290 1 address, static address, or stack address. After making that
161 0291 1 determination, the routine attempts to locate symbols bound to that
162 0292 1 address by searching the set module chain (in the case of a register
163 0293 1 address), by searching the SAT (in the case of a static address), or by
164 0294 1 searching the call stack (in the case of a stack address). (Symbols
165 0295 1 whose addresses are too complicated to compute in this last search are
166 0296 1 ignored.) Finally the Global Symbol Chain is searched.
167 0297 1
168 0298 1 INPUTS
169 0299 1 VERB_NODE - A longword containing the address of the command
170 0300 1 execution tree verb node.
171 0301 1
172 0302 1 MESSAGE_VECT - The address of a longword to contain the address
173 0303 1 of a standard message argument vector on error.
174 0304 1
175 0305 1 OUTPUTS
176 0306 1 An unsigned integer longword completion code.
177 0307 1
178 0308 1 On success, all possible symbolizations for the virtual address are
179 0309 1 sent to the output device.
180 0310 1
181 0311 1 On failure (no symbolization found), a message to that effect is printed.
182 0312 1
183 0313 1
184 0314 2 BEGIN
185 0315 2
186 0316 2 MAP
187 0317 2 VERB_NODE: REF DBG$VERB_NODE; ! Ptr to verb node.
188 0318 2
189 0319 2 LOCAL
190 0320 2 ADDRESS_DESC: REF DBG$ADDRESS_DESC, ! Address descriptor.
191 0321 2 BIT_OFFSET, ! Bit offset from SYMID
192 0322 2 NOUN_NODE: REF DBG$NOUN_NODE, ! Ptr to noun node.
193 0323 2 SYMID: REF RST$ENTRY; ! Unique symbol identifier.
194 0324 2
195 0325 2
196 0326 2 ! Recover the noun node.
197 0327 2
198 0328 2 NOUN_NODE = .VERB_NODE[DBG$L_VERB_OBJECT_PTR];
199 0329 2
200 0330 2
201 0331 2 ! Loop through tree.
202 0332 2
203 0333 2 WHILE .NOUN_NODE NEQ 0 DO
204 0334 2 BEGIN
205 0335 2
206 0336 2
207 0337 2 ! Set up tabs.
208 0338 2
```



```
209 0339 3
210 0340 3
211 0341 3
212 0342 3
213 0343 3
214 0344 3
215 0345 3
216 0346 3
217 0347 3
218 0348 3
219 0349 3
220 0350 3
221 0351 3
222 0352 3
223 0353 3
224 0354 4
225 0355 4
226 0356 4
227 0357 4
228 0358 4
229 0359 4
230 0360 4
231 0361 4
232 0362 3
233 0363 3
234 0364 3
235 0365 3
236 0366 3
237 0367 3
238 0368 2
239 0369 2
240 0370 2
241 0371 2
242 0372 2
243 0373 2
244 0374 1

!
DBG$PRINT_CONTROL (DBG$K_PRT_RESET);

! Recover address descriptor.
ADDRESS_DESC = .NOUN_NODE[DBG$L_NOUN_VALUE];

! Call the routine that actually does all the work.
! If no symbolization is possible for the address, print a message to
! that effect.
IF NOT DBG$STA_GETSYMOFF (.ADDRESS_DESC, SYMID, BIT_OFFSET, TRUE)
THEN
BEGIN
DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS_DESC[DBG$L_ADDRESS_BYTE_ADDR]);
PRINT BIT_OFFSET(ADDRESS_DESC[DBG$L_ADDRESS_BIT_OFFSET]);
DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
DBG$NEWLINE();
DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
DBG$PRINT (UPLIT BYTE (%ASCIC 'no symbolization'), 0);
DBG$NEWLINE();
END;

! Get next noun node.
NOUN_NODE = .NOUN_NODE[DBG$L_NOUN_LINK];
END;

! Routine end.
RETURN ST$K_SUCCESS;
END;
```

```

.TITLE DBGSYMBOLZ
.IDENT \V04-000\

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

4C 58 21 20 73 73 65 72 64 64 61 0B 00000 P.AAA: .ASCII <11>\address !XL\
3C 01 0000C P.AAB: .ASCII <1>\<\
21 03 0000E P.AAC: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 00012 P.AAD: .ASCII <7>\, 0, 0>\
20 3A 02 0001A P.AAE: .ASCII <2>\: \
69 74 61 7A 69 6C 6F 62 6D 79 73 20 6F 6E 10 0001D P.AAF: .ASCII <16>\no symbolization\
6E 6F 0002C

.PSECT DBG$OWN,NOEXE, PIC,2

00000 OWN_MODULE_BEGIN:
.B[KB 4
00004 OWN_MODULE_END:
.B[KB 4
```


				.EXTRN	DBG\$BUILD_INVOC_RST	
				.EXTRN	DBG\$BUILD_PRIMARY_SUBNODE	
				.EXTRN	DBG\$GET_TEMP_MEM	
				.EXTRN	DBG\$GET_DST_NAME	
				.EXTRN	DBG\$MAKE_SKELETON_DESC	
				.EXTRN	DBG\$NEWLINE, DBG\$NGET_ADDRESS	
				.EXTRN	DBG\$NMAKE_ARG_VECT	
				.EXTRN	DBG\$NMATCH, DBG\$NNEXT_WORD	
				.EXTRN	DBG\$NPARSE_ADDRESS	
				.EXTRN	DBG\$PC_TO_LINE_LOOKUP	
				.EXTRN	DBG\$PRIM_TO_VAL	
				.EXTRN	DBG\$PRINT, DBG\$PRINT_CONTROL	
				.EXTRN	DBG\$PRINT_OFFSET	
				.EXTRN	DBG\$PRINT_SYMBOL_PATHNAME	
				.EXTRN	DBG\$STA_ADDRESS_TO_REGDESCR	
				.EXTRN	DBG\$STA_GETSYM_OFF	
				.EXTRN	DBG\$STA_LINE_NUM_RST	
				.EXTRN	DBG\$STA_REGISTER_NAME	
				.EXTRN	DBG\$STA_SETCONTEXT	
				.EXTRN	DBG\$STA_SYMKIND	
				.EXTRN	DBG\$STA_SYMNAME	
				.EXTRN	DBG\$STA_SYMSIZE	
				.EXTRN	DBG\$STA_SYMTYPE	
				.EXTRN	DBG\$STA_SYMVALUE	
				.EXTRN	DBG\$STA_TYP_RECORD	
				.EXTRN	DBG\$STA_TYPEFCODE	
				.EXTRN	DBG\$STA_VARIANT_SELECT	
				.EXTRN	SYSS\$FAO, DBG\$GL_CURRENT_PRIMARY	
				.EXTRN	DBG\$GB_RADIX, DBG\$GL_ARRSUB_FLAG	
				.EXTRN	DBG\$GL_RECCMP_FLAG	
				.EXTRN	DBG\$GB_LANGUAGE	
				.EXTRN	DBG\$FINAL_HANDL	
				.EXTRN	DBG\$REG_VALUES, DBG\$RUNFRAME	
				.EXTRN	RST\$START_ADDR, SAT\$START_ADDR	
				.PSECT	DBG\$CODE, NOWRT, SHR, PIC, 0	
				.ENTRY	DBG\$NEXECUTE_SYMBOLIZE, Save R2, R3, R4, R5, -	0282
					R6, R7	
				MOVAB	DBG\$PRINT_CONTROL, R7	
				MOVAB	DBG\$NEWLINE, R6	
				MOVAB	DBG\$PRINT, R5	
				MOVAB	P.AAA, R4	
				SUBL2	#8, SP	
				MOVL	VERB_NODE, R0	0329
				MOVL	8(R0), NOUN_NODE	
				BEQL	4\$	0334
				PUSHL	#5	0340
				CALLS	#1, DBG\$PRINT_CONTROL	
				MOVL	(NOUN_NODE), ADDRESS_DESC	0345
				PUSHL	#1	0352
				PUSHAB	BIT OFFSET	
				PUSHAB	SYMTD	
				PUSHL	ADDRESS_DESC	
				CALLS	#4, DBG\$STA_GETSYM_OFF	
				BLBS	R0, 3\$	

				00FC 00000	
57	00000000G	00	9E	00002	
56	00000000G	00	9E	00009	
55	00000000G	00	9E	00010	
54	00000000G	EF	9E	00017	
5E		08	C2	0001E	
50	04	AC	D0	00021	
53	08	A0	D0	00025	
		5E	13	00029	1\$:
		05	DD	0002B	
67		01	FB	0002D	
52		63	D0	00030	
		01	DD	00033	
	04	AE	9F	00035	
	0C	AE	9F	00038	
		52	DD	0003B	
00000000G	00	04	FB	0003D	
	3C	50	EB	00044	

DBGSYMBLZ
V04-000

E 3
16-Sep-1984 02:41:46 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:51 [DEBUG.SRC]DBGSYMBLZ.B32;1

Page 8
(4)

		62	DD	00047	PUSHL	(ADDRESS_DESC)	: 0355
		54	DD	00049	PUSHL	R4	: 0356
65		02	FB	0004B	CALLS	#2, DBG\$PRINT	: 0357
	04	A2	D5	0004E	TSTL	4(ADDRESS_DESC)	: 0358
		15	13	00051	BEQL	2\$	: 0359
	0C	A4	9F	00053	PUSHAB	P.AAB	: 0360
65		01	FB	00056	CALLS	#1, DBG\$PRINT	: 0361
	04	A2	DD	00059	PUSHL	4(ADDRESS_DESC)	: 0362
	0E	A4	9F	0005C	PUSHAB	P.AAC	: 0363
65		02	FB	0005F	CALLS	#2, DBG\$PRINT	: 0364
	12	A4	9F	00062	PUSHAB	P.AAD	: 0365
65		01	FB	00065	CALLS	#1, DBG\$PRINT	: 0366
	1A	A4	9F	00068	PUSHAB	P.AAE	: 0367
65		01	FB	0006B	CALLS	#1, DBG\$PRINT	: 0368
66		00	FB	0006E	CALLS	#0, DBG\$NEWLINE	: 0369
		04	DD	00071	PUSHL	#4	: 0370
		01	DD	00073	PUSHL	#1	: 0371
67		02	FB	00075	CALLS	#2, DBG\$PRINT_CONTROL	: 0372
		7E	D4	00078	CLRL	-(SP)	: 0373
	1D	A4	9F	0007A	PUSHAB	P.AAF	: 0374
65		02	FB	0007D	CALLS	#2, DBG\$PRINT	: 0375
66		00	FB	00080	CALLS	#0, DBG\$NEWLINE	: 0376
53	0B	A3	D0	00083	MOVL	8(NOUN_NODE), NOUN_NODE	: 0377
		A0	11	00087	BRB	1\$	: 0378
50		01	D0	00089	MOVL	#1, R0	: 0379
		04	0008C	RET			: 0380

; Routine Size: 141 bytes, Routine Base: DBG\$CODE + 0000


```
246 0375 1 GLOBAL ROUTINE DBG$NPARSE_SYMBOLIZE (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
247 0376 1
248 0377 1 FUNCTION
249 0378 1 Establishes the parse network for the SYMBOLIZE command, consisting of
250 0379 1 a verb_node and a linked list of noun nodes. Each noun node points to
251 0380 1 an address that is to be symbolized, if possible.
252 0381 1
253 0382 1 INPUTS
254 0383 1 INPUT_DESC - a longword containing the address of the command
255 0384 1 input descriptor. This should point to the first
256 0385 1 character after the SYMBOLIZE keyword.
257 0386 1
258 0387 1 VERB_NODE - A longword containing the address of the command
259 0388 1 execution tree verb node.
260 0389 1
261 0390 1 MESSAGE_VECT - The address of a longword to contain the address
262 0391 1 of a standard message argument vector on error.
263 0392 1
264 0393 1 OUTPUTS
265 0394 1 An unsigned integer longword completion code.
266 0395 1
267 0396 1 The pointer to the input descriptor is advanced to the end of the
268 0397 1 complete symbolize command. Normally this means pointing to the
269 0398 1 terminating carriage return.
270 0399 1
271 0400 1 On success, the command execution tree is constructed; ie, noun nodes
272 0401 1 containing the addresses descriptor to be symbolized are linked to
273 0402 1 the verb node.
274 0403 1
275 0404 1 On failure, the routine signals an error.
276 0405 1
277 0406 1
278 0407 2 BEGIN
279 0408 2
280 0409 2 MAP
281 0410 2 INPUT_DESC: REF BLOCK [,BYTE], ! Ptr. to input desc.
282 0411 2 VERB_NODE: REF DBG$VERB_NODE; ! Ptr. to verb node.
283 0412 2
284 0413 2 LOCAL
285 0414 2 ADDRPTR, ! Pointer to an address
286 0415 2 ! representation
287 0416 2 ADDRESS_DESC: REF DBG$ADDRESS_DESC, ! Pointer to an Addr. Desc.
288 0417 2 NOUN_NODE: REF DBG$NOUN_NODE, ! Object of verb
289 0418 2 NEXT_NOUN_NODE: REF DBG$NOUN_NODE, ! Temporary object.
290 0419 2 STATUS;
291 0420 2
292 0421 2
293 0422 2 BIND
294 0423 2 DBG$CS_CR = UPLIT BYTE (1, DBG$K_CAR_RETURN),
295 0424 2 DBG$CS_COMMA = UPLIT BYTE (1, DBG$K_COMMA);
296 0425 2
297 0426 2
298 0427 2 ! Need to be sure that the user has provided some sort of input after
299 0428 2 ! the verb. If not, construct message_vect and return.
300 0429 2
301 0430 2 IF DBG$NMATCH (.INPUT_DESC, DBG$CS_CR, 1)
302 0431 2 THEN
```



```
303 0432 2 SIGNAL (DBG$_NEEDMORE);
304 0433
305 0434
306 0435 ! Construct the first noun node.
307 0436
308 0437 NOUN_NODE = DBG$GET_TEMPMEM (DBG$K_NOUN_NODE_SIZE);
309 0438
310 0439
311 0440 ! Link in the noun node.
312 0441
313 0442 VERB_NODE[DBG$L_VERB_OBJECT_PTR] = .NOUN_NODE;
314 0443
315 0444
316 0445 ! Loop until input is exhausted.
317 0446
318 0447 WHILE (NOT DBG$NMATCH (.INPUT_DESC, DBG$CS_CR, 1)) AND
319 0448 (.INPUT_DESC[DSC$_LENGTH] NEQ 0) DO
320 0449 BEGIN
321 0450
322 0451
323 0452 ! Attempt to parse the input_desc. If successful, dbg$nparsed_address
324 0453 ! returns the address of an address expression descriptor.
325 0454
326 0455 STATUS = DBG$NPARSE_ADDRESS (.INPUT_DESC, ADDRPTR,
327 0456 .DBG$GB_RADIX[DBG$B_RADIX_INPUT],
328 0457 TOKEN$K_TERM_COMMA, .MESSAGE_VECT);
329 0458
330 0459 SELECTONE .STATUS OF
331 0460 SET
332 0461
333 0462 ! If successful, then get the address bound to the entity
334 0463 ! described by the address expression descriptor. Success
335 0464 ! also implies the end of input.
336 0465
337 0466 [ST$K_SUCCESS]:
338 0467 BEGIN
339 0468 ADDRESS_DESC = DBG$GET_TEMPMEM (DBG$K_ADDRESS_DESC_SIZE);
340 0469 IF NOT GET_ADDRESS_DESC (.ADDRPTR, .ADDRESS_DESC, .MESSAGE_VECT)
341 0470 THEN
342 0471 RETURN ST$K_SEVERE;
343 0472 NOUN_NODE[DBG$L_NOUN_VALUE] = .ADDRESS_DESC;
344 0473 END;
345 0474
346 0475
347 0476 ! If a warning is returned, a comma may have been discovered in
348 0477 ! the input stream. Check for this. If there is no comma, a
349 0478 ! syntax error has occurred. If a comma is present, there must
350 0479 ! be more input or else error. If all is OK, then get the address
351 0480 ! bound to the entity described by the address expression
352 0481 ! descriptor. A warning implies further input, so must link in
353 0482 ! another noun node.
354 0483
355 0484 [ST$K_WARNING]:
356 0485 BEGIN
357 0486 IF NOT DBG$NMATCH (.INPUT_DESC, DBG$CS_COMMA, 1)
358 0487 THEN
359 0488 SIGNAL (DBG$_CMDSYNERR, 1, DBG$NNEXT_WORD (.INPUT_DESC));
```



```

360 0489 4
361 0490 4
362 0491 4
363 0492 4
364 0493 4
365 0494 4
366 0495 4
367 0496 4
368 0497 4
369 0498 4
370 0499 4
371 0500 4
372 0501 4
373 0502 3
374 0503 3
375 0504 3
376 0505 3
377 0506 3
378 0507 3
379 0508 4
380 0509 4
381 0510 3
382 0511 3
383 0512 3
384 0513 3
385 0514 3
386 0515 3
387 0516 3
388 0517 2
389 0518 2
390 0519 2
391 0520 2
392 0521 2
393 0522 2
394 0523 1

```

```

IF DBG$NMATCH (.INPUT_DESC, DBG$CS_CR, 1)
THEN
    SIGNAL (DBG$_NEEDMORE);

ADDRESS_DESC = DBG$GET_TEMPMEM (DBG$K_ADDRESS_DESC_SIZE);
IF NOT GET_ADDRESS_DESC (.ADDRPTR, .ADDRESS_DESC, .MESSAGE_VECT)
THEN
    RETURN ST$K_SEVERE;
NOUN_NODE[DBG$L_NOUN_VALUE] = .ADDRESS_DESC;
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 ST$K_SEVERE;
END;

TES;

! End of WHILE loop.
END;

! End of routine.
RETURN ST$K_SUCCESS;
END;

```

```

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
0D 01 0002E P.AAG: .BYTE 1, 13
2C 01 00030 P.AAH: .BYTE 1, 44

```

```

DBG$CS_CR= P.AAG
DBG$CS_COMMA= P.AAH

```

```

.PSECT DBG$CODE,NOWRT, SHR, PIC,0
07FC 00000 .ENTRY DBG$NPARSE_SYMBOLIZE, Save R2,R3,R4,R5,R6,- : 0375
5A 00000000G 00 9E 00002 MOVAB LIB$SIGNAL, R10
59 00000000G EF 9E 00009 MOVAB DBG$CS_CR, R9
58 00000000G 00 9E 00010 MOVAB DBG$GET_TEMPMEM, R8
57 00000000G 00 9E 00017 MOVAB DBG$NMATCH, R7
5E 04 C2 0001E SUBL2 #4, SP
01 DD 00021 PUSHL #1

```

0430

		59	DD	00023	PUSHL	R9		
52	04	AC	DD	00025	MOVL	INPUT_DESC, R2		
		52	DD	00029	PUSHL	R2		
67		03	FB	0002B	CALLS	#3, DBG\$NMATCH		
09		50	E9	0002E	BLBC	R0, 1\$		
	000280D0	8F	DD	00031	PUSHL	#164048		0432
6A		01	FB	00037	CALLS	#1, LIB\$SIGNAL		
		04	DD	0003A	1\$: PUSHL	#4		0437
68		01	FB	0003C	CALLS	#1, DBG\$GET_TEMPMEM		
53		50	DD	0003F	MOVL	R0, NOUN_NODE		
50	08	AC	DD	00042	MOVL	VERB_NODE, R0		0442
08	A0	53	DD	00046	MOVL	NOUN_NODE, 8(R0)		
		01	DD	0004A	2\$: PUSHL	#1		0447
	0204	8F	BB	0004C	PUSHR	#^M<R2,R9>		
67		03	FB	00050	CALLS	#3, DBG\$NMATCH		
03		50	E9	00053	BLBC	R0, 4\$		
		00AD	31	00056	3\$: BRW	10\$		
		62	B5	00059	4\$: TSTW	(R2)		0448
		F9	13	0005B	BEQL	3\$		
	0C	AC	DD	0005D	PUSHL	MESSAGE_VECT		0457
		01	DD	00060	PUSHL	#1		0455
7E	00000000G	00	9A	00062	MOVZBL	DBG\$GB_RADIX, -(SP)		0456
	0C	AE	9F	00069	PUSHAB	ADDRPTR		0455
		52	DD	0006C	PUSHL	R2		
00000000G	00	05	FB	0006E	CALLS	#5, DBG\$NPARSE_ADDRESS		
	55	50	DD	00075	MOVL	R0, STATUS		
	01	55	D1	00078	CPL	STATUS, #1		0466
		1D	12	0007B	BNEQ	6\$		
		02	DD	0007D	PUSHL	#2		0468
68		01	FB	0007F	CALLS	#1, DBG\$GET_TEMPMEM		
54		50	DD	00082	MOVL	R0, ADDRESS_DESC		
	0C	AC	DD	00085	PUSHL	MESSAGE_VECT		0469
		54	DD	00088	PUSHL	ADDRESS_DESC		
	08	AE	DD	0008A	PUSHL	ADDRPTR		
0000V	CF	03	FB	0008D	CALLS	#3, GET_ADDRESS_DESC		
	6D	50	E9	00092	BLBC	R0, 9\$		
	63	54	DD	00095	MOVL	ADDRESS_DESC, (NOUN_NODE)		0472
		B0	11	00098	5\$: BRB	2\$		0458
		55	D5	0009A	6\$: TSTL	STATUS		0484
		64	12	0009C	BNEQ	9\$		
		01	DD	0009E	PUSHL	#1		0486
	02	A9	9F	000A0	PUSHAB	DBG\$CS_COMMA		
		52	DD	000A3	PUSHL	R2		
67		03	FB	000A5	CALLS	#3, DBG\$NMATCH		
16		50	E8	000A8	BLBS	R0, 7\$		
		52	DD	000AB	PUSHL	R2		0488
00000000G	00	01	FB	000AD	CALLS	#1, DBG\$NNEXT_WORD		
		50	DD	000B4	PUSHL	R0		
		01	DD	000B6	PUSHL	#1		
	00028EB8	8F	DD	000B8	PUSHL	#167608		
6A		03	FB	000BE	CALLS	#3, LIB\$SIGNAL		0490
		01	DD	000C1	7\$: PUSHL	#1		
	0204	8F	BB	000C3	PUSHR	#^M<R2,R9>		
67		03	FB	000C7	CALLS	#3, DBG\$NMATCH		
09		50	E9	000CA	BLBC	R0, 8\$		
	000280D0	8F	DD	000CD	PUSHL	#164048		0492
6A		01	FB	000D3	CALLS	#1, LIB\$SIGNAL		

	68		02	DD	000D6	8%:	PUSHL	#2		0494
	54		01	FB	000D8		CALLS	#1, DBG\$GET_TEMP MEM		
			50	DD	000DB		MOVL	R0, ADDRESS_DESC		
		0C	AC	DD	000DE		PUSHL	MESSAGE_VECT		0495
			54	DD	000E1		PUSHL	ADDRESS_DESC		
		08	AE	DD	000E3		PUSHL	ADDRPTR		
0000V	CF		03	FB	000E6		CALLS	#3, GET_ADDRESS_DESC		
	14		50	E9	000EB		BLBC	R0, 9%		
	63		54	DD	000EE		MOVL	ADDRESS_DESC, (NOUN_NODE)		0498
			04	DD	000F1		PUSHL	#4		0499
	68		01	FB	000F3		CALLS	#1, DBG\$GET_TEMP MEM		
	56		50	DD	000F6		MOVL	R0, NEXT_NOUN_NODE		
08	A3		56	DD	000F9		MOVL	NEXT_NOUN_NODE, 8(NOUN_NODE)		0500
	53		56	DD	000FD		MOVL	NEXT_NOUN_NODE, NOUN_NODE		0501
			96	11	00100		BRB	5%		0458
	50		04	DD	00102	9%:	MOVL	#4, R0		0509
					04	00105	RET			
	50		01	DD	00106	10%:	MOVL	#1, R0		0522
					04	00109	RET			0523

; Routine Size: 266 bytes, Routine Base: DBG\$CODE + 008D


```
396 0524 1 GLOBAL ROUTINE DBG$SEARCH_BIN_SAT (BASE_NODE, ADDRESS,  
397 0525 1 MACRO_OR_SHARE_FLAG, PSECT_FLAG)=  
398 0526 1  
399 0527 1 FUNCTION  
400 0528 1 This routine searches a linked SAT list starting at BASE_NODE  
401 0529 1 and returns a pointer to a SAT entry containing ADDRESS (or zero  
402 0530 1 if no SAT entry was found).  
403 0531 1  
404 0532 1 This routine is called to search a module-level SAT. It is called  
405 0533 1 by DBG$SEARCH_PROG_SAT.  
406 0534 1  
407 0535 1 The SAT chain that this routine searches is a linked list which  
408 0536 1 is sorted by start address of the symbol. We just loop through  
409 0537 1 the list looking for the static symbol which contains the given  
410 0538 1 address and is the best match.  
411 0539 1  
412 0540 1 On a linear SAT, which we currently have,  
413 0541 1 this routine is functionally equivalent to DBG$SEARCH_BIN_SAT_REC.  
414 0542 1 The REC routine is more general in that it also accepts a SAT  
415 0543 1 which is organized as a binary tree. However, until we start  
416 0544 1 building binary tree SATs, we will stick with this routine.  
417 0545 1 The REC routine differs in implementation in that it is recursive,  
418 0546 1 which can be a problem in that it uses a lot of stack.  
419 0547 1  
420 0548 1 INPUTS  
421 0549 1 BASE_NODE - The base of the SAT list  
422 0550 1 ADDRESS - The desired address.  
423 0551 1 MACRO_OR_SHARE_FLAG- True for language MACRO and also true  
424 0552 1 for static address chains that come from shareable  
425 0553 1 images.  
426 0554 1 PSECT_FLAG- Says whether it is OK to symbolize to a PSECT.  
427 0555 1  
428 0556 1 OUTPUTS  
429 0557 1 A pointer to the found SAT entry is returned.  
430 0558 1 Zero is returned if no SAT entry was found.  
431 0559 1  
432 0560 2 BEGIN  
433 0561 2 LOCAL  
434 0562 2 BEST SATPTR: REF SAT$ENTRY,  
435 0563 2 SATPTR: REF SAT$ENTRY;  
436 0564 2  
437 0565 2 ! Loop through the SAT chain.  
438 0566 2 !  
439 0567 2 BEST SATPTR = 0;  
440 0568 2 SATPTR = .BASE_NODE;  
441 0569 2 WHILE .SATPTR NEQ 0 DO  
442 0570 2 BEGIN  
443 0571 2  
444 0572 2 ! If the address we are looking for is before the current SAT entry  
445 0573 2 ! then stop searching.  
446 0574 2 !  
447 0575 2 IF .ADDRESS LSSA .SATPTR[SAT$L_START]  
448 0576 2 THEN  
449 0577 2 EXITLOOP;  
450 0578 2  
451 0579 2  
452 0580 2
```



```
: 453      0581      3
: 454      0582      3
: 455      0583      3
: 456      0584      3
: 457      0585      3
: 458      0586      4
: 459      0587      3
: 460      0588      4
: 461      0589      3
: 462      0590      3
: 463      0591      3
: 464      0592      3
: 465      0593      3
: 466      0594      3
: 467      0595      2
: 468      0596      2
: 469      0597      2
: 470      0598      2
: 471      0599      2
: 472      0600      2
: 473      0601      2
: 474      0602      2
: 475      0603      1
```

```
! Choose this symbol if its range covers the address. Don't choose
! psects unless PSECT_FLAG is set. Don't check the end address
! for language MACRO or for symbols from shareable images.
IF .PSECT_FLAG OR (NOT IS_SYMBOL_PSECT(.SATPTR[SAT$L_RSTPTR]))
THEN
    IF .MACRO_OR_SHARE_FLAG OR (.ADDRESS LEQA .SATPTR[SAT$L_END])
    THEN
        BEST_SATPTR = .SATPTR;

! Link to next SAT entry.
SATPTR = .SATPTR[SAT$L_FLINK];
END;

! Return the SATPTR we have found. This will be zero if we
! fell out of the loop not finding a match.
RETURN .BEST_SATPTR;
END;
```

```
000C 00000
53 D4 00002
04 52 04 AC D0 00004
29 13 00008 1$:
04 A2 08 AC D1 0000A
22 1F 0000F
0B 10 AC E8 00011
0C A2 DD 00015
0000V CF 01 FB 00018
0E 50 E8 0001D
07 0C AC E8 00020 2$:
08 A2 08 AC D1 00024
03 1A 00029
53 52 D0 0002B 3$:
52 62 D0 0002E 4$:
D5 11 00031
50 53 D0 00033 5$:
04 00036
```

```
.ENTRY DBG$SEARCH_BIN_SAT, Save R2,R3
CLRL BEST_SATPTR
MOVL BASE_NODE, SATPTR
BEQL 5$
CMPL ADDRESS, 4(SATPTR)
BLSSU 5$
BLBS PSECT_FLAG, 2$
PUSHL 12(SATPTR)
CALLS #1, IS_SYMBOL_PSECT
BLBS R0, 4$
BLBS MACRO_OR_SHARE_FLAG, 3$
CMPL ADDRESS, 8(SATPTR)
BGTRU 4$
MOVL SATPTR, BEST_SATPTR
MOVL (SATPTR), SATPTR
BRB 1$
MOVL BEST_SATPTR, R0
RET
```

```
: 0524
: 0568
: 0569
: 0570
: 0577
: 0586
: 0588
: 0590
: 0595
: 0570
: 0602
: 0603
```

; Routine Size: 55 bytes, Routine Base: DBG\$CODE + 0197


```

: 477      0604 1 GLOBAL ROUTINE DBG$SEARCH_PROG_SAT (BASE_NODE, ADDRESS, BYTE_OFFSET) =
: 478      0605 1
: 479      0606 1 FUNCTION
: 480      0607 1     This routine searches the program-level
: 481      0608 1     SAT list starting at BASE_NODE
: 482      0609 1     and returns a pointer to the SAT entry which contains ADDRESS
: 483      0610 1     and whose start address is closest to ADDRESS. (or zero
: 484      0611 1     if no SAT entry was found).
: 485      0612 1
: 486      0613 1     This routine walks the linear list of program-level SATs, finding
: 487      0614 1     one whose range covers the given address. Then, for each one of
: 488      0615 1     those that it finds, it calls DBG$SEARCH_BIN_SAT to search the
: 489      0616 1     module-level SAT for the closest symbol whose address range
: 490      0617 1     covers the given address.
: 491      0618 1
: 492      0619 1     On a linear SAT, which we currently have,
: 493      0620 1     this routine is functionally equivalent to DBG$SEARCH_PROG_SAT_REC.
: 494      0621 1     The REC routine is more general in that it also accepts a SAT
: 495      0622 1     which is organized as a binary tree. However, until we start
: 496      0623 1     building binary tree SATs, we will stick with this routine.
: 497      0624 1     The REC routine differs in implementation in that it is recursive,
: 498      0625 1     which can be a problem in that it uses a lot of stack.
: 499      0626 1
: 500      0627 1 INPUTS
: 501      0628 1     BASE_NODE - The base of the SAT list
: 502      0629 1     ADDRESS - The desired address.
: 503      0630 1     BYTE_OFFSET - Return the byte offset associated with the symbolization
: 504      0631 1     in this address.
: 505      0632 1
: 506      0633 1 OUTPUTS
: 507      0634 1     A pointer to the found SAT entry is returned.
: 508      0635 1     Zero is returned if no SAT entry was found.
: 509      0636 1
: 510      0637 2 BEGIN
: 511      0638 2 LOCAL
: 512      0639 2     BEST_SATPTR: REF SAT$ENTRY,
: 513      0640 2     BEST_OFFSET,
: 514      0641 2     OFFSET,
: 515      0642 2     MODRSTPTR: REF RST$ENTRY,
: 516      0643 2     SATPTR: REF SAT$ENTRY,
: 517      0644 2     TEMP_SATPTR: REF SAT$ENTRY;
: 518      0645 2
: 519      0646 2
: 520      0647 2 ! Loop through the SAT chain.
: 521      0648 2 !
: 522      0649 2 BEST_SATPTR = 0;
: 523      0650 2 BEST_OFFSET = 999999999;
: 524      0651 2 SATPTR = .BASE_NODE;
: 525      0652 2 WHILE .SATPTR NEQ 0 DO
: 526      0653 2 BEGIN
: 527      0654 2
: 528      0655 2
: 529      0656 2 ! If the address we are looking for is before the current SAT entry
: 530      0657 2 ! then terminate the search.
: 531      0658 2 !
: 532      0659 2 IF .ADDRESS LSSA .SATPTR[SAT$L_START]
: 533      0660 2 THEN
```



```

534 0661 3      EXITLOOP;
535 0662 3
536 0663 3
537 0664 3      ! If the address we are looking for lies within the current
538 0665 3      ! program-level SAT then search the module-level SAT.
539 0666 3
540 0667 3      IF .ADDRESS LEQA .SATPTR[SAT$L_END]
541 0668 3      THEN
542 0669 4          BEGIN
543 0670 4              MODRSTPTR = .SATPTR[SAT$L_RSTPTR];
544 0671 4              IF .MODRSTPTR[RST$V_MODSET]
545 0672 4              THEN
546 0673 5                  BEGIN
547 0674 5                      TEMP_SATPTR = DBG$SEARCH_BIN SAT (
548 0675 5                          .MODRSTPTR[RST$L_SAT_PTR],
549 0676 5                          .ADDRESS,
550 0677 5                          (.MODRSTPTR[RST$B_LANGUAGE] EQL DBG$K_MACRO) OR
551 0678 5                          .MODRSTPTR[RST$V_SHARE_IMAGE],
552 0679 5                          FALSE);
553 0680 5
554 0681 5                  ! If the module-level search yielded a candidate, choose
555 0682 5                  ! it if it is a closer match than the best so far.
556 0683 5
557 0684 5                  IF .TEMP_SATPTR NEQ 0
558 0685 5                  THEN
559 0686 6                      BEGIN
560 0687 6                          OFFSET = .ADDRESS - .TEMP_SATPTR[SAT$L_START];
561 0688 6                          IF .OFFSET LSS .BEST_OFFSET
562 0689 6                          THEN
563 0690 7                              BEGIN
564 0691 7                                  BEST_SATPTR = .TEMP_SATPTR;
565 0692 7                                  BEST_OFFSET = .OFFSET;
566 0693 7                              END
567 0694 6                          ELSE IF .OFFSET EQL .BEST_OFFSET
568 0695 6                          THEN
569 0696 7                              BEGIN
570 0697 7                                  IF .TEMP_SATPTR[SAT$L_END] LSS .BEST_SATPTR[SAT$L_END]
571 0698 7                                  THEN
572 0699 8                                      BEGIN
573 0700 8                                          BEST_SATPTR = .TEMP_SATPTR;
574 0701 8                                          BEST_OFFSET = .OFFSET;
575 0702 7                                      END;
576 0703 6                                  END;
577 0704 5                              END;
578 0705 4                          END;
579 0706 3                      END;
580 0707 3
581 0708 3      ! Link to the next SATPTR and loop.
582 0709 3      SATPTR = .SATPTR[SAT$L_FLINK];
583 0710 3      END;
584 0711 3
585 0712 2      ! Return the best candidate.
586 0713 2
587 0714 2      IF .BEST_SATPTR NEQ 0
588 0715 2      THEN
589 0716 2          .BYTE_OFFSET = .BEST_OFFSET;
590 0717 2
```


: 591
: 592

0718 2
0719 1

RETURN .BEST_SATPTR;
END;

				00FC 00000		.ENTRY	DBG\$SEARCH PROG_SAT, Save R2,R3,R4,R5,R6,R7	0604
				55 D4 00002		CLRL	BEST_SATPTR	0649
	56	3B9AC9FF		8F D0 00004		MOVL	#999999999, BEST_OFFSET	0650
	53	04		AC D0 0000B		MOVL	BASE_NODE, SATPTR	0651
				58 13 0000F	1\$:	BEQL	5\$	0652
	04	A3	08	AC D1 00011		CMPL	ADDRESS, 4(SATPTR)	0659
				51 1F 00016		BLSSU	5\$	
	08	A3	08	AC D1 00018		CMPL	ADDRESS, 8(SATPTR)	0667
				45 1A 0001D		BGTRU	4\$	
	52	0C		A3 D0 0001F		MOVL	12(SATPTR), MODRSTPTR	0670
	3D	28		A2 E9 00023		BLBC	40(MODRSTPTR), 4\$	0671
				7E D4 00027		CLRL	-(SP)	0674
				51 D4 00029		CLRL	R1	0677
			29	A2 95 0002B		TSTB	41(MODRSTPTR)	
				02 12 0002E		BNEQ	2\$	
				51 D6 00030		INCL	R1	
57	28	A2	01	04 EF 00032	2\$:	EXTZV	#4, #1, 40(MODRSTPTR), R7	0678
		7E	51	57 C9 00038		BISL3	R7, R1, -(SP)	
				08 AC DD 0003C		PUSHL	ADDRESS	0676
			18	A2 DD 0003F		PUSHL	24(MODRSTPTR)	0675
	83	AF		04 FB 00042		CALLS	#4, DBG\$SEARCH_BIN_SAT	
				50 D5 00046		TSTL	TEMP_SATPTR	0684
				1A 13 00048		BEQL	4\$	
	54	08	04	A0 C3 0004A		SUBL3	4(TEMP_SATPTR), ADDRESS, OFFSET	0687
		56		54 D1 00050		CMPL	OFFSET, BEST_OFFSET	0688
				09 19 00053		BLSS	3\$	
				0D 12 00055		BNEQ	4\$	0694
	08	A5	08	A0 D1 00057		CMPL	8(TEMP_SATPTR), 8(BEST_SATPTR)	0697
				06 18 0005C		BGEQ	4\$	
		55		50 D0 0005E	3\$:	MOVL	TEMP_SATPTR, BEST_SATPTR	0700
		56		54 D0 00061		MOVL	OFFSET, BEST_OFFSET	0701
		53		63 D0 00064	4\$:	MOVL	(SATPTR), SATPTR	0710
				A6 11 00067		BRB	1\$	0652
				55 D5 00069	5\$:	TSTL	BEST_SATPTR	0715
				04 13 0006B		BEQL	6\$	
	0C	BC		56 D0 0006D		MOVL	BEST_OFFSET, @BYTE_OFFSET	0717
		50		55 D0 00071	6\$:	MOVL	BEST_SATPTR, R0	0718
				04 00074		RET		0719

: Routine Size: 117 bytes, Routine Base: DBG\$CODE + 01CE


```
594 0720 1 GLOBAL ROUTINE DBG$SEARCH_BIN_SAT_REC (BASE_NODE, ADDRESS,  
595 0721 1 MACRO_OR_SHARE_FLAG, PSECT_FLAG)=  
596 0722 1  
597 0723 1 FUNCTION  
598 0724 1 This routine searches the binary SAT tree starting at BASE_NODE  
599 0725 1 and returns a pointer to a SAT entry containing ADDRESS (or zero  
600 0726 1 if no SAT entry was found).  
601 0727 1  
602 0728 1 This routine is called to search a module-level SAT. It is called  
603 0729 1 by DBG$SEARCH_PROG_SAT.  
604 0730 1  
605 0731 1 The tree is organized so that at any point in the search, if we  
606 0732 1 are at node NODE, then: If the desired address is less than the  
607 0733 1 range in NODE, then follow the RIGHT_SON link. If the desired  
608 0734 1 address is greater than the range in NODE, then follow the  
609 0735 1 LEFT_SON link. If we try to follow a link and the link is zero  
610 0736 1 then we failed to find a matching SAT entry.  
611 0737 1 If the address is in the range, then of course we are done.  
612 0738 1  
613 0739 1 This routine is designed so that it will work on the 'old' linear  
614 0740 1 list data structure (this just being the extreme case of an  
615 0741 1 unbalanced tree). However, for now we are still calling the  
616 0742 1 old non-recursive search_sat routines. This one is not yet  
617 0743 1 being used.  
618 0744 1  
619 0745 1 INPUTS  
620 0746 1 BASE_NODE - The base of the SAT tree  
621 0747 1 ADDRESS - The desired address.  
622 0748 1 MACRO_OR_SHARE_FLAG- True for language MACRO and also true  
623 0749 1 for static address chains that come from shareable  
624 0750 1 images.  
625 0751 1 PSECT_FLAG- Says whether it is OK to symbolize to a PSECT.  
626 0752 1  
627 0753 1 OUTPUTS  
628 0754 1 A pointer to the found SAT entry is returned.  
629 0755 1 Zero is returned if no SAT entry was found.  
630 0756 1  
631 0757 2 BEGIN  
632 0758 2 MAP  
633 0759 2 BASE_NODE: REF SAT$ENTRY;  
634 0760 2 LOCAL  
635 0761 2 SATPTR: REF SAT$ENTRY;  
636 0762 2  
637 0763 2 ! Bottom level of the recursion.  
638 0764 2  
639 0765 2 IF .BASE_NODE EQL 0  
640 0766 2 THEN  
641 0767 2 RETURN 0;  
642 0768 2  
643 0769 2 ! If the address we are looking for is before the current SAT entry  
644 0770 2 then follow the left son.  
645 0771 2  
646 0772 2 IF .ADDRESS LSSA .BASE_NODE[SAT$L_START]  
647 0773 2 THEN  
648 0774 2 RETURN DBG$SEARCH_BIN_SAT_REC(.BASE_NODE[SAT$L_LEFTSON], .ADDRESS,  
649 0775 2 .MACRO_OR_SHARE_FLAG, .PSECT_FLAG);  
650 0776 2
```



```

: 651      0777      2      ! If the address we are looking for falls after the start of the
: 652      0778      2      ! current SAT entry then follow the right son. If this search
: 653      0779      2      ! yields a matching SAT then return it.
: 654      0780      2
: 655      0781      2      SATPTR = DBG$SEARCH_BIN_SAT_REC(.BASE_NODE[SAT$L_RIGHTSON], .ADDRESS,
: 656      0782      2      .MACRO_OR_SHARE_FLAG, .PSECT_FLAG);
: 657      0783      2
: 658      0784      2      IF .SATPTR NEQ 0
: 659      0785      2      THEN
: 660      0786      2          RETURN .SATPTR;
: 661      0787      2
: 662      0788      2      ! Don't symbolize to PSECTs unless PSECT_FLAG is set.
: 663      0789      2
: 664      0790      2      IF (.PSECT_FLAG) OR
: 665      0791      2      (NOT IS_SYMBOL_PSECT (.BASE_NODE[SAT$L_RSTPTR]))
: 666      0792      2      THEN
: 667      0793      2          BEGIN
: 668      0794      2              ! Check for the address being in the range of the SAT entry.
: 669      0795      2              ! But, in language MACRO, choose this symbolization - don't
: 670      0796      2              ! do the check for the address being in the correct range.
: 671      0797      2              ! Also, don't do this check for shareable image modules.
: 672      0798      2              ! The reason is that in these cases we do not get the right
: 673      0799      2              ! length information for the static item.
: 674      0800      2
: 675      0801      2              IF .MACRO OR SHARE_FLAG OR
: 676      0802      2              (.ADDRESS [EQA .BASE_NODE[SAT$L_END]])
: 677      0803      2              THEN
: 678      0804      2                  RETURN .BASE_NODE;
: 679      0805      2              END;
: 680      0806      2
: 681      0807      2      ! Failed to find a match. Return 0.
: 682      0808      2
: 683      0809      2      RETURN 0;
: 684      0810      1      END;
```

			0004	00000	.ENTRY	DBG\$SEARCH_BIN_SAT_REC, Save R2	: 0720
	52	04	AC	D0 00002	MOVL	BASE_NODE, R2	: 0765
			45	13 00006	BEQL	4\$	
04	A2	08	AC	D1 00008	CMPL	ADDRESS, 4(R2)	: 0772
			0F	1E 0000D	BGEQU	1\$	
	7E	0C	AC	7D 0000F	MOVQ	MACRO_OR_SHARE_FLAG, -(SP)	: 0775
		08	AC	DD 00013	PUSHL	ADDRESS	: 0774
		10	A2	DD 00016	PUSHL	16(R2)	
E3	AF		04	FB 00019	CALLS	#4, DBG\$SEARCH_BIN_SAT_REC	
				04 0001D	RET		
	7E	0C	AC	7D 0001E 1\$:	MOVQ	MACRO_OR_SHARE_FLAG, -(SP)	: 0782
		08	AC	DD 00022	PUSHL	ADDRESS	: 0781
			62	DD 00025	PUSHL	(R2)	
D5	AF		04	FB 00027	CALLS	#4, DBG\$SEARCH_BIN_SAT_REC	
			50	D5 0002B	TSTL	SATPTR	: 0783
			20	12 0002D	BNEQ	5\$	
	0B	10	AC	E8 0002F	BLBS	PSECT_FLAG, 2\$	: 0789
		0C	A2	DD 00033	PUSHL	12(R2)	: 0790

DBGSYMBLZ
V04-000

E 4
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 21
(8)

0000V	CF		01	FB	00036	CALLS	#1, IS_SYMBOL_PSECT	:
	OF		50	EB	0003B	BLBS	R0, 4\$	:
	07	0C	AC	EB	0003E	BLBS	MACRO_OR_SHARE_FLAG, 3\$	0801
08	A2	08	AC	D1	00042	CMPL	ADDRESS, -8(R2)	0802
			04	1A	00047	BGTRU	4\$	:
	50		52	D0	00049	MOVL	R2, R0	0804
				04	0004C	RET		:
			50	D4	0004D	CLRL	R0	0810
			04	0004F	5\$:	RET		:

; Routine Size: 80 bytes, Routine Base: DBG\$CODE + 0243


```

: 686      0811 1 GLOBAL ROUTINE DBG$SEARCH_PROG_SAT_REC (BASE_NODE, ADDRESS, BYTE_OFFSET) =
: 687      0812 1
: 688      0813 1 FUNCTION
: 689      0814 1     This routine searches the binary program-level
: 690      0815 1     SAT tree starting at BASE_NODE
: 691      0816 1     and returns a pointer to the SAT entry which contains ADDRESS
: 692      0817 1     and whose start address is closest to ADDRESS. (or zero
: 693      0818 1     if no SAT entry was found).
: 694      0819 1
: 695      0820 1     The tree is organized so that at any point in the search, if we
: 696      0821 1     are at node NODE, then: If the desired address is less than the
: 697      0822 1     range in NODE, then follow the LEFT_SON link. If the desired
: 698      0823 1     address is greater than the range in NODE, then follow the
: 699      0824 1     RIGHT_SON link. If we try to follow a link and the link is zero
: 700      0825 1     then we terminate the search.
: 701      0826 1
: 702      0827 1     Once a candidate program-level SAT is found, this routine calls
: 703      0828 1     SEARCH_BIN_SAT to search the module-level SAT.
: 704      0829 1
: 705      0830 1     This routine is designed so that it will work on the "old" linear
: 706      0831 1     list data structure (this just being the extreme case of an
: 707      0832 1     unbalanced tree). However, for now we are still calling the
: 708      0833 1     old non-recursive search_sat routines. This one is not yet
: 709      0834 1     being used.
: 710      0835 1
: 711      0836 1     The routine is intended to be a replacement for the DBG$SEARCH_SAT
: 712      0837 1     routine.
: 713      0838 1
: 714      0839 1 INPUTS
: 715      0840 1     BASE_NODE - The base of the SAT tree
: 716      0841 1     ADDRESS - The desired address.
: 717      0842 1     BYTE_OFFSET - Return the byte offset associated with the symbolization
: 718      0843 1     in this address.
: 719      0844 1
: 720      0845 1 OUTPUTS
: 721      0846 1     A pointer to the found SAT entry is returned.
: 722      0847 1     Zero is returned if no SAT entry was found.
: 723      0848 1
: 724      0849 2 BEGIN
: 725      0850 2 MAP
: 726      0851 2     BASE_NODE: REF SAT$ENTRY;
: 727      0852 2
: 728      0853 2 LOCAL
: 729      0854 2     OFFSET,
: 730      0855 2     OFFSET_1,
: 731      0856 2     MODRSTPTR: REF RST$ENTRY,
: 732      0857 2     SATPTR: REF SAT$ENTRY,
: 733      0858 2     SATPTR_1: REF SAT$ENTRY;
: 734      0859 2
: 735      0860 2 ! Bottom level of the recursion.
: 736      0861 2 !
: 737      0862 2 IF .BASE_NODE EQL 0
: 738      0863 2 THEN
: 739      0864 2     RETURN 0;
: 740      0865 2
: 741      0866 2 ! If the address we are looking for is before the current SAT entry
: 742      0867 2 ! then follow the left son.
```



```
!
IF .ADDRESS LSSA .BASE_NODE[SAT$L_START]
THEN
    RETURN DBG$SEARCH_PROG_SAT_REC(.BASE_NODE[SAT$L_LEFTSON], .ADDRESS,
                                   .BYTE_OFFSET);

! If the address we are looking for falls after the start of the
! current SAT entry then follow the right son.
SATPTR = DBG$SEARCH_PROG_SAT_REC(.BASE_NODE[SAT$L_RIGHTSON], .ADDRESS, OFFSET);

! If the address we are looking for also lies within the current
! program-level SAT then search the module-level SAT.
IF .ADDRESS LEQA .BASE_NODE[SAT$L_END]
THEN
    BEGIN
        MODRSTPTR = .BASE_NODE[SAT$L_RSTPTR];
        IF .MODRSTPTR[RST$V_MODSET]
        THEN
            SATPTR_1 = DBG$SEARCH_BIN_SAT_REC (
                .MODRSTPTR[RST$L_SAT_PTR],
                .ADDRESS,
                (.MODRSTPTR[RST$B_LANGUAGE] EQL DBG$K_MACRO) OR
                .MODRSTPTR[RST$V_SHARE_IMAGE],
                FALSE)

        ELSE
            SATPTR_1 = 0;
        IF .SATPTR_1 NEQ 0
        THEN
            OFFSET_1 = .ADDRESS - .SATPTR_1[SAT$L_START];
        END;

! If both of our candidate SATPTR's are zero then we have failed
! to find a match so return zero.
IF (.SATPTR EQL 0) AND (.SATPTR_1 EQL 0)
THEN
    RETURN 0;

! If only one candidate is zero then return the other one.
IF .SATPTR EQL 0
THEN
    BEGIN
        .BYTE_OFFSET = .OFFSET_1;
        RETURN .SATPTR_1;
    END;
IF .SATPTR_1 EQL 0
THEN
    BEGIN
        .BYTE_OFFSET = .OFFSET;
        RETURN .SATPTR;
    END;

! Both are non-zero. Return the one with the smaller offset.
!
```



```

: 800      0925      2
: 801      0926
: 802      0927
: 803      0928
: 804      0929
: 805      0930
: 806      0931
: 807      0932
: 808      0933
: 809      0934
: 810      0935
: 811      0936
: 812      0937
: 813      0938
: 814      0939
: 815      0940
: 816      0941
: 817      0942
: 818      0943
: 819      0944
: 820      0945
: 821      0946
: 822      0947
: 823      0948
: 824      0949
: 825      0950
: 826      0951      1

IF .OFFSET LSS .OFFSET_1
THEN
    BEGIN
        .BYTE_OFFSET = .OFFSET;
        RETURN .SATPTR;
    END;
IF .OFFSET GTR .OFFSET_1
THEN
    BEGIN
        .BYTE_OFFSET = .OFFSET_1;
        RETURN .SATPTR_1;
    END;
! The offsets are equal. Return the one with the smaller end address.
IF .SATPTR[SAT$L_END] LSS .SATPTR_1[SAT$L_END]
THEN
    BEGIN
        .BYTE_OFFSET = .OFFSET;
        RETURN .SATPTR;
    END
ELSE
    BEGIN
        .BYTE_OFFSET = .OFFSET_1;
        RETURN .SATPTR_1;
    END;
END;
```

			001C	00000	.ENTRY	DBG\$SEARCH_PROG_SAT_REC, Save R2,R3,R4	0811	
	5E		04	C2	00002	SUBL2	#4, SP	
	52	04	AC	D0	00005	MOVL	BASE_NODE, R2	0862
			68	13	00009	BEQL	6\$	
	53	08	AC	D0	0000B	MOVL	ADDRESS, R3	0869
04	A2		53	D1	0000F	CMPL	R3, 4(R2)	
			0D	1E	00013	BGEQU	1\$	
		0C	AC	DD	00015	PUSHL	BYTE_OFFSET	0872
			53	DD	00018	PUSHL	R3	0871
		10	A2	DD	0001A	PUSHL	16(R2)	
DF	AF		03	FB	0001D	CALLS	#3, DBG\$SEARCH_PROG_SAT_REC	
			04	00021	RET			
		4008	8F	BB	00022	PUSHR	#^M<R3,SP>	0877
			62	DD	00026	PUSHL	(R2)	
D4	AF		03	FB	00028	CALLS	#3, DBG\$SEARCH_PROG_SAT_REC	
	54		50	D0	0002C	MOVL	R0, SATPTR	
08	A2		53	D1	0002F	CMPL	R3, 8(R2)	0882
			34	1A	00033	BGTRU	5\$	
	50	0C	A2	D0	00035	MOVL	12(R2), MODRSTPTR	0885
	21	28	A0	E9	00039	BLBC	40(MODRSTPTR), 3\$	0886
			7E	D4	0003D	CLRL	-(SP)	0888
			51	D4	0003F	CLRL	R1	0891
		29	A0	95	00041	TSTB	41(MODRSTPTR)	
			02	12	00044	BNEQ	2\$	
			51	D6	00046	INCL	R1	

DBGSYMBLZ
V04-000

1 4
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 25
(9)

52	28	A0	01	04	EF	00048	2\$:	EXTZV	#4, #1, 40(MODRSTPTR), R2	0892
		7E	51	52	C9	0004E		BISL3	R2, R1, -(SP)	
				53	DD	00052		PUSHL	R3	0890
				18	A0	DD	00054	PUSHL	24(MODRSTPTR)	0889
		FF54	CF	04	FB	00057		CALLS	#4, DBG\$SEARCH_BIN_SAT_REC	
				02	11	0005C		BRB	4\$	0888
				50	D4	0005E	3\$:	CLRL	SATPTR_1	0895
				50	D5	00060	4\$:	TSTL	SATPTR_1	0896
				05	13	00062		BEQL	5\$	
	51		53	04	A0	C3	00064	SUBL3	4(SATPTR_1), R3, OFFSET_1	0898
				53	D4	00069	5\$:	CLRL	R3	0904
				54	D5	0006B		TSTL	SATPTR	
				06	12	0006D		BNEQ	7\$	
				53	D6	0006F		INCL	R3	
				50	D5	00071		TSTL	SATPTR_1	
				22	13	00073	6\$:	BEQL	10\$	
		1A		53	E8	00075	7\$:	BLBS	R3, 9\$	0910
				50	D5	00078		TSTL	SATPTR_1	0916
				0E	13	0007A		BEQL	8\$	
			51	6E	D1	0007C		CMPL	OFFSET, OFFSET_1	0925
				09	19	0007F		BLSS	8\$	
				0F	14	00081		BGTR	9\$	0931
	08	A0	08	A4	D1	00083		CMPL	8(SATPTR), 8(SATPTR_1)	0940
				08	18	00088		BGEQ	9\$	
	0C	BC		6E	D0	0008A	8\$:	MOVL	OFFSET, @BYTE_OFFSET	0943
		50		54	D0	0008E		MOVL	SATPTR, R0	0947
					04	00091		RET		
	0C	BC		51	D0	00092	9\$:	MOVL	OFFSET_1, @BYTE_OFFSET	0948
					04	00096		RET		0947
				50	D4	00097	10\$:	CLRL	R0	0951
					04	00099		RET		

; Routine Size: 154 bytes, Routine Base: DBG\$CODE + 0293


```

828 0952 1 GLOBAL ROUTINE DBG$SEARCH_GLOBAL(ADDR, P_SYMID, P_BIT_OFFSET, PRINT_FLAG) =
829 0953 1
830 0954 1 FUNCTION
831 0955 1     This routine searches the GLOBAL symbol chain in an attempt to
832 0956 1     symbolize the address.
833 0957 1
834 0958 1     NOTE: This routine is searching the whole Global symbol chain
835 0959 1     if the proper global symbol is not found. If we can be sure
836 0960 1     the global symbol chain is ordered. We can stop the searching
837 0961 1     as soon as we have passed the address range.
838 0962 1
839 0963 1 INPUTS
840 0964 1     ADDR                - Address Descriptor.
841 0965 1
842 0966 1     P_SYMID             - The address of a longword location to receive the
843 0967 1                       unique symbol RST pointer.
844 0968 1
845 0969 1     P_BIT_OFFSET        - The bit offset off of P_SYMID.
846 0970 1
847 0971 1     PRINT_FLAG          - If false, no output will be done. If true, then
848 0972 1                       the symbolization of the virtual address will be
849 0973 1                       output to the terminal.
850 0974 1
851 0975 1 OUTPUTS
852 0976 1     If successful, the routine prints the symbolization of the address
853 0977 1     (if the print flag was set). P_SYMID contains the RST pointer of
854 0978 1     the symbol that matched. P_BIT_OFFSET contains the bit offset.
855 0979 1     The routine returns TRUE. If the address is not found in the
856 0980 1     global symbol chain, then P_SYMID is zero and P_BIT_OFFSET is set
857 0981 1     to the bit offset field from ADDR.
858 0982 1
859 0983 1
860 0984 2 BEGIN
861 0985 2
862 0986 2 MAP
863 0987 2     ADDR: REF DBG$ADDRESS_DESC;
864 0988 2
865 0989 2 BIND
866 0990 2     SYMID                = .P_SYMID: REF RST$ENTRY,
867 0991 2     BIT_OFFSET          = .P_BIT_OFFSET;
868 0992 2
869 0993 2 LOCAL
870 0994 2     ADDRESS,                : Byte Address
871 0995 2     BEST_MATCH_VALUE,      : The best match value we found
872 0996 2     BYTE_OFFSET,           : Byte offset
873 0997 2     DSTPTR: REF DST$RECORD, : Pointer to DST record in GST
874 0998 2     GSTPTR: REF RST$ENTRY,  : GST RST pointer
875 0999 2     GST_VALUE,             : GST value
876 1000 2     NAME: REF VECTOR[,BYTE]; : Pointer to counted ASCII string
877 1001 2
878 1002 2
879 1003 2     ADDRESS = .ADDR[DBG$ ADDRESS BYTE ADDR];
880 1004 2     BIT_OFFSET = .ADDR[DBG$ ADDRESS_BIT_OFFSET];
881 1005 2     BYTE_OFFSET = 0;
882 1006 2     SYMID = 0;
883 1007 2     GSTPTR = .RST$START_ADDR[RST$L_SYMCHNPTR];
884 1008 2 WHILE TRUE DO
```



```
.. 885 1009 3
   886 1010 3
   887 1011 3
   888 1012 3
   889 1013 3
   890 1014 3
   891 1015 3
   892 1016 3
   893 1017 3
   894 1018 3
   895 1019 3
   896 1020 3
   897 1021 3
   898 1022 3
   899 1023 3
  900 1024 4
  901 1025 4
  902 1026 4
  903 1027 3
  904 1028 3
  905 1029 3
  906 1030 3
  907 1031 3
  908 1032 3
  909 1033 3
  910 1034 4
  911 1035 4
  912 1036 4
  913 1037 4
  914 1038 4
  915 1039 4
  916 1040 4
  917 1041 4
  918 1042 4
  919 1043 4
  920 1044 4
  921 1045 4
  922 1046 4
  923 1047 4
  924 1048 4
  925 1049 5
  926 1050 5
  927 1051 5
  928 1052 5
  929 1053 5
  930 1054 4
  931 1055 4
  932 1056 3
  933 1057 3
  934 1058 3
  935 1059 3
  936 1060 3
  937 1061 3
  938 1062 2
  939 1063 2
  940 1064 2
  941 1065 2
```

```
BEGIN

! Check to see if we have no more Global Symbols.
! IF .GSTPTR EQL 0 THEN EXITLOOP;
$ABORT ON CONTROL Y;
GST_VALUE = GSTADDRESS(.GSTPTR);

! First check for an exact match because then
! we can abandon any further looking.
IF .ADDRESS EQLA .GST_VALUE
THEN
  BEGIN
    SYMID = .GSTPTR;
    EXITLOOP;
  END;

! Inexact matches are still better than nothing.
IF .ADDRESS GTRA .GST_VALUE
THEN
  BEGIN

    ! A match. See if we already have one. If we do not have
    ! one, Any one is better than none.
    IF .SYMID EQL 0
    THEN
      SYMID = .GSTPTR

    ! Take the new one only if this symbol
    ! is closer than the previous best one.
    ELSE
      BEGIN
        BEST_MATCH_VALUE = GSTADDRESS(.SYMID);
        IF .BEST_MATCH_VALUE LSSA .GST_VALUE
        THEN
          SYMID = .GSTPTR;
        END;
      END;

    END;

    ! Go back and process the next record.
    ! GSTPTR = .GSTPTR[RST$L_SYMCHNPTR];
    END;

! If we didn't find any possible match,
```



```

: 942      1066  2      ! return failure status.
: 943      1067  2      !
: 944      1068  2      IF .SYMID EQL 0
: 945      1069  2      THEN
: 946      1070  2      RETURN 0;
: 947      1071  2
: 948      1072  2      BEST_MATCH_VALUE = GSTADDRESS(.SYMID);
: 949      1073  2      BYTE_OFFSET = .ADDRESS - .BEST_MATCH_VALUE;
: 950      1074  2      IF .BYTE_OFFSET EQL .ADDRESS THEN RETURN 0;
: 951      1075  2
: 952      1076  2
: 953      1077  2      ! If the BEST_MATCH_VALUE is not in the range cover by the MODULE and
: 954      1078  2      ! if the offset is too large, don't take it.
: 955      1079  2      !
: 956      1080  2      IF .BYTE_OFFSET NEQ 0
: 957      1081  2      THEN
: 958      1082  3      BEGIN
: 959      1083  4      IF NOT (.ADDRESS GEQ .OWN_MODULE_BEGIN AND
: 960      1084  4      .ADDRESS LEQ .OWN_MODULE_END)
: 961      1085  3      THEN
: 962      1086  4      BEGIN
: 963      1087  4      IF .BYTE_OFFSET GTR %X'1000'
: 964      1088  4      THEN
: 965      1089  5      BEGIN
: 966      1090  5      SYMID = 0;
: 967      1091  5      BYTE_OFFSET = 0;
: 968      1092  5      RETURN 0;
: 969      1093  4      END;
: 970      1094  4
: 971      1095  3      END;
: 972      1096  3
: 973      1097  2      END;
: 974      1098  2
: 975      1099  2      IF .PRINT_FLAG
: 976      1100  2      THEN
: 977      1101  3      BEGIN
: 978      1102  3      DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
: 979      1103  3      PRINT BIT_OFFSET(BIT_OFFSET);
: 980      1104  3      DBG$PRINT (UPLIT BYTE(%ASCIC ': (global)'));
: 981      1105  3      DBG$NEWLINE();
: 982      1106  3      DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
: 983      1107  3      DSTPTR = .SYMID[RST$L_DSTPTR];
: 984      1108  3      NAME = DBG$GET_DST_NAME(.DSTPTR);
: 985      1109  3      DBG$PRINT (UPLIT BYTE(%ASCIC '!AC'), .NAME);
: 986      1110  3      IF .BYTE_OFFSET NEQ 0
: 987      1111  3      THEN
: 988      1112  3      DBG$PRINT_OFFSET(.BYTE_OFFSET);
: 989      1113  3      PRINT BIT_OFFSET(BIT_OFFSET);
: 990      1114  3      DBG$NEWLINE();
: 991      1115  2      END;
: 992      1116  2
: 993      1117  2      BIT_OFFSET = .BIT_OFFSET + .BYTE_OFFSET * 8;
: 994      1118  2      RETURN TRUE;
: 995      1119  1      END;
```



```

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
4C 58 21 20 73 73 65 72 64 64 61 0B 00032 P.AAI: .ASCII <11>\address !XL\
3C 01 0003E P.AAJ: .ASCII <1>\<\
21 03 00040 P.AAK: .ASCII <3>\!UL\
2C 07 00044 P.AAL: .ASCII <7>\, 0, 0>\
29 6C 61 62 6F 6C 67 28 20 3A 0A 0004C P.AAM: .ASCII <10>\: (global)\
43 41 21 03 00057 P.AAN: .ASCII <3>\!AC\
3C 01 0005B P.AAO: .ASCII <1>\<\
21 03 0005D P.AAP: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 00061 P.AAQ: .ASCII <7>\, 0, 0>\

.EXTRN DBG$GV_CONTROL

.PSECT DBG$CODE,NOWRT, SHR, PIC,0
.OFFC 00000
.ENTRY DBG$SEARCH GLOBAL, Save R2,R3,R4,R5,R6,R7,- 0952
R8,R9,R10,R11
MOVAB GSTADDRESS, R11
MOVAB DBG$NEWLINE, R10
MOVAB DBG$PRINT, R9
MOVAB P.AAI, R8
MOVL P_SYMID, R5 0990
MOVL ADDR, R0 1003
MOVL (R0), ADDRESS
MOVL 4(R0), @P_BIT_OFFSET 1004
CLRL BYTE_OFFSET 1005
CLRL (R5) 1006
MOVL RST$START_ADDR, R0 1007
MOVL 8(R0), GSTPTR
BEQL 6$ 1014
BBC #1, DBG$GV_CONTROL+1, 2$
PUSHL #164072
CALLS #1, LIB$SIGNAL
PUSHL GSTPTR 1016
CALLS #1, GSTADDRESS
MOVL R0, GST_VALUE
CMPL ADDRESS, GST_VALUE 1022
BNEQ 3$
MOVL GSTPTR, (R5) 1025
BRB 6$ 1024
BLEQU 5$ 1032
TSTL (R5) 1040
BEQL 4$ 1050
PUSHL (R5)
CALLS #1, GSTADDRESS
MOVL R0, BEST_MATCH_VALUE
CMPL BEST_MATCH_VALUE, GST_VALUE 1051
BGEQU 5$
MOVL GSTPTR, (R5) 1053
MOVL 8(GSTPTR), GSTPTR 1061
BRB 1$ 1008
TSTL (R5) 1068
BEQL 8$
PUSHL (R5) 1072
CALLS #1, GSTADDRESS
MOVL R0, BEST_MATCH_VALUE

```


54	53	57	C3	0008C	SUBL3	BEST_MATCH_VALUE, ADDRESS, BYTE_OFFSET	1073
	53	54	D1	00090	CMPL	BYTE_OFFSET, ADDRESS	1074
		23	13	00093	BEQL	8\$	
		54	D5	00095	TSTL	BYTE_OFFSET	1080
		22	13	00097	BEQL	9\$	
00000000'	EF	53	D1	00099	CMPL	ADDRESS, OWN_MODULE_BEGIN	1083
		09	19	000A0	BLSS	7\$	
00000000'	EF	53	D1	000A2	CMPL	ADDRESS, OWN_MODULE_END	1084
		10	15	000A9	BLEQ	9\$	
00001000	8F	54	D1	000AB	CMPL	BYTE_OFFSET, #4096	1087
		07	15	000B2	BLEQ	9\$	
		65	D4	000B4	CLRL	(R5)	1090
		54	D4	000B6	CLRL	BYTE_OFFSET	1091
		008B	31	000B8	BRW	14\$	1092
	7A	10	AC	E9	000BB	9\$: BLBC	PRINT_FLAG, 13\$
			53	DD	000BF	PUSHL	ADDRESS
			58	DD	000C1	PUSHL	R8
	69		02	FB	000C3	CALLS	#2, DBG\$PRINT
	53	0C	BC	D0	000C6	MOVL	@P_BIT_OFFSET, R3
			52	D4	000CA	CLRL	R2
			53	D5	000CC	TSTL	R3
			16	13	000CE	BEQL	10\$
			52	D6	000D0	INCL	R2
		0C	A8	9F	000D2	PUSHAB	P.AAJ
	69		01	FB	000D5	CALLS	#1, DBG\$PRINT
			53	DD	000D8	PUSHL	R3
		0E	A8	9F	000DA	PUSHAB	P.AAK
	69		02	FB	000DD	CALLS	#2, DBG\$PRINT
		12	A8	9F	000E0	PUSHAB	P.AAL
	69		01	FB	000E3	CALLS	#1, DBG\$PRINT
		1A	A8	9F	000E6	10\$: PUSHAB	P.AAM
	69		01	FB	000E9	CALLS	#1, DBG\$PRINT
	6A		00	FB	000EC	CALLS	#0, DBG\$NEWLINE
			04	DD	000EF	PUSHL	#4
			01	DD	000F1	PUSHL	#1
00000000G	00		02	FB	000F3	CALLS	#2, DBG\$PRINT_CONTROL
	50		65	D0	000FA	MOVL	(R5), R0
	50	0C	A0	D0	000FD	MOVL	12(R0), DSTPTR
			50	DD	00101	PUSHL	DSTPTR
00000000G	00		01	FB	00103	CALLS	#1, DBG\$GET_DST_NAME
			50	DD	0010A	PUSHL	NAME
		25	A8	9F	0010C	PUSHAB	P.AAN
	69		02	FB	0010F	CALLS	#2, DBG\$PRINT
			54	D5	00112	TSTL	BYTE_OFFSET
			09	13	00114	BEQL	11\$
			54	DD	00116	PUSHL	BYTE_OFFSET
00000000G	00		01	FB	00118	CALLS	#1, DBG\$PRINT_OFFSET
	14		52	E9	0011F	11\$: BLBC	R2, 12\$
		29	A8	9F	00122	PUSHAB	P.AAO
	69		01	FB	00125	CALLS	#1, DBG\$PRINT
			53	DD	00128	PUSHL	R3
		2B	A8	9F	0012A	PUSHAB	P.AAP
	69		02	FB	0012D	CALLS	#2, DBG\$PRINT
		2F	A8	9F	00130	PUSHAB	P.AAQ
	69		01	FB	00133	CALLS	#1, DBG\$PRINT
	6A		00	FB	00136	12\$: CALLS	#0, DBG\$NEWLINE
	50	0C	BC	D0	00139	13\$: MOVL	@P_BIT_OFFSET, R0

DBGSYMBLZ
V04-000

B 5
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 31
(10)

0C	BC	6044	7E	0013D	MOVAQ	(R0)[BYTE_OFFSET], @P_BIT_OFFSET
	50	01	D0	00142	MOVL	#1, R0
			04	00145	RET	
		50	D4	00146	CLRL	R0
			04	00148	RET	

14\$:

: 1118
: 1119
:

; Routine Size: 329 bytes, Routine Base: DBG\$CODE + 032D


```

: 997      1120 1 GLOBAL ROUTINE DBGSDUMP_GLOBAL: NOVALUE =
: 998      1121 1
: 999      1122 1 FUNCTION
: 1000     1123 1     This routine just counts the number of entries in the GST and
: 1001     1124 1     displays the information on the terminal. It is called by the
: 1002     1125 1     DUMP GST command (which is enabled by SET DEVELOPER 0) and can
: 1003     1126 1     be used as an aid in measuring DEBUG performance.
: 1004     1127 1
: 1005     1128 1 INPUTS
: 1006     1129 1     none.
: 1007     1130 1
: 1008     1131 1 OUTPUTS
: 1009     1132 1     Displays size of GST.
: 1010     1133 1
: 1011     1134 2 BEGIN
: 1012     1135 2 LOCAL
: 1013     1136 2     COUNT,
: 1014     1137 2     GSTPTR: REF RST$ENTRY;
: 1015     1138 2
: 1016     1139 2     ! Loop through the GST, counting the number of entries.
: 1017     1140 2     !
: 1018     1141 2     GSTPTR = .RST$START_ADDR[RST$_SYMCHNPTR];
: 1019     1142 2     COUNT = 0;
: 1020     1143 2     WHILE .GSTPTR NEQ 0 DO
: 1021     1144 2     BEGIN
: 1022     1145 2         COUNT = .COUNT + 1;
: 1023     1146 2         GSTPTR = .GSTPTR[RST$_SYMCHNPTR];
: 1024     1147 2     END;
: 1025     1148 2
: 1026     1149 2     ! Display the information.
: 1027     1150 2     !
: 1028     1151 2     DBG$PRINT(UPLOT BYTE (%ASCII 'There are !UL entries in the GST'), .COUNT);
: 1029     1152 2     DBG$NEWLINE();
: 1030     1153 1     END;

```

```

: 20 4C 55 21 20 65 72 61 20 65 72 65 68 54 20 00069 P.AAR: .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
: 20 65 68 74 20 6E 69 20 73 65 69 72 74 6E 65 00078 .ASCII \ There are !UL entries in the GST\
: 54 53 47 00087

```

```

: 50 00000000G 00 D0 00000 0000 .ENTRY DBGSDUMP_GLOBAL, Save nothing : 1120
: 50 08 A0 D0 00009 00002 MOVL RST$START_ADDR, R0 : 1141
: 51 D4 0000D 00000 MOVL 8(R0), GSTPTR : 1142
: 50 D5 0000F 1$: CLRL COUNT : 1143
: 08 13 00011 TSTL GSTPTR : 1145
: 51 D6 00013 BEQL 2$ : 1146
: 50 08 A0 D0 00015 INCL COUNT : 1143
: F4 11 00019 BRB 1$ : 1151
: 51 DD 0001B 2$: PUSHL COUNT

```


DBGSYMBLZ
V04-000

D 5
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 33
(11)

00000000G 00 00000000' EF 9F 0001D
00000000G 00 02 FB 00023
00000000G 00 00 FB 0002A
04 00031

PUSHAB P,AAR
CALLS #2, DBG\$PRINT
CALLS #0, DBG\$NEWLINE
RET

:
:
: 1152
: 1153

; Routine Size: 50 bytes. Routine Base: DBG\$CODE + 0476


```
1032 1154 1 GLOBAL ROUTINE DBG$SEARCH_SAT(ADDR, P_SYMID, P_BIT_OFFSET, PRINT_FLAG) =
1033 1155 1
1034 1156 1 FUNCTION
1035 1157 1     This routine searches both the program level and the module level SAT in an
1036 1158 1     attempt to symbolize the input address.  If the address is found to be
1037 1159 1     an instruction address, the routine will attempt to symbolize it as a line
1038 1160 1     number and byte offset from the start of the line.
1039 1161 1
1040 1162 1     If the address turns out to be a data address, this routine will see if it
1041 1163 1     corresponds to any static symbol.  If so, the symbol most immediately preceding
1042 1164 1     the address (or an exact match) and a positive offset (or 0) will be returned,
1043 1165 1     along with the value TRUE.
1044 1166 1     Symbolization will not be done to array elements or record components; only the
1045 1167 1     outer level static data item will be returned as the symbol.
1046 1168 1
1047 1169 1     If no suitable symbolization can be found, then the routine returns false, and
1048 1170 1     symid is 0.
1049 1171 1
1050 1172 1 INPUTS:
1051 1173 1     ADDR          - Address descriptor.
1052 1174 1
1053 1175 1     P_SYMID       - The address of a longword location to receive the "symbol
1054 1176 1                     identifier".
1055 1177 1
1056 1178 1     P_BIT_OFFSET  - The bit offset off of P_SYMID.
1057 1179 1
1058 1180 1     PRINT_FLAG   - If false, no output will be done.  If true, then all symbols that
1059 1181 1                     match the virtual address will be output to the terminal.  (There
1060 1182 1                     may be more than one match, such as would occur in FORTRAN common.)
1061 1183 1
1062 1184 1     5th Parameter - If this routine is called from DBG$PC_TO_SYMID, this
1063 1185 1                     parameter is set to TRUE.  If this is the case, the
1064 1186 1                     line symid will not be created.
1065 1187 1
1066 1188 1 OUTPUTS:
1067 1189 1     SYMID         - A symbol identifier which uniquely identifies the symbol which best
1068 1190 1                     symbolizes the input address.
1069 1191 1
1070 1192 1     If the PRINT_FLAG has a value of TRUE, then output of all matched symbols
1071 1193 1     to the terminal will occur.
1072 1194 1
1073 1195 1
1074 1196 2 BEGIN
1075 1197 2
1076 1198 2 MAP
1077 1199 2     ADDR: REF DBG$ADDRESS_DESC;          ! Pointer to Address Desc.
1078 1200 2
1079 1201 2 BUILTIN
1080 1202 2     ACTUALCOUNT;
1081 1203 2
1082 1204 2 BIND
1083 1205 2     SYMID          = .P_SYMID: REF RST$ENTRY, ! Best symbol identifier.
1084 1206 2     BIT_OFFSET     = .P_BIT_OFFSET;          ! Bit offset.
1085 1207 2
1086 1208 2 LOCAL
1087 1209 2     ADDRESS,        ! Byte address from Addr. Desc.
1088 1210 2     BYTE_OFFSET,    ! Byte offset from RSTPTR
```



```
: 1089      1211      2      MODRSTPTR:      REF RST$ENTRY,      ! The module RST ptr.
: 1090      1212      2      MODULE_FOUND,      ! Module found/not found flag.
: 1091      1213      2      PROG_SATPTR:      REF SAT$ENTRY,      ! Program level SAT ptr.
: 1092      1214      2      RSTPTR:      REF RST$ENTRY,      ! The symbol's RST ptr.
: 1093      1215      2      SYMBOL_NAME:      REF BLOCK[, BYTE],      ! Ptr to counted string.
: 1094      1216      2      TMP_BYTE_OFFSET;      ! Tmp offset for calc closest
: 1095      1217      2      symbol.
: 1096      1218      2
: 1097      1219      2
: 1098      1220      2      ! Initialize.
: 1099      1221      2
: 1100      1222      2      ADDRESS = .ADDR[DBG$L_ADDRESS_BYTE_ADDR];
: 1101      1223      2      BIT_OFFSET = .ADDR[DBG$L_ADDRESS_BIT_OFFSET];
: 1102      1224      2      BYTE_OFFSET = -1;
: 1103      1225      2      SYMID = 0;
: 1104      1226      2      MODULE_FOUND = FALSE;
: 1105      1227      2      OWN_MODULE_BEGIN = 0;
: 1106      1228      2      OWN_MODULE_END = 0;
: 1107      1229      2
: 1108      1230      2
: 1109      1231      2      ! Search through the program level static address table, looking
: 1110      1232      2      ! for a module that covers the address.
: 1111      1233      2
: 1112      1234      2      PROG_SATPTR = .SAT$START_ADDR;
: 1113      1235      2      WHILE .PROG_SATPTR NEQ 0 DO
: 1114      1236      2      BEGIN
: 1115      1237      2
: 1116      1238      2
: 1117      1239      2      ! If current SAT entry is past the address we are looking for,
: 1118      1240      2      ! then exitloop. Can do this because SAT is sorted on start
: 1119      1241      2      ! address.
: 1120      1242      2
: 1121      1243      2      IF .PROG_SATPTR[SAT$L_START] GTRA .ADDRESS
: 1122      1244      2      THEN
: 1123      1245      2      EXITLOOP;
: 1124      1246      2
: 1125      1247      2
: 1126      1248      2      ! If the given address is in the correct range, then go ahead
: 1127      1249      2      ! and search the module-level SAT.
: 1128      1250      2
: 1129      1251      2      IF (.PROG_SATPTR[SAT$L_START] LEQA .ADDRESS) AND
: 1130      1252      2      (.PROG_SATPTR[SAT$L_END] GEQA .ADDRESS)
: 1131      1253      2      THEN
: 1132      1254      2      BEGIN
: 1133      1255      2
: 1134      1256      2      ! Search the module level SAT.
: 1135      1257      2
: 1136      1258      2      SEARCH_MODULE_SAT (.PROG_SATPTR, .ADDRESS, MODRSTPTR, RSTPTR,
: 1137      1259      2      TMP_BYTE_OFFSET, .PRINT_FLAG);
: 1138      1260      2
: 1139      1261      2
: 1140      1262      2      ! If this is the first module found, then print the address.
: 1141      1263      2
: 1142      1264      2      IF (.PRINT_FLAG) AND (NOT .MODULE_FOUND)
: 1143      1265      2      THEN
: 1144      1266      2      BEGIN
: 1145      1267      2      DBG$PRINT (UPLIT BYTE (%ASCII 'address !XL'), .ADDRESS);
```


1146 1268 5
1147 1269 5
1148 1270 5
1149 1271 5
1150 1272 4
1151 1273 4
1152 1274 4
1153 1275 4
1154 1276 4
1155 1277 4
1156 1278 4
1157 1279 4
1158 1280 4
1159 1281 4
1160 1282 4
1161 1283 4
1162 1284 4
1163 1285 4
1164 1286 5
1165 1287 5
1166 1288 5
1167 1289 6
1168 1290 6
1169 1291 6
1170 1292 6
1171 1293 5
1172 1294 4
1173 1295 4
1174 1296 4
1175 1297 4
1176 1298 4
1177 1299 4
1178 1300 4
1179 1301 4
1180 1302 5
1181 1303 5
1182 1304 5
1183 1305 6
1184 1306 6
1185 1307 6
1186 1308 6
1187 1309 5
1188 1310 4
1189 1311 4
1190 1312 4
1191 1313 4
1192 1314 4
1193 1315 4
1194 1316 5
1195 1317 5
1196 1318 5
1197 1319 5
1198 1320 5
1199 1321 5
1200 1322 5
1201 1323 5
1202 1324 6

```
PRINT BIT_OFFSET(BIT_OFFSET);
DBG$PRINT (UPLIT BYTE (%ASCII ': '));
DBG$NEWLINE();
DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
END;
MODULE_FOUND = TRUE;

! Calculate offset; determine what to print.
!
SELECTONE TRUE OF
SET

! If the module is not set, then print a message to that
! effect.
[.MODRSTPTR[RST$V_MODSET] EQL 0]:
BEGIN
IF .PRINT_FLAG
THEN
BEGIN
DBG$STA SYMNAME (.MODRSTPTR, SYMBOL_NAME);
DBG$PRINT (UPLIT BYTE (%ASCII 'in module !AC, module not set'), .SYMBOL_NAME);
DBG$NEWLINE();
END;
END;

! If the module is set, but the RST pointer is 0, then the
! address was found in the module, but could not be symbolized.
! Print a message to this effect.
[.RSTPTR EQL 0]:
BEGIN
IF .PRINT_FLAG
THEN
BEGIN
DBG$STA SYMNAME (.MODRSTPTR, SYMBOL_NAME);
DBG$PRINT (UPLIT BYTE (%ASCII 'in module !AC, no symbolization'), .SYMBOL_NAME);
DBG$NEWLINE();
END;
END;

! There is a symbolization.
[OTHERWISE]:
BEGIN

! If there is currently no symbolization for the address, then
! assume this one to be the best.
! If there is a prior symbolization, see if this one is better.
! If the offset is 0, there can be no better match.
IF (.MODRSTPTR[RST$V_MODSET]) AND (.BYTE_OFFSET NEQ 0)
```


1203 1325 5
1204 1326 5
1205 1327 6
1206 1328 5
1207 1329 6
1208 1330 6
1209 1331 6
1210 1332 5
1211 1333 5
1212 1334 5
1213 1335 5
1214 1336 5
1215 1337 5
1216 1338 5
1217 1339 6
1218 1340 6
1219 1341 6
1220 1342 6
1221 1343 6
1222 1344 6
1223 1345 6
1224 1346 6
1225 1347 6
1226 1348 6
1227 1349 6
1228 1350 5
1229 1351 5
1230 1352 4
1231 1353 4
1232 1354 3
1233 1355 3
1234 1356 3
1235 1357 3
1236 1358 3
1237 1359 3
1238 1360 2
1239 1361 2
1240 1362 2
1241 1363 2
1242 1364 3
1243 1365 3
1244 1366 3
1245 1367 3
1246 1368 2
1247 1369 2
1248 1370 2
1249 1371 2
1250 1372 2
1251 1373 2
1252 1374 2
1253 1375 2
1254 1376 3
1255 1377 3
1256 1378 3
1257 1379 2
1258 1380 2
1259 1381 2

```
THEN
  IF (.BYTE_OFFSET LSS 0) OR
    (.TMP_BYTE_OFFSET LSS .BYTE_OFFSET)
  THEN
    BEGIN
      SYMID = .RSTPTR;
      BYTE_OFFSET = .TMP_BYTE_OFFSET;
    END;

    ! If the print_flag is set, output the symbolization.
    IF .PRINT_FLAG
    THEN
      BEGIN

        ! Build the ASCII string for the symbol.
        DBG$PRINT SYMBOL_PATHNAME(.RSTPTR);
        IF .TMP_BYTE_OFFSET NEQ 0
        THEN
          DBG$PRINT OFFSET (.TMP_BYTE_OFFSET);
          PRINT BIT_OFFSET(BIT_OFFSET);
          DBG$NEWLINE();
        END;
      END;
    END;
  END;
  TES;
END;

! Get the next program level SAT entry.
!
! PROG_SATPTR = .PROG_SATPTR[SAT$L_FLINK];
!
END;

IF .SYMID EQL 0
THEN
  BEGIN
    BYTE_OFFSET = 0;
    BIT_OFFSET = .BYTE_OFFSET * 8 + .BIT_OFFSET;
    RETURN .MODULE_FOUND;
  END;

! This is called from DBG$PC_TO_SYMID. Line rst entry will not be
! created. DBG$SEARCH_SAT serves as RST look up.
IF ACTUALCOUNT() EQL 5
THEN
  BEGIN
    BIT_OFFSET = .BYTE_OFFSET * 8 + .BIT_OFFSET;
    RETURN .MODULE_FOUND;
  END;
```



```
: 1260      1382  2      ! If the address is a code address, then print the symbol
: 1261      1383  2      ! name and corresponding line number.  If there is no line
: 1262      1384  2      ! number, then just print the routine name and offset.
: 1263      1385  2
: 1264      1386  2
: 1265      1387  2      IF (.SYMID[RST$B_KIND] EQL RST$K_ROUTINE) OR
: 1266      1388  2      (.SYMID[RST$B_KIND] EQL RST$K_BLOCK) OR
: 1267      1389  2      (.SYMID[RST$B_KIND] EQL RST$K_LABEL) OR
: 1268      1390  2      (.SYMID[RST$B_KIND] EQL RST$K_ENTRY)
: 1269      1391  2      THEN
: 1270      1392  2          BEGIN
: 1271      1393  2              LOCAL
: 1272      1394  2                  LINE_NO,           ! Line no. corresponding to code address.
: 1273      1395  2                  STMT_NO,           ! statement no.
: 1274      1396  2                  START_PC,         ! Beginning pc address for line.
: 1275      1397  2                  END_PC,           ! Dummy.
: 1276      1398  2                  MOD_SYMID : REF RST$ENTRY, ! Dummy.
: 1277      1399  2                  TMP_OFFSET,        ! Temporay offset
: 1278      1400  2                  TMP_SYMID : REF RST$ENTRY; ! Temporay symid for %line
: 1279      1401  2
: 1280      1402  2      ! Lookup the line number; if found, then print the symbol name
: 1281      1403  2      ! and line number.
: 1282      1404  2      !
: 1283      1405  2      MOD_SYMID = .MODRSTPTR;
: 1284      1406  2      IF (.DBG$PC_TO_LINE_LOOKUP (.ADDRESS, LINE_NO, STMT_NO,
: 1285      1407  2                  START_PC, END_PC, MOD_SYMID))
: 1286      1408  2      AND ((.LINE_NO NEQ 0) OR (.STMT_NO NEQ 0))
: 1287      1409  2      THEN
: 1288      1410  2          BEGIN
: 1289      1411  2              IF (.BYTE_OFFSET NEQ 0) OR (.SYMID[RST$B_KIND] EQL RST$K_BLOCK)
: 1290      1412  2              THEN
: 1291      1413  2                  BEGIN
: 1292      1414  2                      SYMID = DBG$STA_LINE_NUM_RST(.SYMID, .LINE_NO, .STMT_NO,
: 1293      1415  2                          .START_PC, .END_PC);
: 1294      1416  2                      BYTE_OFFSET = .ADDRESS - .START_PC;
: 1295      1417  2
: 1296      1418  2                      ! Set the following for printing purpose.
: 1297      1419  2                      !
: 1298      1420  2                      TMP_SYMID = .SYMID;
: 1299      1421  2                      TMP_OFFSET = .BYTE_OFFSET;
: 1300      1422  2                      END
: 1301      1423  2
: 1302      1424  2              ELSE
: 1303      1425  2                  BEGIN
: 1304      1426  2                      IF .PRINT_FLAG
: 1305      1427  2                      THEN
: 1306      1428  2                          BEGIN
: 1307      1429  2                              TMP_SYMID = DBG$STA_LINE_NUM_RST(.SYMID, .LINE_NO,
: 1308      1430  2                                  .STMT_NO, .START_PC, .END_PC);
: 1309      1431  2                              TMP_OFFSET = .ADDRESS - .START_PC;
: 1310      1432  2                              TMP_SYMID[RST$L_UPSCOPEPTR] = .SYMID[RST$L_UPSCOPEPTR];
: 1311      1433  2                              END;
: 1312      1434  2                          END;
: 1313      1435  2                      END;
: 1314      1436  2
: 1315      1437  2              IF .PRINT_FLAG
: 1316      1438  2             
```


1317 1439 4
1318 1440 5
1319 1441 5
1320 1442 5
1321 1443 5
1322 1444 5
1323 1445 5
1324 1446 5
1325 1447 5
1326 1448 5
1327 1449 5
1328 1450 5
1329 1451 5
1330 1452 4
1331 1453 3
1332 1454 2
1333 1455 2
1334 1456 2
1335 1457 2
1336 1458 2
1337 1459 2
1338 1460 2
1339 1461 2
1340 1462 2
1341 1463 2
1342 1464 1

```
THEN
BEGIN
DBG$PRINT_SYMBOL_PATHNAME (.TMP_SYMID);

! If the start pc is less than the code address, then
! print the line offset.
IF .TMP_OFFSET GTR 0
THEN
DBG$PRINT_OFFSET (.TMP_OFFSET);
PRINT BIT_OFFSET(BIT_OFFSET);
DBG$NEWLINE();
END;
END;

! End of routine. If the symbol was located in some module, then
! MODULE_FOUND is true; otherwise the routine returns false.
! Set offset to original address if no symbolization.
BIT_OFFSET = .BYTE_OFFSET * 8 + .BIT_OFFSET;
RETURN .MODULE_FOUND;

END;
```

4C 58 21 20 73 73 65 72 64 64 61 0B 0008A P.AAS: .ASCII <11>\address !XL\
3C 01 00096 P.AAT: .ASCII <1>\<\
21 03 00098 P.AAU: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 0009C P.AAV: .ASCII <7>\, 0, 0>\
20 3A 02 000A4 P.AAW: .ASCII <2>\: \
2C 43 41 21 20 65 6C 75 64 6F 6D 20 6E 69 1D 000A7 P.AAX: .ASCII <29>\in module !AC, module not set\
74 65 73 20 74 6F 6E 20 65 6C 75 64 6F 6D 20 000B6
2C 43 41 21 20 65 6C 75 64 6F 6D 20 6E 69 1F 000C5 P.AAY: .ASCII <31>\in module !AC, no symbolization\
69 74 61 7A 69 6C 6F 62 6D 79 73 20 6F 6E 20 000D4
6E 6F 000E3
3C 01 000E5 P.AAZ: .ASCII <1>\<\
21 03 000E7 P.ABA: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 000EB P.ABB: .ASCII <7>\, 0, 0>\
3C 01 000F3 P.ABC: .ASCII <1>\<\
3E 30 20 2C 30 20 2C 03 000F5 P.ABD: .ASCII <3>\!UL\
2C 07 000F9 P.ABE: .ASCII <7>\, 0, 0>\

OFFC 00000
SB 00000000' EF 9E 00002
5E 24 C2 00009
58 08 AC D0 0000C

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0
.ENTRY DBG\$SEARCH_SAT, Save R2,R3,R4,R5,R6,R7,R8,- : 1154
MOVAB P.AAS, R11
SUBL2 #36, \$P
MOVL P_SYMID, R8 : 1205

54	0C	AC	D0	00010	MOVL	P_BIT_OFFSET, R4	1206	
50	04	AC	D0	00014	MOVL	ADDR, -R0	1222	
57		60	D0	00018	MOVL	(R0), ADDRESS		
64	04	A0	D0	0001B	MOVL	4(R0), (R4)	1223	
55		01	CE	0001F	MNEGL	#1, BYTE_OFFSET	1224	
		68	D4	00022	CLRL	(R8)	1225	
		5A	D4	00024	CLRL	MODULE_FOUND	1226	
	00000000'	EF	7C	00026	CLRQ	OWN_MODULE_BEGIN	1227	
53	00000000G	00	D0	0002C	MOVL	SAT\$START_ADDR, PROG_SATPTR	1234	
		03	12	00033	BNEQ	3\$	1235	
		01	31	00035	BRW	16\$		
57	04	A3	D1	00038	CMPL	4(PROG_SATPTR), ADDRESS	1243	
		F7	1A	0003C	BGTRU	2\$		
57	08	A3	D1	0003E	CMPL	8(PROG_SATPTR), ADDRESS	1252	
		03	1E	00042	BGEQU	5\$		
		01	31	00044	BRW	15\$		
59	10	AC	D0	00047	MOVL	PRINT_FLAG, R9	1259	
		59	DD	0004B	PUSHL	R9		
	04	AE	9F	0004D	PUSHAB	TMP_BYTE_OFFSET	1258	
	0C	AE	9F	00050	PUSHAB	RSTPTR		
	14	AE	9F	00053	PUSHAB	MODRSTPTR		
	0088	8F	BB	00056	PUSHR	#*M<R3,R7>		
0000V	CF	06	FB	0005A	CALLS	#6, SEARCH_MODULE_SAT		
	4E	59	E9	0005F	BLBC	R9, 7\$	1264	
	4B	5A	E8	00062	BLBS	MODULE_FOUND, 7\$		
		57	DD	00065	PUSHL	ADDRESS	1267	
		5B	DD	00067	PUSHL	R11		
00000000G	00	02	FB	00069	CALLS	#2, DBG\$PRINT		
		64	D5	00070	TSTL	(R4)	1268	
		20	13	00072	BEQL	6\$		
	0C	AB	9F	00074	PUSHAB	P.AAT		
00000000G	00	01	FB	00077	CALLS	#1, DBG\$PRINT		
		64	DD	0007E	PUSHL	(R4)		
	0E	AB	9F	00080	PUSHAB	P.AAU		
00000000G	00	02	FB	00083	CALLS	#2, DBG\$PRINT		
	12	AB	9F	0008A	PUSHAB	P.AAV		
00000000G	00	01	FB	0008D	CALLS	#1, DBG\$PRINT		
	1A	AB	9F	00094	PUSHAB	P.AAW	1269	
00000000G	00	01	FB	00097	CALLS	#1, DBG\$PRINT		
00000000G	00	00	FB	0009E	CALLS	#0, DBG\$NEWLINE	1270	
		04	DD	000A5	PUSHL	#4	1271	
		01	DD	000A7	PUSHL	#1		
00000000G	00	02	FB	000A9	CALLS	#2, DBG\$PRINT CONTROL		
	5A	01	D0	000B0	MOVL	#1, MODULE_FOUND	1273	
	52	08	AE	D0	MODRSTPTR, -R2		1285	
	17	28	A2	E8	40(R2), 8\$			
	86	59	E9	000BB	BLBC	R9, 4\$	1287	
		0C	AE	9F	000BE	PUSHAB	SYMBOL_NAME	1290
		52	DD	000C1	PUSHL	R2		
00000000G	00	02	FB	000C3	CALLS	#2, DBG\$STA_SYMNAME		
		0C	AE	DD	000CA	PUSHL	SYMBOL_NAME	1291
		1D	AB	9F	000CD	PUSHAB	P.AAX	
		1B	11	000D0	BRB	9\$		
56	04	AE	D0	000D2	MOVL	RSTPTR, R6	1301	
		1E	12	000D6	BNEQ	10\$		
74		59	E9	000D8	BLBC	R9, 15\$	1303	
	0C	AE	9F	000DB	PUSHAB	SYMBOL_NAME	1306	

00000000G	00	0C	52	DD	000DE	PUSHL	R2			
		3B	02	FB	000E0	CALLS	#2, DBG\$STA_SYMNAME			1307
			AE	DD	000E7	PUSHL	SYMBOL_NAME			
00000000G	00		AB	9F	000EA	PUSHAB	P.AAY			
			02	FB	000ED	CALLS	#2, DBG\$PRINT			1308
	11	28	52	11	000F4	BRB	14\$			1324
			A2	E9	000F6	BLBC	40(R2), 12\$			
			55	D5	000FA	TSTL	BYTE_OFFSET			
			0D	13	000FC	BEQL	12\$			
			05	19	000FE	BLSS	11\$			1326
	55		6E	D1	00100	CMPL	TMP_BYTE_OFFSET, BYTE_OFFSET			1327
			06	18	00103	BGEQ	12\$			
	68		56	D0	00105	MOVL	R6, (R8)			1330
	55		6E	D0	00108	MOVL	TMP_BYTE_OFFSET, BYTE_OFFSET			1331
	41		59	E9	0010B	BLBC	R9, 15\$			1337
			56	DD	0010E	PUSHL	R6			1344
00000000G	00		01	FB	00110	CALLS	#1, DBG\$PRINT SYMBOL_PATHNAME			
			6E	D5	00117	TSTL	TMP_BYTE_OFFSET			1345
			09	13	00119	BEQL	13\$			
			6E	DD	0011B	PUSHL	TMP_BYTE_OFFSET			1347
00000000G	00		01	FB	0011D	CALLS	#1, DBG\$PRINT_OFFSET			
			64	D5	00124	TSTL	(R4)			1348
			20	13	00126	BEQL	14\$			
		5B	AB	9F	00128	PUSHAB	P.AAZ			
00000000G	00		01	FB	0012B	CALLS	#1, DBG\$PRINT			
			64	DD	00132	PUSHL	(R4)			
		5D	AB	9F	00134	PUSHAB	P.ABA			
00000000G	00		02	FB	00137	CALLS	#2, DBG\$PRINT			
		61	AB	9F	0013E	PUSHAB	P.ABB			
00000000G	00		01	FB	00141	CALLS	#1, DBG\$PRINT			
00000000G	00		00	FB	00148	CALLS	#0, DBG\$NEWLINE			1349
	53		63	D0	0014F	MOVL	(PROG_SATPTR), PROG_SATPTR			1359
			FEDE	31	00152	BRW	1\$			1235
	56		68	D0	00155	MOVL	(R8), R6			1362
			05	12	00158	BNEQ	18\$			
			55	D4	0015A	CLRL	BYTE_OFFSET			1365
			00DD	31	0015C	BRW	26\$			1366
	05		6C	91	0015F	CMPB	(AP), #5			1374
			F8	13	00162	BEQL	17\$			
	53	14	A6	9A	00164	MOVZBL	20(R6), R3			1386
	02		53	91	00168	CMPB	R3, #2			
			0F	13	0016B	BEQL	19\$			
	03		53	91	0016D	CMPB	R3, #3			1387
			0A	13	00170	BEQL	19\$			
	04		53	91	00172	CMPB	R3, #4			1388
			05	13	00175	BEQL	19\$			
	08		53	91	00177	CMPB	R3, #8			1389
			E0	12	0017A	BNEQ	17\$			
	10	AE	08	AE	D0	0017C	MOVL	MODRSTPTR, MOD_SYMID		1405
			10	AE	9F	00181	PUSHAB	MOD_SYMID		1406
			18	AE	9F	00184	PUSHAB	END_PC		
			20	AE	9F	00187	PUSHAB	START_PC		
			28	AE	9F	0018A	PUSHAB	STMT_NO		
			30	AE	9F	0018D	PUSHAB	LINE_NO		
			57	DD	00190	PUSHL	ADDRESS			
00000000G	00		06	FB	00192	CALLS	#6, DBG\$PC_TO_LINE_LOOKUP			
	C0		50	E9	00199	BLBC	R0, 17\$			

		20	AE	D5	0019C	TSTL	LINE_NO	1408	
		05	12	0019F	BNEQ	20\$			
		1C	AE	D5	001A1	TSTL	STMT_NO		
		B6	13	001A4	BEQL	17\$			
		55	D5	001A6	20\$: TSTL	BYTE_OFFSET	1411		
		05	12	001A8	BNEQ	21\$			
	03	53	91	001AA	CMPB	R3, #3			
		25	12	001AD	BNEQ	22\$			
		14	AE	DD	001AF	21\$: PUSHL	END_PC	1415	
		1C	AE	DD	001B2	PUSHL	START_PC		
		24	AE	DD	001B5	PUSHL	STMT_NO	1414	
		2C	AE	DD	001B8	PUSHL	LINE_NO		
		56	DD	001BB	PUSHL	R6			
	00000000G	00	05	FB	001BD	CALLS	#5, DBG\$STA_LINE_NUM_RST		
	68		50	DO	001C4	MOVL	R0, (R8)		
55	57	18	AE	C3	001C7	SUBL3	START_PC, ADDRESS, BYTE_OFFSET	1416	
	50		68	DO	001CC	MOVL	(R8), TMP_SYMID	1421	
	53		55	DO	001CF	MOVL	BYTE_OFFSET, TMP_OFFSET	1422	
			23	11	001D2	BRB	23\$	1411	
	64	10	AC	E9	001D4	22\$: BLBC	PRINT_FLAG, 26\$	1427	
		14	AE	DD	001D8	PUSHL	END_PC	1431	
		1C	AE	DD	001DB	PUSHL	START_PC		
		24	AE	DD	001DE	PUSHL	STMT_NO		
		2C	AE	DD	001E1	PUSHL	LINE_NO	1430	
			56	DD	001E4	PUSHL	R6		
	00000000G	00	05	FB	001E6	CALLS	#5, DBG\$STA_LINE_NUM_RST		
53	57	18	AE	C3	001ED	SUBL3	START_PC, ADDRESS, TMP_OFFSET	1432	
	10	10	A6	DO	001F2	MOVL	16(R6), 16(TMP_SYMID)	1433	
	41	10	AC	E9	001F7	23\$: BLBC	PRINT_FLAG, 26\$	1438	
			50	DD	001FB	PUSHL	TMP_SYMID	1441	
	00000000G	00	01	FB	001FD	CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME		
			53	D5	00204	TSTL	TMP_OFFSET	1447	
			09	15	00206	BLEQ	24\$		
			53	DD	00208	PUSHL	TMP_OFFSET	1449	
	00000000G	00	01	FB	0020A	CALLS	#1, DBG\$PRINT_OFFSET		
			64	D5	00211	24\$: TSTL	(R4)	1450	
			20	13	00213	BEQL	25\$		
	00000000G	00	69	AB	9F	00215	PUSHAB	P.ABC	
			01	FB	00218	CALLS	#1, DBG\$PRINT		
			64	DD	0021F	PUSHL	(R4)		
	00000000G	00	68	AB	9F	00221	PUSHAB	P.ABD	
			02	FB	00224	CALLS	#2, DBG\$PRINT		
			6F	AB	9F	0022B	PUSHAB	P.ABE	
	00000000G	00	01	FB	0022E	CALLS	#1, DBG\$PRINT		
	00000000G	00	00	FB	00235	25\$: CALLS	#0, DBG\$NEWLINE	1451	
	74		9445	7E	0023C	26\$: MOVAQ	@(R4)+[BYTE_OFFSET], -(R4)	1461	
	50		5A	DO	00240	MOVL	MODULE_FOUND, R0	1462	
			04	00243	RET			1464	

; Routine Size: 580 bytes, Routine Base: DBG\$CODE + 04A8


```
1344 1465 1 GLOBAL ROUTINE DBGSDUMP_SAT: NOVALUE =
1345 1466 1
1346 1467 1 FUNCTION
1347 1468 1     This routine displays information about the Static Address Table.
1348 1469 1     It is called from the DUMP SAT command, which is enabled via
1349 1470 1     SET DEVELOPER 0. The information can be used as an aid to
1350 1471 1     studying the performance of the debugger.
1351 1472 1
1352 1473 1 INPUTS
1353 1474 1     none.
1354 1475 1
1355 1476 1 OUTPUTS
1356 1477 1     The length of each modules SAT chain is displayed at the terminal.
1357 1478 1
1358 1479 2 BEGIN
1359 1480 2 LOCAL
1360 1481 2     COUNT,
1361 1482 2     MODNAME,
1362 1483 2     MODRSTPTR: REF RST$ENTRY,
1363 1484 2     PROGSAT COUNT,
1364 1485 2     PROG SATPTR: REF SAT$ENTRY,
1365 1486 2     SATPTR: REF SAT$ENTRY;
1366 1487 2
1367 1488 2 ! Loop through the program-level SAT.
1368 1489 2
1369 1490 2 PROG SATPTR = .SAT$START_ADDR;
1370 1491 2 PROGSAT COUNT = 0;
1371 1492 2 WHILE .PROG_SATPTR NEQ 0 DO
1372 1493 2 BEGIN
1373 1494 2     PROGSAT COUNT = .PROGSAT COUNT + 1;
1374 1495 2     MODRSTPTR = .PROG_SATPTR[SAT$L_RSTPTR];
1375 1496 2
1376 1497 2     ! Loop through each module-level SAT, keeping a count of entries.
1377 1498 2
1378 1499 2     SATPTR = .MODRSTPTR[RST$L_SAT_PTR];
1379 1500 2     COUNT = 0;
1380 1501 2     WHILE .SATPTR NEQ 0 DO
1381 1502 2     BEGIN
1382 1503 2         COUNT = .COUNT + 1;
1383 1504 2         SATPTR = .SATPTR[SAT$L_FLINK];
1384 1505 2     END;
1385 1506 2
1386 1507 2     ! Display the information for this module SAT.
1387 1508 2
1388 1509 2     DBG$STA SYMNAME (.MODRSTPTR, MODNAME);
1389 1510 2     DBG$PRINT(UPLOT BYTE(%ASCII 'module !AC: !UL'), .MODNAME, .COUNT);
1390 1511 2     DBG$NEWLINE();
1391 1512 2
1392 1513 2     PROG_SATPTR = .PROG_SATPTR[SAT$L_FLINK];
1393 1514 2 END;
1394 1515 2 DBG$PRINT(UPLOT BYTE(%ASCII 'length of program SAT: !UL'), .PROGSAT_COUNT);
1395 1516 2 DBG$NEWLINE();
1396 1517 1 END;
```

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

55	21	20	3A	43	41	21	20	65	6C	75	64	6F	6D	0F	00101	P.ABF: .ASCII <15>\module !AC: !UL\	:
														4C	00110		:
67	6F	72	70	20	66	6F	20	68	74	67	6E	65	6C	1A	00111	P.ABG: .ASCII <26>\length of program SAT: !UL\	:
			4C	55	21	20	3A	54	41	53	20	6D	61	72	00120		:

															.PSECT	DBG\$CODE,NOWRT, SHR, PIC,0	
															.ENTRY	DBG\$DUMP_SAT, Save R2,R3,R4,R5,R6,R7,R8	: 1465
58	00000000G	00	9E	00002	57	00000000G	00	9E	00009	MOVAB	DBG\$NEWLINE, R8	:					
5E		04	C2	00010	MOVAB	DBG\$PRINT, R7	:										
52	00000000G	00	D0	00013	SUBL2	#4, SP	:										
		55	D4	0001A	MOVL	SAT\$START_ADDR, PROG_SATPTR	: 1490										
		52	D5	0001C	CLRL	PROGSAT_COUNT	: 1491										
		38	13	0001E	TSTL	PROG_SATPTR	: 1492										
		55	D6	00020	BEQL	4\$	:										
53	0C	A2	D0	00022	INCL	PROGSAT_COUNT	: 1494										
54	18	A3	D0	00026	MOVL	12(PROG_SATPTR), MODRSTPTR	: 1495										
		56	D4	0002A	MOVL	24(MODRSTPTR), SATPTR	: 1499										
		54	D5	0002C	CLRL	COUNT	: 1500										
		07	13	0002E	TSTL	SATPTR	: 1501										
		56	D6	00030	BEQL	3\$	:										
54		64	D0	00032	INCL	COUNT	: 1503										
		F5	11	00035	MOVL	(SATPTR), SATPTR	: 1504										
	4008	8F	BB	00037	BRB	2\$	: 1501										
00000000G	00	02	FB	0003B	PUSHR	#*M<R3,SP>	: 1509										
		56	DD	00042	CALLS	#2, DBG\$STA_SYMNAME	:										
	04	AE	DD	00044	PUSHL	COUNT	: 1510										
	00000000'	EF	9F	00047	PUSHL	MODNAME	:										
67		03	FB	0004D	PUSHAB	P.ABF	:										
68		00	FB	00050	CALLS	#3, DBG\$PRINT	:										
52		62	D0	00053	CALLS	#0, DBG\$NEWLINE	: 1511										
		C4	11	00056	MOVL	(PROG_SATPTR), PROG_SATPTR	: 1513										
		55	DD	00058	BRB	1\$	: 1492										
	00000000'	EF	9F	0005A	PUSHL	PROGSAT_COUNT	: 1515										
67		02	FB	00060	PUSHAB	P.ABG	:										
68		00	FB	00063	CALLS	#2, DBG\$PRINT	: 1516										
		04	00066	CALLS	#0, DBG\$NEWLINE	: 1517											
				RET													

; Routine Size: 103 bytes, Routine Base: DBG\$CODE + 06EC


```
1398 1518 1 GLOBAL ROUTINE DBG$SEARCH_VAX_CALL_STACK(ADDR, P_SYMID, P_BIT_OFFSET, PRINT_FLAG) =
1399 1519 1
1400 1520 1 FUNCTION
1401 1521 1     This routine searches the VAX call stack in an attempt to symbolize the
1402 1522 1     address.
1403 1523 1
1404 1524 1 INPUTS
1405 1525 1     ADDR          - Address Descriptor.
1406 1526 1
1407 1527 1     P_SYMID       - The address of a longword location to receive the
1408 1528 1                   unique symbol identifier.
1409 1529 1
1410 1530 1     P_BIT_OFFSET  - The bit offset off of P_SYMID.
1411 1531 1
1412 1532 1
1413 1533 1     PRINT_FLAG    - If false, no output will be done. If true, then
1414 1534 1                   the symbolization of the virtual address will be
1415 1535 1                   output to the terminal.
1416 1536 1
1417 1537 1 OUTPUTS
1418 1538 1     If successful, the routine prints the symbolization of the address
1419 1539 1     (if the print flag was set). The location defined by P_SYMID contains
1420 1540 1     the RST pointer (unique symbol identifier) of the symbol that matched.
1421 1541 1     The location defined by P_BIT_OFFSET contains the bit offset. The routine
1422 1542 1     returns true. All of this occurs even if only a partial symbolization
1423 1543 1     is possible (eg. just routine name).
1424 1544 1
1425 1545 1     If the address is not found in the stack frame, then P_SYMID is 0, and
1426 1546 1     P_BIT_OFFSET is ADDR's bit offset field. The routine returns false.
1427 1547 1
1428 1548 1
1429 1549 2 BEGIN
1430 1550 2
1431 1551 2 MAP
1432 1552 2     ADDR: REF DBG$ADDRESS_DESC;
1433 1553 2
1434 1554 2 BIND
1435 1555 2     SYMID          = .P_SYMID: REF RST$ENTRY,  ! Unique symbol identifier.
1436 1556 2     BIT_OFFSET     = .P_BIT_OFFSET;           ! Offset off of FP.
1437 1557 2
1438 1558 2 LITERAL
1439 1559 2     FP = 13,
1440 1560 2     SP = 14,
1441 1561 2     LONGWORD = 4;
1442 1562 2
1443 1563 2 LOCAL
1444 1564 2     ADDRESS,          ! Byte address
1445 1565 2     BYTE_OFFSET,     ! Byte offset
1446 1566 2     COUNT,           ! Invocation count
1447 1567 2     ENDADDR,         ! End address of routine
1448 1568 2     FRAME_FOUND_FLAG,
1449 1569 2     FRAMEPTR         : REF BLOCK[, BYTE],   ! Ptr to current VAX call frame.
1450 1570 2     FRAMEPTR2        : REF BLOCK[, BYTE],   ! Ptr to current VAX call frame.
1451 1571 2     PCVAL,           ! Current PC.
1452 1572 2     PCVAL2,          ! Current PC.
1453 1573 2     TMP_BYTE_OFFSET, ! Temp offset for calc closest symbol.
1454 1574 2     VALSPEC          : REF DST$VAL_SPEC,    ! Ptr to DST value spec.
```



```
: 1455      1575      2      TMP SYMID      : REF RST$ENTRY,      : Temp ptr to symbol RST entry.
: 1456      1576      2      MODRSTPTR      : REF RST$ENTRY,      : Module RST pointer
: 1457      1577      2      RSTPTR      : REF RST$ENTRY,      : Routine RST ptr.
: 1458      1578      2      RSTPTR2      : REF RST$ENTRY,      : Routine RST ptr.
: 1459      1579      2      SATPTR      : REF SAT$ENTRY,
: 1460      1580      2      STARTADDR,      : Start address of routine
: 1461      1581      2      SYMBOL_NAME      : REF BLOCK[ , BYTE],      : Ptr to counted ASCII string.
: 1462      1582      2      SPVAL      : REF VECTOR[ , LONG],      : Current call frame SP value.
: 1463      1583      2      REGMASK      : BITVECTOR[16];      : Reg save mask from the call frame.
: 1464      1584      2
: 1465      1585      2
: 1466      1586      2      : Initialize the SP, PC and frame pointer to their current (top of stack)
: 1467      1587      2      : values.
: 1468      1588      2
: 1469      1589      2      PCVAL = .DBG$RUNFRAME[DBG$USER_PC];
: 1470      1590      2      SPVAL = .DBG$RUNFRAME[DBG$USER_SP];
: 1471      1591      2      FRAMEPTR = .DBG$RUNFRAME[DBG$USER_FP];
: 1472      1592      2
: 1473      1593      2
: 1474      1594      2      : Assume symbolization not possible.
: 1475      1595      2
: 1476      1596      2      SYMID = 0;
: 1477      1597      2      BYTE OFFSET = -1;
: 1478      1598      2      ADDRESS = .ADDR[DBG$ADDRESS_BYTE_ADDR];
: 1479      1599      2      BIT_OFFSET = .ADDR[DBG$ADDRESS_BIT_OFFSET];
: 1480      1600      2
: 1481      1601      2
: 1482      1602      2      : Loop through the stack.
: 1483      1603      2
: 1484      1604      2      WHILE TRUE DO
: 1485      1605      2          BEGIN
: 1486      1606      2
: 1487      1607      2
: 1488      1608      2          : If we are at the bottom of the stack, then exit the loop.
: 1489      1609      2
: 1490      1610      2          IF (.PCVAL EQL 0) OR
: 1491      1611      2              (.FRAMEPTR[SF$A_HANDLER] EQL DBG$FINAL_HANDL)
: 1492      1612      2          THEN
: 1493      1613      2              EXITLOOP;
: 1494      1614      2
: 1495      1615      2
: 1496      1616      2          : See if there is local data on top of the stack.
: 1497      1617      2
: 1498      1618      2          IF .FRAMEPTR GTRA .SPVAL
: 1499      1619      2          THEN
: 1500      1620      2
: 1501      1621      2
: 1502      1622      2              : There is, so see if address is within the data area.
: 1503      1623      2
: 1504      1624      2              IF (.ADDRESS GEQA .SPVAL) AND
: 1505      1625      2                  (.ADDRESS LSSA .FRAMEPTR)
: 1506      1626      2              THEN
: 1507      1627      2                  BEGIN
: 1508      1628      2                      RSTPTR = FIND_ROUTINE (.PCVAL, .PRINT_FLAG);
: 1509      1629      2                      IF .RSTPTR EQL 0
: 1510      1630      2                      THEN
: 1511      1631      2                          BEGIN
```



```

: 1512      1632      5
: 1513      1633      5
: 1514      1634      6
: 1515      1635      6
: 1516      1636      6
: 1517      1637      6
: 1518      1638      6
: 1519      1639      6
: 1520      1640      6
: 1521      1641      6
: 1522      1642      5
: 1523      1643      5
: 1524      1644      5
: 1525      1645      4
: 1526      1646      4
: 1527      1647      4
: 1528      1648      4
: 1529      1649      5
: 1530      1650      5
: 1531      1651      5
: 1532      1652      5
: 1533      1653      5
: 1534      1654      5
: 1535      1655      5
: 1536      1656      5
: 1537      1657      5
: 1538      1658      6
: 1539      1659      6
: 1540      1660      6
: 1541      1661      6
: 1542      1662      6
: 1543      1663      6
: 1544      1664      6
: 1545      1665      6
: 1546      1666      6
: 1547      1667      6
: 1548      1668      6
: 1549      1669      6
: 1550      1670      6
: 1551      1671      6
: 1552      1672      6
: 1553      1673      5
: 1554      1674      5
: 1555      1675      4
: 1556      1676      4
: 1557      1677      4
: 1558      1678      4
: 1559      1679      4
: 1560      1680      4
: 1561      1681      4
: 1562      1682      4
: 1563      1683      4
: 1564      1684      4
: 1565      1685      4
: 1566      1686      4
: 1567      1687      4
: 1568      1688      4

      IF .PRINT_FLAG
      THEN
      BEGIN
      DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
      PRINT BIT_OFFSET(BIT_OFFSET);
      DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
      DBG$NEWLINE();
      DBG$PRINT CONTROL (DBG$K_PRTSET LMARGIN, TAB);
      DBG$PRINT (UPLIT BYTE (%ASCIC 'data item on stack in unknown module'));
      DBG$NEWLINE();
      END;

      RETURN TRUE;
      END;

      IF .RSTPTR[RST$B_KIND] EQL RST$K_MODULE
      THEN
      BEGIN

      ! We have located the PC in a module, but since the module
      ! is not set, can't get any more information other than
      ! module name. Print that.

      IF .PRINT_FLAG
      THEN
      BEGIN
      DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
      PRINT BIT_OFFSET(BIT_OFFSET);
      DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
      DBG$NEWLINE();
      DBG$PRINT CONTROL (DBG$K_PRTSET LMARGIN, TAB);
      DBG$STA SYMNAME (.RSTPTR, SYMBOL_NAME);
      DBG$PRINT (UPLIT BYTE (%ASCIC 'data item on stack in module !AC'), .SYMBOL_NAME);
      IF .RSTPTR[RST$V_MODSET]
      THEN
      DBG$PRINT (UPLIT BYTE (%ASCIC ', module set'))
      ELSE
      DBG$PRINT (UPLIT BYTE (%ASCIC ', module not set'));

      DBG$NEWLINE();
      END;
      RETURN TRUE;
      END;

      ! We have found the routine containing the PC. Try for further
      ! symbolization.

      MODRSTPTR = .RSTPTR;
      WHILE .MODRSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
      MODRSTPTR = .MODRSTPTR[RST$L_UPSCOPEPTR];
      TMP_SYMID = .MODRSTPTR[RST$L_SYMCHNPTR];

      ! Search through the routine to find the symbol most immediately
      ! preceding the given address.
```



```
: 1569      1689      4
: 1570      1690      4
: 1571      1691      5
: 1572      1692      5
: 1573      1693      5
: 1574      1694      5
: 1575      1695      5
: 1576      1696      5
: 1577      1697      5
: 1578      1698      5
: 1579      1699      5
: 1580      1700      5
: 1581      1701      5
: 1582      1702      5
: 1583      1703      6
: 1584      1704      6
: 1585      1705      6
: 1586      1706      6
: 1587      1707      6
: 1588      1708      6
: 1589      1709      6
: 1590      1710      6
: 1591      1711      6
: 1592      1712      6
: 1593      1713      6
: 1594      1714      6
: 1595      1715      6
: 1596      1716      6
: 1597      1717      7
: 1598      1718      7
: 1599      1719      7
: 1600      1720      7
: 1601      1721      7
: 1602      1722      7
: 1603      1723      6
: 1604      1724      6
: 1605      1725      5
: 1606      1726      5
: 1607      1727      5
: 1608      1728      5
: 1609      1729      5
: 1610      1730      6
: 1611      1731      6
: 1612      1732      6
: 1613      1733      6
: 1614      1734      6
: 1615      1735      6
: 1616      1736      6
: 1617      1737      6
: 1618      1738      6
: 1619      1739      7
: 1620      1740      7
: 1621      1741      7
: 1622      1742      8
: 1623      1743      8
: 1624      1744      8
: 1625      1745      8
```

```
!
WHILE .TMP_SYMID NEQ 0 DO
  BEGIN
    LOCAL
      TMPPTR;

    ! Get the DST value spec for the symbol. If the symbol
    ! is not bound to FP (or SP), then the valspec will be 0.
    VALSPEC = IS_SYMBOL_BOUND_TO_REG (.TMP_SYMID, FP);
    TMPPTR = .FRAMEPTR;
    IF .VALSPEC EQL 0
    THEN
      BEGIN
        VALSPEC = IS_SYMBOL_BOUND_TO_REG (.TMP_SYMID, SP);

        ! In here we need to do a special check for JSB routine
        ! local variables. (usually it is defined as SP - offset
        ! in DST). The address of those variables are usually
        ! relative to SP pointer at the execution time. We do
        ! not have the correct information for SP. So we do
        ! our best guess at this case. Ignore it if we think
        ! we have found this case.
        IF .VALSPEC NEQ 0
        THEN
          BEGIN
            IF .VALSPEC[DST$L_VS_VALUE] LSS 0
            THEN
              VALSPEC = 0
            ELSE
              TMPPTR = .SPVAL;
            END;
          END;
        END;

      IF .VALSPEC NEQ 0
      THEN
        BEGIN

          ! See if the symbol is a displacement off of the FP,
          ! or SP.
          IF NOT .VALSPEC[DST$V_VS_INDIRECT] AND
            .VALSPEC[DST$V_VS_DISP]
          THEN
            BEGIN
              IF .TMP_SYMID[RST$L_UPSCOPEPTR] EQL .RSTPTR
              THEN
                BEGIN

                  ! If this is the first match, or a better
```



```
: 1626      1746      8
: 1627      1747      8
: 1628      1748      8
: 1629      1749      8
: 1630      1750      9
: 1631      1751      9
: 1632      1752      9
: 1633      1753      8
: 1634      1754      9
: 1635      1755      9
: 1636      1756      9
: 1637      1757      8
: 1638      1758      7
: 1639      1759      6
: 1640      1760      5
: 1641      1761      5
: 1642      1762      5
: 1643      1763      5
: 1644      1764      5
: 1645      1765      5
: 1646      1766      5
: 1647      1767      5
: 1648      1768      5
: 1649      1769      5
: 1650      1770      5
: 1651      1771      5
: 1652      1772      5
: 1653      1773      4
: 1654      1774      4
: 1655      1775      4
: 1656      1776      4
: 1657      1777      4
: 1658      1778      4
: 1659      1779      4
: 1660      1780      4
: 1661      1781      4
: 1662      1782      4
: 1663      1783      4
: 1664      1784      4
: 1665      1785      5
: 1666      1786      5
: 1667      1787      5
: 1668      1788      5
: 1669      1789      5
: 1670      1790      5
: 1671      1791      6
: 1672      1792      7
: 1673      1793      6
: 1674      1794      7
: 1675      1795      7
: 1676      1796      7
: 1677      1797      7
: 1678      1798      7
: 1679      1799      7
: 1680      1800      7
: 1681      1801      7
: 1682      1802      7
```

```
! match, set symid and offset.
TMP_BYTE_OFFSET = .ADDRESS -
    (.TMPPTR + .VALSPEC[DST$L VS VALUE]);
IF ((.TMP_BYTE_OFFSET LSS .BYTE_OFFSET) AND
    (.TMP_BYTE_OFFSET GEQ 0))
OR (.BYTE_OFFSET LSS 0)
THEN
    BEGIN
        SYMID = .TMP_SYMID;
        BYTE_OFFSET = .TMP_BYTE_OFFSET;
    END;
END;
END;

! If the offset is 0, have found a perfect match.
IF .BYTE_OFFSET EQL 0
THEN
    EXITLOOP;

! Loop for next symbol.
TMP_SYMID = .TMP_SYMID[RST$L_SYMCHNPTR];
END;

! Before returning, we want to take care of the case
! where the found SYMID was declared in a routine
! with several invocations on the stack. We walk down
! the stack again, counting the times we pass the routine.
! This enables us to attach an invocation number to
! the SYMID.
COUNT = 0;
IF .SYMID NEQ 0
THEN
    BEGIN
        STARTADDR = .RSTPTR[RST$L_STARTADDR];
        ENDADDR = .RSTPTR[RST$L_ENDADDR];
        PCVAL2 = .DBG$RUNFRAME[DBG$L_USER_PC];
        FRAMEPTR2 = .DBG$RUNFRAME[DBG$L_USER_FP];
        WHILE .FRAMEPTR2 LSS .FRAMEPTR DO
            BEGIN
                IF (.PCVAL2 GEQU .STARTADDR) AND (.PCVAL2 LEQU .ENDADDR)
                THEN
                    BEGIN
                        ! The PC from this CALL frame is in the address range of the routine
                        ! we are looking for. However, to make sure the PC is not really in
                        ! a nested routine within the desired routine, we search the Module
                        ! SAT starting at the desired routine's SAT entry looking for nested
                        ! routines which cover the CALL frame's PC value. If we find such a
                        ! routine, the CALL frame is not for the desired routine.
                    END;
                END;
            END;
        END;
    END;
```



```
: 1683 1803 7
: 1684 1804 7
: 1685 1805 7
: 1686 1806 7
: 1687 1807 7
: 1688 1808 7
: 1689 1809 7
: 1690 1810 7
: 1691 1811 7
: 1692 1812 7
: 1693 1813 7
: 1694 1814 7
: 1695 1815 7
: 1696 1816 7
: 1697 1817 7
: 1698 1818 7
: 1699 1819 7
: 1700 1820 7
: 1701 1821 7
: 1702 1822 7
: 1703 1823 7
: 1704 1824 7
: 1705 1825 7
: 1706 1826 8
: 1707 1827 8
: 1708 1828 8
: 1709 1829 8
: 1710 1830 8
: 1711 1831 8
: 1712 1832 8
: 1713 1833 8
: 1714 1834 8
: 1715 1835 8
: 1716 1836 8
: 1717 1837 8
: 1718 1838 8
: 1719 1839 8
: 1720 1840 8
: 1721 1841 8
: 1722 1842 8
: 1723 1843 8
: 1724 1844 8
: 1725 1845 9
: 1726 1846 8
: 1727 1847 9
: 1728 1848 9
: 1729 1849 9
: 1730 1850 8
: 1731 1851 8
: 1732 1852 8
: 1733 1853 8
: 1734 1854 8
: 1735 1855 7
: 1736 1856 7
: 1737 1857 7
: 1738 1858 7
: 1739 1859 7
```

```
:
: FRAME FOUND FLAG = TRUE;
: SATPTR = .RSTPTR[RST$RSTNSATPTR];
:
: WARNING -- We can get into trouble here. Previously, we have
: assumed that the SAT is always around. This may not be the
: case if this module has been canceled. There are times when
: the module could be canceled and then set again to make us
: believe the the SAT is valid for this RST, but it is not! To
: correct the problem, when a module is canceled the field
: RST$RSTNSATPTR is set to ZERO for each routine.
: So if the module for this RST has been canceled, SATPTR will
: be zero from the above statement. The problem is that this
: assumes there are no nested routines that truly require the
: correct context information. This is, of course, WRONG. A
: way of saving and getting to the SAT information must be
: found in the future. B.A. Becker MAY-1984
:
: IF .SATPTR NEQ 0
: THEN
:   SATPTR = .SATPTR[SAT$FLINK];
:
: WHILE TRUE DO
:   BEGIN
:     LOCAL
:       NEW_RSTPTR: REF RST$ENTRY;
:
:     : If there are no more SAT entries in the chain or if they no
:     : longer cover the PCVAL address, exit the SAT loop.
:
:     IF .SATPTR EQL 0 THEN EXITLOOP;
:     IF .SATPTR[SAT$START] GTRU .PCVAL2 THEN EXITLOOP;
:
:     : If this SAT entry is for a routine which covers the PCVAL
:     : address, we clear FRAME_FOUND_FLAG because the PC is in this
:     : nested routine instead of the routine we are looking for.
:
:     NEW_RSTPTR = .SATPTR[SAT$RSTPTR];
:     IF (.PCVAL2 GEQU .SATPTR[SAT$START]) AND
:       (.PCVAL2 LEQU .SATPTR[SAT$END]) AND
:       (.NEW_RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE)
:     THEN
:       BEGIN
:         FRAME_FOUND_FLAG = FALSE;
:         EXITLOOP;
:       END;
:
:     : Link on to the next SAT entry.
:
:     SATPTR = .SATPTR[SAT$FLINK];
:   END;
:
: : If the CALL frame we found really is for the desired routine,
: : then increment the invocation count.
```



```
: 1740 1860 7
: 1741 1861 7
: 1742 1862 7
: 1743 1863 6
: 1744 1864 6
: 1745 1865 6
: 1746 1866 6
: 1747 1867 6
: 1748 1868 6
: 1749 1869 5
: 1750 1870 4
: 1751 1871 4
: 1752 1872 4
: 1753 1873 4
: 1754 1874 4
: 1755 1875 4
: 1756 1876 4
: 1757 1877 4
: 1758 1878 4
: 1759 1879 4
: 1760 1880 4
: 1761 1881 4
: 1762 1882 4
: 1763 1883 4
: 1764 1884 5
: 1765 1885 5
: 1766 1886 5
: 1767 1887 5
: 1768 1888 5
: 1769 1889 5
: 1770 1890 5
: 1771 1891 5
: 1772 1892 6
: 1773 1893 6
: 1774 1894 6
: 1775 1895 6
: 1776 1896 6
: 1777 1897 5
: 1778 1898 6
: 1779 1899 6
: 1780 1900 6
: 1781 1901 6
: 1782 1902 6
: 1783 1903 6
: 1784 1904 5
: 1785 1905 5
: 1786 1906 4
: 1787 1907 4
: 1788 1908 4
: 1789 1909 4
: 1790 1910 4
: 1791 1911 4
: 1792 1912 3
: 1793 1913 3
: 1794 1914 3
: 1795 1915 3
: 1796 1916 3
```

```
IF .FRAME_FOUND_FLAG
THEN
    COUNT = .COUNT + 1;
END;

! Keep looping through the stack.
PCVAL2 = .FRAMEPTR2[SF$SAVE_PC];
FRAMEPTR2 = .FRAMEPTR2[SF$SAVE_FP];
END;
END;

! Now build a new SYMID which includes the invocation number.
IF .COUNT GTR 0
THEN
    SYMID = DBG$BUILD_INVOC_RST(.SYMID, .COUNT);

! If a symbol was not found, then just print the routine name.
! Otherwise, print the symbol pathname (and offset, if
! applicable).
IF .PRINT_FLAG
THEN
    BEGIN
        DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
        PRINT BIT_OFFSET(BIT_OFFSET);
        DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
        DBG$NEWLINE();
        DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
        IF .SYMID EQL 0
        THEN
            BEGIN
                DBG$PRINT (UPLIT BYTE (%ASCIC 'data item on stack in routine '));
                DBG$PRINT_SYMBOL_PATHNAME(.RSTPTR);
            END
        ELSE
            BEGIN
                DBG$PRINT_SYMBOL_PATHNAME(.SYMID);
                IF .BYTE_OFFSET NEQ 0
                THEN
                    DBG$PRINT_OFFSET (.BYTE_OFFSET);
                PRINT_BIT_OFFSET(BIT_OFFSET);
            END;
            DBG$NEWLINE();
        END;

        IF .BYTE_OFFSET LSS 0 THEN SYMID = 0;
        IF .SYMID EQL 0 THEN BYTE_OFFSET = 0;
        BIT_OFFSET = .BYTE_OFFSET * 8 + .BIT_OFFSET;
        RETURN TRUE;
    END;
```

```
! It was not a local data item, so see if it is a call frame address.
! Determine what the value of SP will be when this call frame is torn
```



```

: 1797
: 1798
: 1799
: 1800
: 1801
: 1802
: 1803
: 1804
: 1805
: 1806
: 1807
: 1808
: 1809
: 1810
: 1811
: 1812
: 1813
: 1814
: 1815
: 1816
: 1817
: 1818
: 1819
: 1820
: 1821
: 1822
: 1823
: 1824
: 1825
: 1826
: 1827
: 1828
: 1829
: 1830
: 1831
: 1832
: 1833
: 1834
: 1835
: 1836
: 1837
: 1838
: 1839
: 1840
: 1841
: 1842
: 1843
: 1844
: 1845
: 1846
: 1847
: 1848
: 1849
: 1850
: 1851
: 1852
: 1853

```

```

1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
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
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973

```

```

! down by the RET instruction. The new SP value will be the bottom
! boundary of this call frame.
REGMASK = .FRAMEPTR[SF$W_SAVE_MASK];
SPVAL = FRAMEPTR[SF$SL_SAVE_REGS];
INCR I FROM 0 TO 11 DO
    IF .REGMASK[I] THEN SPVAL = .SPVAL + LONGWORD;
SPVAL = .SPVAL + .FRAMEPTR[SF$V_STACKOFFS];
IF .FRAMEPTR[SF$V_CALLS]
THEN
    SPVAL = .SPVAL + 4*(.SPVAL[0] + 1);

! See if the address falls into the current call frame (which is bounded
! by the current frameptr and the sp value just calculated).
IF (.ADDRESS GEQA .FRAMEPTR) AND
    (.ADDRESS LSSA .SPVAL)
THEN
    BEGIN

        ! Find the routine that contains the PC.
        RSTPTR = FIND_ROUTINE (.PCVAL, .PRINT_FLAG);
        IF .RSTPTR EQL 0
        THEN
            BEGIN
                IF .PRINT_FLAG
                THEN
                    BEGIN
                        DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
                        PRINT_BIT_OFFSET(BIT_OFFSET);
                        DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
                        DBG$NEWLINE();
                        DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
                        DBG$PRINT (UPLIT BYTE (%ASCIC 'stack frame address in unknown module'));
                        DBG$NEWLINE();
                    END;
                RETURN TRUE;
            END;
        IF .RSTPTR[RST$B_KIND] EQL RST$K_MODULE
        THEN
            BEGIN

                ! We have located the PC in a module, but since the module
                ! is not set, can't get any more information other than
                ! module name. Print that.
                IF .PRINT_FLAG
                THEN
                    BEGIN
                        DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
                        PRINT_BIT_OFFSET(BIT_OFFSET);
                    END;
            END;
        END;
    END;

```



```

: 1854      1974 6      DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
: 1855      1975 6      DBG$NEWLINE();
: 1856      1976 6      DBG$PRINT CONTROL (DBG$K_PRTSET LMARGIN, TAB);
: 1857      1977 6      DBG$STA SYMNAME (.RSTPTR, SYMBOL_NAME);
: 1858      1978 6      DBG$PRINT (UPLIT BYTE (%ASCIC
: 1859      1979 6      'stack frame address in module !AC'), .SYMBOL_NAME);
: 1860      1980 6      IF .RSTPTR[RST$V_MODSET]
: 1861      1981 6      THEN
: 1862      1982 6          DBG$PRINT( UPLIT BYTE (%ASCIC ', module set'))
: 1863      1983 6      ELSE
: 1864      1984 6          DBG$PRINT( UPLIT BYTE (%ASCIC ', module not set'));
: 1865      1985 6
: 1866      1986 6      DBG$NEWLINE();
: 1867      1987 5      END;
: 1868      1988 5
: 1869      1989 5      RETURN TRUE;
: 1870      1990 5      END
: 1871      1991 5
: 1872      1992 4      ELSE
: 1873      1993 4
: 1874      1994 4
: 1875      1995 4          ! Module is set. Print the routine name.
: 1876      1996 4          !
: 1877      1997 5          BEGIN
: 1878      1998 5          IF .PRINT_FLAG
: 1879      1999 5          THEN
: 1880      2000 6              BEGIN
: 1881      2001 6                  DBG$PRINT ( UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
: 1882      2002 6                  PRINT BIT_OFFSET(BIT_OFFSET);
: 1883      2003 6                  DBG$PRINT( UPLIT BYTE (%ASCIC ': '));
: 1884      2004 6                  DBG$NEWLINE();
: 1885      2005 6                  DBG$PRINT CONTROL (DBG$K_PRTSET LMARGIN, TAB);
: 1886      2006 6                  DBG$PRINT( UPLIT BYTE (%ASCIC 'stack call frame'), 0);
: 1887      2007 6                  TMP_BYTE_OFFSET = .ADDRESS - .FRAMEPTR;
: 1888      2008 6                  IF .TMP_BYTE_OFFSET NEQ 0
: 1889      2009 6                  THEN
: 1890      2010 6                      DBG$PRINT OFFSET(.TMP_BYTE_OFFSET);
: 1891      2011 6                      PRINT BIT_OFFSET(BIT_OFFSET);
: 1892      2012 6                      DBG$PRINT( UPLIT BYTE (%ASCIC ' for routine '));
: 1893      2013 6                      DBG$PRINT SYMBOL_PATHNAME(.RSTPTR);
: 1894      2014 6                      DBG$NEWLINE();
: 1895      2015 5                  END;
: 1896      2016 5
: 1897      2017 5          RETURN TRUE;
: 1898      2018 4          END;
: 1899      2019 4      END;
: 1900      2020 3
: 1901      2021 3
: 1902      2022 3
: 1903      2023 3          ! Not found yet. Get information for next stack frame.
: 1904      2024 3          !
: 1905      2025 3          PCVAL = .FRAMEPTR[SFS$ SAVE_PC];
: 1906      2026 3          FRAMEPTR = .FRAMEPTR[SFS$ SAVE_FP];
: 1907      2027 3
: 1908      2028 2      END;
: 1909      2029 2
: 1910      2030 2
```



```
: 1911      2031 2      ! Address was not found in a call frame.
: 1912      2032 2      !
: 1913      2033 2      RETURN FALSE;
: 1914      2034 2
: 1915      2035 1      END;
```

```
.PSECT DBGSPLIT,NOWRT, SHR, PIC,0

      4C 58 21 20 73 73 65 72 64 64 61 0B 0012C P.ABH: .ASCII <11>\address !XL\
      3C 01 00138 P.ABI: .ASCII <1>\<\
      21 03 0013A P.ABJ: .ASCII <3>\!UL\
      3E 30 20 2C 30 20 2C 07 0013E P.ABK: .ASCII <7>\, 0, 0>\
      20 3A 02 00146 P.ABL: .ASCII <2>\: \
73 20 6E 6F 20 6D 65 74 69 20 61 74 61 64 24 00149 P.ABM: .ASCII \$data item on stack in unknown module\
6E 77 6F 6E 6B 6E 75 20 6E 69 20 6B 63 61 74 00158
      65 6C 75 64 6F 6D 20 00167
      4C 58 21 20 73 73 65 72 64 64 61 0B 0016E P.ABN: .ASCII <11>\address !XL\
      3C 01 0017A P.ABO: .ASCII <1>\<\
      21 03 0017C P.ABP: .ASCII <3>\!UL\
      3E 30 20 2C 30 20 2C 07 00180 P.ABQ: .ASCII <7>\, 0, 0>\
      20 3A 02 00188 P.ABR: .ASCII <2>\: \
73 20 6E 6F 20 6D 65 74 69 20 61 74 61 64 20 0018B P.ABS: .ASCII \ data item on stack in module !AC\
20 65 6C 75 64 6F 6D 20 6E 69 20 6B 63 61 74 0019A
      43 41 21 001A9
73 20 74 65 73 20 65 6C 75 64 6F 6D 20 2C 0C 001AC P.ABT: .ASCII <12>\, module set\
73 20 74 6F 6E 20 65 6C 75 64 6F 6D 20 2C 10 001B9 P.ABU: .ASCII <16>\, module not set\
      74 65 001C8
      4C 58 21 20 73 73 65 72 64 64 61 0B 001CA P.ABV: .ASCII <11>\address !XL\
      3C 01 001D6 P.ABW: .ASCII <1>\<\
      21 03 001D8 P.ABX: .ASCII <3>\!UL\
      3E 30 20 2C 30 20 2C 07 001DC P.ABY: .ASCII <7>\, 0, 0>\
      20 3A 02 001E4 P.ABZ: .ASCII <2>\: \
73 20 6E 6F 20 6D 65 74 69 20 61 74 61 64 1E 001E7 P.ACA: .ASCII <30>\data item on stack in routine \
65 6E 69 74 75 6F 72 20 6E 69 20 6B 63 61 74 001F6
      20 00205
      3C 01 00206 P.ACB: .ASCII <1>\<\
      21 03 00208 P.ACC: .ASCII <3>\!UL\
      2C 07 0020C P.ACD: .ASCII <7>\, 0, 0>\
      4C 58 21 20 73 73 65 72 64 64 61 0B 00214 P.ACE: .ASCII <11>\address !XL\
      3C 01 00220 P.ACF: .ASCII <1>\<\
      21 03 00222 P.ACG: .ASCII <3>\!UL\
      3E 30 20 2C 30 20 2C 07 00226 P.ACH: .ASCII <7>\, 0, 0>\
      20 3A 02 0022E P.ACI: .ASCII <2>\: \
64 61 20 65 6D 61 72 66 20 6B 63 61 74 73 25 00231 P.ACJ: .ASCII \%stack frame address in unknown module\
77 6F 6E 6B 6E 75 20 6E 69 20 73 73 65 72 64 00240
      65 6C 75 64 6F 6D 20 6E 0024F
      4C 58 21 20 73 73 65 72 64 64 61 0B 00257 P.ACK: .ASCII <11>\address !XL\
      3C 01 00263 P.ACL: .ASCII <1>\<\
      21 03 00265 P.ACM: .ASCII <3>\!UL\
      3E 30 20 2C 30 20 2C 07 00269 P.ACN: .ASCII <7>\, 0, 0>\
      20 3A 02 00271 P.ACO: .ASCII <2>\: \
64 61 20 65 6D 61 72 66 20 6B 63 61 74 73 21 00274 P.ACP: .ASCII \!stack frame address in module !AC\
65 6C 75 64 6F 6D 20 6E 69 20 73 73 65 72 64 00283
      43 41 21 20 00292
      74 65 73 20 65 6C 75 64 6F 6D 20 2C 0C 00296 P.ACQ: .ASCII <12>\, module set\
```


73	20	74	6F	6E	20	65	6C	75	64	6F	6D	20	2C	10	002A3	P.ACR:	.ASCII	<16>\, module not set\
			4C	58	21	20	73	73	65	72	64	64	61	0B	002B2			
													3C	01	002B4	P.ACS:	.ASCII	<11>\address !XL\
											4C	55	21	03	002C0	P.ACT:	.ASCII	<1>\<\
							3E	30	20	2C	30	20	2C	07	002C2	P.ACU:	.ASCII	<3>\!UL\
												20	3A	02	002C6	P.ACV:	.ASCII	<7>\, 0, 0>\
61	72	66	20	6C	6C	61	63	20	6B	63	61	74	73	10	002CE	P.ACW:	.ASCII	<2>\: \
													65	6D	002D1	P.ACX:	.ASCII	<16>\stack call frame\
													3C	01	002E0			
											4C	55	21	03	002E2	P.ACY:	.ASCII	<1>\<\
							3E	30	20	2C	30	20	2C	07	002E4	P.ACZ:	.ASCII	<3>\!UL\
	20	65	6E	69	74	75	6F	72	20	72	6F	66	20	0D	002E8	P.ADA:	.ASCII	<7>\, 0, 0>\
															002F0	P.ADB:	.ASCII	<13>\ for routine \

```

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

```

OFFC 00000

```
.ENTRY DBGSSEARCH VAX CALL STACK, Save R2,R3,R4,-
      R5,R6,R7,R8,R9,R10,R11
```

	SE		30	C2	00002	SUBL2	#48, SP	: 1555
28	AE	08	AC	D0	00005	MOVL	P_SYMID, 40(SP)	: 1556
	59	0C	AC	D0	0000A	MOVL	P_BIT_OFFSET, R9	: 1590
20	AE	00000000G	00	7D	0000E	MOVQ	DBG\$RUNFRAME+60, SPVAL	: 1591
	54	00000000G	00	D0	00016	MOVL	DBG\$RUNFRAME+56, FRAMEPTR	: 1596
		28	BE	D4	0001D	CLRL	@40(SP)	: 1597
1C	AE		01	CE	00020	MNEGL	#1, BYTE_OFFSET	: 1598
	50	04	AC	D0	00024	MOVL	ADDR, RO	:
	58		60	D0	00028	MOVL	(R0), ADDRESS	:
	69	04	A0	D0	0002B	MOVL	4(R0), (R9)	: 1599
		24	AE	D5	0002F	TSTL	PCVAL	: 1610
			0A	13	00032	BEQL	2\$	:
	50	00000000G	00	9E	00034	MOVAB	DBG\$FINAL_HANDL, RO	: 1611
	50		64	D1	0003B	CMPL	(FRAMEPTR), RO	:
			03	12	0003E	BNEQ	3\$	:
		055F	31	00040	BRW	59\$		:
20	AE		54	D1	00043	CMPL	FRAMEPTR, SPVAL	: 1618
			03	1A	00047	BGTRU	5\$	:
		031F	31	00049	BRW	38\$		:
20	AE		58	D1	0004C	CMPL	ADDRESS, SPVAL	: 1624
			F7	1F	00050	BLSSU	4\$	:
	54		58	D1	00052	CMPL	ADDRESS, FRAMEPTR	: 1625
			F2	1E	00055	BGEQU	4\$	:
		10	AC	DD	00057	PUSHL	PRINT_FLAG	: 1628
		28	AE	DD	0005A	PUSHL	PCVAL	:
0000V	CF		02	FB	0005D	CALLS	#2, FIND_ROUTINE	:
	52		50	D0	00062	MOVL	R0, RSTPTR	:
			68	12	00065	BNEQ	7\$	: 1629
	6D	10	AC	E9	00067	BLBC	PRINT_FLAG, 8\$	: 1632
			58	DD	0006B	PUSHL	ADDRESS	: 1635
		00000000'	EF	9F	0006D	PUSHAB	P.ABH	:
00000000G	00		02	FB	00073	CALLS	#2, DBG\$PRINT	:
			69	D5	0007A	TSTL	(R9)	: 1636
			29	13	0007C	BEQL	6\$	:
		00000000'	EF	9F	0007E	PUSHAB	P.ABI	:
00000000G	00		01	FB	00084	CALLS	#1, DBG\$PRINT	:
			69	DD	0008B	PUSHL	(R9)	:

00000000G	00	00000000'	EF	9F	0008D	PUSHAB	P.ABJ	:
			02	FB	00093	CALLS	#2, DBG\$PRINT	:
00000000G	00	00000000'	EF	9F	0009A	PUSHAB	P.ABK	:
			01	FB	000A0	CALLS	#1, DBG\$PRINT	:
00000000G	00	00000000'	EF	9F	000A7	PUSHAB	P.ABL	1637
			01	FB	000AD	CALLS	#1, DBG\$PRINT	:
00000000G	00		00	FB	000B4	CALLS	#0, DBG\$NEWLINE	1638
			04	DD	000BB	PUSHL	#4	1639
			01	DD	000BD	PUSHL	#1	:
00000000G	00		02	FB	000BF	CALLS	#2, DBG\$PRINT_CONTROL	:
		00000000'	EF	9F	000C6	PUSHAB	P.ABM	1640
			0099	31	000CC	BRW	12\$	:
	01	14	A2	91	000CF	CMPB	20(RSTPTR), #1	1647
			03	13	000D3	BEQL	8\$	:
			009A	31	000D5	BRW	13\$	:
	03	10	AC	E8	000D8	BLBS	PRINT_FLAG, 9\$	1656
			04B3	31	000DC	BRW	57\$	:
			58	DD	000DF	PUSHL	ADDRESS	1659
		00000000'	EF	9F	000E1	PUSHAB	P.ABN	:
00000000G	00		02	FB	000E7	CALLS	#2, DBG\$PRINT	:
			69	D5	000EE	TSTL	(R9)	1660
			29	13	000F0	BEQL	10\$	:
		00000000'	EF	9F	000F2	PUSHAB	P.ABO	:
00000000G	00		01	FB	000F8	CALLS	#1, DBG\$PRINT	:
			69	DD	000FF	PUSHL	(R9)	:
		00000000'	EF	9F	00101	PUSHAB	P.ABP	:
00000000G	00		02	FB	00107	CALLS	#2, DBG\$PRINT	:
		00000000'	EF	9F	0010E	PUSHAB	P.ABQ	:
00000000G	00		01	FB	00114	CALLS	#1, DBG\$PRINT	:
		00000000'	EF	9F	0011B	PUSHAB	P.ABR	1661
00000000G	00		01	FB	00121	CALLS	#1, DBG\$PRINT	:
00000000G	00		00	FB	00128	CALLS	#0, DBG\$NEWLINE	1662
			04	DD	0012F	PUSHL	#4	1663
			01	DD	00131	PUSHL	#1	:
00000000G	00		02	FB	00133	CALLS	#2, DBG\$PRINT_CONTROL	:
		2C	AE	9F	0013A	PUSHAB	SYMBOL_NAME	1664
			52	DD	0013D	PUSHL	RSTPTR	:
00000000G	00		02	FB	0013F	CALLS	#2, DBG\$STA_SYMNAME	:
		2C	AE	DD	00146	PUSHL	SYMBOL_NAME	1665
		00000000'	EF	9F	00149	PUSHAB	P.ABS	:
00000000G	00		02	FB	0014F	CALLS	#2, DBG\$PRINT	:
	08	28	A2	E9	00156	BLBC	40(RSTPTR), 11\$	1666
		00000000'	EF	9F	0015A	PUSHAB	P.ABT	1668
			06	11	00160	BRB	12\$	:
		00000000'	EF	9F	00162	PUSHAB	P.ABU	1670
00000000G	00		01	FB	00168	CALLS	#1, DBG\$PRINT	:
			0419	31	0016F	BRW	56\$	1672
	55		52	DD	00172	MOVL	RSTPTR, MODRSTPTR	1681
	01	14	A5	91	00175	CMPB	20(MODRSTPTR), #1	1682
			06	13	00179	BEQL	15\$	:
	55	10	A5	DD	0017B	MOVL	16(MODRSTPTR), MODRSTPTR	1683
			F4	11	0017F	BRB	14\$	:
	56	08	A5	DD	00181	MOVL	8(MODRSTPTR), TMP_SYMID	1684
			6F	13	00185	BEQL	22\$	1690
			0D	DD	00187	PUSHL	#13	1699
			56	DD	00189	PUSHL	TMP_SYMID	:
0000V	CF		02	FB	0018B	CALLS	#2, IS_SYMBOL_BOUND_TO_REG	:

		57		50	D0	00190	MOVL	R0, VALSPEC	1700
		5B		54	D0	00193	MOVL	FRAMEPTR, TMPPTR	1701
				57	D5	00196	TSTL	VALSPEC	1704
				1B	12	00198	BNEQ	18\$	
				0E	DD	0019A	PUSHL	#14	
				56	DD	0019C	PUSHL	TMP_SYMID	
	0000V	CF		02	FB	0019E	CALLS	#2, IS_SYMBOL_BOUND_TO_REG	
		57		50	D0	001A3	MOVL	R0, VALSPEC	
				0D	13	001A6	BEQL	18\$	1715
			01	A7	D5	001A8	TSTL	1(VALSPEC)	1718
				04	18	001AB	BGEQ	17\$	
				57	D4	001AD	CLRL	VALSPEC	1720
				04	11	001AF	BRB	18\$	
		5B		AE	D0	001B1	MOVL	SPVAL, TMPPTR	1722
			20	57	D5	001B5	TSTL	VALSPEC	1728
				32	13	001B7	BEQL	21\$	
		67		02	E0	001B9	BBS	#2, (VALSPEC), 21\$	1736
	2E	67		03	E1	001BD	BBC	#3, (VALSPEC), 21\$	1737
	2A	52		A6	D1	001C1	CMPL	16(TMP_SYMID), RSTPTR	1740
			10	24	12	001C5	BNEQ	21\$	
	50	5B		A7	C1	001C7	ADDL3	1(VALSPEC), TMPPTR, R0	1749
18	AE	58		50	C3	001CC	SUBL3	R0, ADDRESS, TMP_BYTE_OFFSET	
		AE		AE	D1	001D1	CMPL	TMP_BYTE_OFFSET, BYTE_OFFSET	1750
	1C			05	18	001D6	BGEQ	19\$	
			18	AE	D5	001D8	TSTL	TMP_BYTE_OFFSET	1751
				05	18	001DB	BGEQ	20\$	
			1C	AE	D5	001DD	TSTL	BYTE_OFFSET	1752
				09	18	001E0	BGEQ	21\$	
	28	BE		56	D0	001E2	MOVL	TMP_SYMID, @40(SP)	1755
	1C	AE		AE	D0	001E6	MOVL	TMP_BYTE_OFFSET, BYTE_OFFSET	1756
			1C	AE	D5	001EB	TSTL	BYTE_OFFSET	1765
				06	13	001EE	BEQL	22\$	
		56		A6	D0	001F0	MOVL	8(TMP_SYMID), TMP_SYMID	1772
				8F	11	001F4	BRB	16\$	1690
			10	AE	D4	001F6	CLRL	COUNT	1782
			28	BE	D5	001F9	TSTL	@40(SP)	1783
				6D	13	001FC	BEQL	27\$	
	04	AE		A2	D0	001FE	MOVL	24(RSTPTR), STARTADDR	1786
		6E		A2	D0	00203	MOVL	28(RSTPTR), ENDADDR	1787
	14	AE	00000000G	00	D0	00207	MOVL	DBG\$RUNFRAME+64, PCVAL2	1788
		5A	00000000G	00	D0	0020F	MOVL	DBG\$RUNFRAME+56, FRAMEPTR2	1789
		54		5A	D1	00216	CMPL	FRAMEPTR2, FRAMEPTR	1790
				50	18	00219	BGEQ	27\$	
	04	AE		AE	D1	0021B	CMPL	PCVAL2, STARTADDR	1792
				3E	1F	00220	BLSSU	26\$	
		6E		AE	D1	00222	CMPL	PCVAL2, ENDADDR	
				38	1A	00226	BGTRU	26\$	
	0C	AE		01	D0	00228	MOVL	#1, FRAME_FOUND_FLAG	1804
		53		A2	D0	0022C	MOVL	32(RSTPTR), SATPTR	1805
				27	13	00230	BEQL	25\$	1821
		53		63	D0	00232	MOVL	(SATPTR), SATPTR	1823
				22	13	00235	BEQL	25\$	1834
	14	AE		A3	D1	00237	CMPL	4(SATPTR), PCVAL2	1835
				1B	1A	0023C	BGTRU	25\$	
		50		A3	D0	0023E	MOVL	12(SATPTR), NEW_RSTPTR	1842
	04	A3		AE	D1	00242	CMPL	PCVAL2, 4(SATPTR)	1843
				E9	1F	00247	BLSSU	24\$	

08	A3	14	AE	D1	00249	CMPL	PCVAL2, 8(SATPTR)	1844
			E2	1A	0024E	BGTRU	24\$	
	02	14	A0	91	00250	CMPB	20(NEW_RSTPTR), #2	1845
			DC	12	00254	BNEQ	24\$	
		0C	AE	D4	00256	CLRL	FRAME_FOUND_FLAG	1848
	03	0C	AE	E9	00259	BLBC	FRAME_FOUND_FLAG, 26\$	1860
		10	AE	D6	0025D	INCL	COUNT	1862
14	AE	10	AA	D0	00260	MOVL	16(FRAMEPTR2), PCVAL2	1867
	5A	0C	AA	D0	00265	MOVL	12(FRAMEPTR2), FRAMEPTR2	1868
			AB	11	00269	BRB	23\$	1790
		10	AE	D5	0026B	TSTL	COUNT	1874
			11	15	0026E	BLEQ	28\$	
		10	AE	DD	00270	PUSHL	COUNT	1876
		2C	BE	DD	00273	PUSHL	@44(SP)	
00000000G	00		02	FB	00276	CALLS	#2, DBG\$BUILD_INVOC_RST	
28	BE		50	D0	0027D	MOVL	R0, @40(SP)	
	03	10	AC	E8	00281	BLBS	PRINT_FLAG, 29\$	1882
			00C8	31	00285	BRW	34\$	
			58	DD	00288	PUSHL	ADDRESS	1885
		00000000'	EF	9F	0028A	PUSHAB	P.ABV	
00000000G	00		02	FB	00290	CALLS	#2, DBG\$PRINT	
			5B	D4	00297	CLRL	R11	1886
			69	D5	00299	TSTL	(R9)	
			2B	13	0029B	BEQL	30\$	
		00000000'	5B	D6	0029D	INCL	R11	
00000000G	00		EF	9F	0029F	PUSHAB	P.ABW	
			01	FB	002A5	CALLS	#1, DBG\$PRINT	
		00000000'	69	DD	002AC	PUSHL	(R9)	
00000000G	00		EF	9F	002AE	PUSHAB	P.ABX	
		00000000'	02	FB	002B4	CALLS	#2, DBG\$PRINT	
00000000G	00		EF	9F	002BB	PUSHAB	P.ABY	
		00000000'	01	FB	002C1	CALLS	#1, DBG\$PRINT	
00000000G	00		EF	9F	002C8	PUSHAB	P.ABZ	1887
00000000G	00		01	FB	002CE	CALLS	#1, DBG\$PRINT	
00000000G	00		00	FB	002D5	CALLS	#0, DBG\$NEWLINE	1888
			04	DD	002DC	PUSHL	#4	1889
			01	DD	002DE	PUSHL	#1	
00000000G	00		02	FB	002E0	CALLS	#2, DBG\$PRINT_CONTROL	
		28	BE	D5	002E7	TSTL	@40(SP)	1890
			18	12	002EA	BNEQ	31\$	
		00000000'	EF	9F	002EC	PUSHAB	P.ACA	1893
00000000G	00		01	FB	002F2	CALLS	#1, DBG\$PRINT	
			52	DD	002F9	PUSHL	RSTPTR	1894
00000000G	00		01	FB	002FB	CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME	
			45	11	00302	BRB	33\$	1890
00000000G	00	28	BE	DD	00304	PUSHL	@40(SP)	1899
			01	FB	00307	CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME	
		1C	AE	D5	0030E	TSTL	BYTE_OFFSET	1900
			0A	13	00311	BEQL	32\$	
		1C	AE	DD	00313	PUSHL	BYTE_OFFSET	1902
00000000G	00		01	FB	00316	CALLS	#1, DBG\$PRINT_OFFSET	
	29		5B	E9	0031D	BLBC	R11, 33\$	1903
		00000000'	EF	9F	00320	PUSHAB	P.ACB	
00000000G	00		01	FB	00326	CALLS	#1, DBG\$PRINT	
			69	DD	0032D	PUSHL	(R9)	
		00000000'	EF	9F	0032F	PUSHAB	P.ACC	
00000000G	00		02	FB	00335	CALLS	#2, DBG\$PRINT	

			00000000G	00	00000000'	EF	9F	0033C	PUSHAB	P.ACD		
			00000000G	00		01	FB	00342	CALLS	#1, DBG\$PRINT		
						00	FB	00349	CALLS	#0, DBG\$NEWLINE	1905	
					1C	AE	D5	00350	TSTL	BYTE_OFFSET	1908	
						03	18	00353	BGEQ	35\$		
					28	BE	D4	00355	CLRL	@40(SP)		
					28	BE	D5	00358	TSTL	@40(SP)	1909	
						03	12	0035B	BNEQ	36\$		
					1C	AE	D4	0035D	CLRL	BYTE_OFFSET		
				50	1C	AE	D0	00360	MOVL	BYTE_OFFSET, R0	1910	
				79		9940	7E	00364	MOVAQ	@(R9)+[R0], -(R9)		
						0227	31	00368	BRW	57\$	1911	
			08	AE	06	A4	B0	0036B	MOVW	6(FRAMEPTR), REGMASK	1920	
			20	AE	14	A4	9E	00370	MOVAB	20(R4), SPVAL	1921	
						50	D4	00375	CLRL	I	1922	
			04	08	AE	50	E1	00377	BBC	I, REGMASK, 40\$	1923	
			20	AE		04	C0	0037C	ADDL2	#4, SPVAL		
			F3	50		0B	F3	00380	AOBLEQ	#11, I, 39\$		
50	07		A4	02		06	EF	00384	EXTZV	#6, #2, 7(FRAMEPTR), R0	1924	
				20	AE	50	C0	0038A	ADDL2	R0, SPVAL		
			0E	07	A4	05	E1	0038E	BBC	#5, 7(FRAMEPTR), 41\$	1925	
				50		20	BE	00393	MOVL	@SPVAL, R0	1927	
			51	20	AE	04	C1	00397	ADDL3	#4, SPVAL, R1		
				20	AE	6140	DE	0039C	MOVAL	(R1)[R0], SPVAL		
					54	58	D1	003A1	CMPL	ADDRESS, FRAMEPTR	1933	
						03	1E	003A4	BGEQU	43\$		
						01ED	31	003A6	BRW	58\$		
				20	AE	58	D1	003A9	CMPL	ADDRESS, SPVAL	1934	
						F7	1E	003AD	BGEQU	42\$		
					10	AC	DD	003AF	PUSHL	PRINT_FLAG	1941	
					28	AE	DD	003B2	PUSHL	PCVAL		
			0000V	CF		02	FB	003B5	CALLS	#2, FIND ROUTINE		
				52		50	D0	003BA	MOVL	R0, RSTPTR		
						68	12	003BD	BNEQ	45\$	1942	
				A5	10	AC	E9	003BF	BLBC	PRINT_FLAG, 37\$	1945	
						58	DD	003C3	PUSHL	ADDRESS	1948	
			00000000G	00	00000000'	EF	9F	003C5	PUSHAB	P.ACE		
						02	FB	003CB	CALLS	#2, DBG\$PRINT		
						69	D5	003D2	TSTL	(R9)	1949	
						29	13	003D4	BEQL	44\$		
			00000000G	00	00000000'	EF	9F	003D6	PUSHAB	P.ACF		
						01	FB	003DC	CALLS	#1, DBG\$PRINT		
						69	DD	003E3	PUSHL	(R9)		
			00000000G	00	00000000'	EF	9F	003E5	PUSHAB	P.ACG		
						02	FB	003EB	CALLS	#2, DBG\$PRINT		
			00000000G	00	00000000'	EF	9F	003F2	PUSHAB	P.ACH		
						01	FB	003F8	CALLS	#1, DBG\$PRINT		
			00000000G	00	00000000'	EF	9F	003FF	PUSHAB	P.ACI	1950	
						01	FB	00405	CALLS	#1, DBG\$PRINT		
			00000000G	00		00	FB	0040C	CALLS	#0, DBG\$NEWLINE	1951	
						04	DD	00413	PUSHL	#4	1952	
						01	DD	00415	PUSHL	#1		
			00000000G	00		02	FB	00417	CALLS	#2, DBG\$PRINT_CONTROL		
					00000000'	EF	9F	0041E	PUSHAB	P.ACJ	1953	
						FD41	31	00424	BRW	12\$		
				01	14	A2	91	00427	CMPB	20(RSTPTR), #1	1960	
						03	13	0042B	BEQL	46\$		

		0093	31	0042D	BRW	51\$	
	03	10	AC	EB 00430	46\$:	BLBS	PRINT_FLAG, 47\$
			015B	31 00434		BRW	57\$
			58	DD 00437	47\$:	PUSHL	ADDRESS
00000000G	00	00000000'	EF	9F 00439		PUSHAB	P.ACK
			02	FB 0043F		CALLS	#2, DBG\$PRINT
			69	D5 00446		TSTL	(R9)
			29	13 00448		BEQL	48\$
00000000G	00	00000000'	EF	9F 0044A		PUSHAB	P.ACL
			01	FB 00450		CALLS	#1, DBG\$PRINT
			69	DD 00457		PUSHL	(R9)
00000000G	00	00000000'	EF	9F 00459		PUSHAB	P.ACM
			02	FB 0045F		CALLS	#2, DBG\$PRINT
00000000G	00	00000000'	EF	9F 00466		PUSHAB	P.ACN
			01	FB 0046C		CALLS	#1, DBG\$PRINT
00000000G	00	00000000'	EF	9F 00473	48\$:	PUSHAB	P.ACO
			01	FB 00479		CALLS	#1, DBG\$PRINT
00000000G	00		00	FB 00480		CALLS	#0, DBG\$NEWLINE
			04	DD 00487		PUSHL	#4
			01	DD 00489		PUSHL	#1
00000000G	00		02	FB 0048B		CALLS	#2, DBG\$PRINT_CONTROL
		2C	AE	9F 00492		PUSHAB	SYMBOL_NAME
			52	DD 00495		PUSHL	RSTPTR
00000000G	00		02	FB 00497		CALLS	#2, DBG\$STA_SYMNAME
		2C	AE	DD 0049E		PUSHL	SYMBOL_NAME
		00000000'	EF	9F 004A1		PUSHAB	P.ACP
00000000G	00		02	FB 004A7		CALLS	#2, DBG\$PRINT
	08	28	A2	E9 004AE		BLBC	40(RSTPTR), 49\$
		00000000'	EF	9F 004B2		PUSHAB	P.ACQ
			06	11 004B8		BRB	50\$
		00000000'	EF	9F 004BA	49\$:	PUSHAB	P.ACR
			FC A5	31 004C0	50\$:	BRW	12\$
	03	10	AC	EB 004C3	51\$:	BLBS	PRINT_FLAG, 52\$
			00C8	31 004C7		BRW	57\$
			58	DD 004CA	52\$:	PUSHL	ADDRESS
00000000G	00	00000000'	EF	9F 004CC		PUSHAB	P.ACS
			02	FB 004D2		CALLS	#2, DBG\$PRINT
			5B	D4 004D9		CLRL	R11
			69	D5 004DB		TSTL	(R9)
			2B	13 004DD		BEQL	53\$
			5B	D6 004DF		INCL	R11
00000000G	00	00000000'	EF	9F 004E1		PUSHAB	P.ACT
			01	FB 004E7		CALLS	#1, DBG\$PRINT
			69	DD 004EE		PUSHL	(R9)
00000000G	00	00000000'	EF	9F 004F0		PUSHAB	P.ACU
			02	FB 004F6		CALLS	#2, DBG\$PRINT
00000000G	00	00000000'	EF	9F 004FD		PUSHAB	P.ACV
			01	FB 00503		CALLS	#1, DBG\$PRINT
00000000G	00	00000000'	EF	9F 0050A	53\$:	PUSHAB	P.ACW
			01	FB 00510		CALLS	#1, DBG\$PRINT
00000000G	00		00	FB 00517		CALLS	#0, DBG\$NEWLINE
			04	DD 0051E		PUSHL	#4
			01	DD 00520		PUSHL	#1
00000000G	00		02	FB 00522		CALLS	#2, DBG\$PRINT_CONTROL
			7E	D4 00529		CLRL	-(SP)
		00000000'	EF	9F 0052B		PUSHAB	P.ACX
00000000G	00		02	FB 00531		CALLS	#2, DBG\$PRINT

18	AE	58	54	C3	00538	SUBL3	FRAMEPTR, ADDRESS, TMP_BYTE_OFFSET	:	2007
			0A	13	0053D	BEQL	54\$	:	2008
		18	AE	DD	0053F	PUSHL	TMP_BYTE_OFFSET	:	2010
00000000G	00		01	FB	00542	CALLS	#1, -DBG\$PRINT_OFFSET	:	
	29		5B	E9	00549	BLBC	R11, 55\$	:	2011
		00000000'	EF	9F	0054C	PUSHAB	P.ACZ	:	
00000000G	00		01	FB	00552	CALLS	#1, DBG\$PRINT	:	
		00000000'	69	DD	00559	PUSHL	(R9)	:	
00000000G	00		EF	9F	0055B	PUSHAB	P.ACZ	:	
		00000000'	02	FB	00561	CALLS	#2, DBG\$PRINT	:	
00000000G	00		EF	9F	00568	PUSHAB	P.ADA	:	
		00000000'	01	FB	0056E	CALLS	#1, DBG\$PRINT	:	
00000000G	00		EF	9F	00575	PUSHAB	P.ADB	:	2012
			01	FB	0057B	CALLS	#1, DBG\$PRINT	:	
00000000G	00		52	DD	00582	PUSHL	RS1PTR	:	2013
00000000G	00		01	FB	00584	CALLS	#1, DBG\$PRINT SYMBOL_PATHNAME	:	
00000000G	00		00	FB	0058B	CALLS	#0, DBG\$NEWLINE	:	2014
	50		01	D0	00592	MOVL	#1, R0	:	2017
			04	00595	RET			:	1997
24	AE	10	A4	D0	00596	MOVL	16(FRAMEPTR), PCVAL	:	2025
	54	0C	A4	D0	0059B	MOVL	12(FRAMEPTR), FRAMEPTR	:	2026
			FA8D	31	0059F	BRW	1\$	:	1604
			50	D4	005A2	CLRL	R0	:	2033
			04	005A4	RET			:	2035

; Routine Size: 1445 bytes, Routine Base: DBG\$CODE + 0753


```
: 1917 2036 1 GLOBAL ROUTINE DBG$SYMBOLIZE_REG(ADDR, P_SYMID, P_BIT_OFFSET, PRINT_FLAG): =
: 1918 2037 1
: 1919 2038 1 FUNCTION
: 1920 2039 1     This routine searches through all set modules in the module chain,
: 1921 2040 1     looking for those symbols in the DST that are bound to the register
: 1922 2041 1     REGNUM. Once found, they are printed. If none are found, a message
: 1923 2042 1     to that effect is output.
: 1924 2043 1
: 1925 2044 1 INPUTS
: 1926 2045 1     ADDR          - Address descriptor.
: 1927 2046 1
: 1928 2047 1     P_SYMID        - The address of a longword to receive the best
: 1929 2048 1                   possible symbolization of the register address.
: 1930 2049 1
: 1931 2050 1     P_BIT_OFFSET   - The bit offset off of SYMID.
: 1932 2051 1
: 1933 2052 1 OUTPUTS
: 1934 2053 1     On success, prints those symbols (full pathname) that are bound to
: 1935 2054 1     the register. Also returns the best (first) symbolization.
: 1936 2055 1
: 1937 2056 1     On not finding any symbols bound to the register, a message to that
: 1938 2057 1     effect is output.
: 1939 2058 1
: 1940 2059 1
: 1941 2060 1 BEGIN
: 1942 2061 2
: 1943 2062 2 MAP
: 1944 2063 2     ADDR          : REF DBG$ADDRESS_DESC;
: 1945 2064 2
: 1946 2065 2 BIND
: 1947 2066 2     SYMID          = .P_SYMID: REF RST$ENTRY,      ! Best symbolization.
: 1948 2067 2     BIT_OFFSET   = .P_BIT_OFFSET;              ! Bit offset.
: 1949 2068 2
: 1950 2069 2 LOCAL
: 1951 2070 2     ADDRESS,          ! Byte address
: 1952 2071 2     BYTE_OFFSET,    ! Byte offset
: 1953 2072 2     REGNUM,        ! Register number.
: 1954 2073 2     SYMBOL_FOUND,   ! One or more symbols found.
: 1955 2074 2     REGNAME         : REF VECTOR[, BYTE],      ! ASCII register name.
: 1956 2075 2     MODRSTPTR      : REF RST$ENTRY,          ! Ptr to current module
: 1957 2076 2                                     ! RST entry.
: 1958 2077 2     RSTPTR          : REF RST$ENTRY,          ! Unique symbol ID.
: 1959 2078 2     VALSPEC        : REF DST$VAL_SPEC,        ! Ptr to DST record's
: 1960 2079 2                                     ! value spec.
: 1961 2080 2     BLITRLR        : REF DST$BLI_TRAILER1,    ! Ptr to Bliss DST record
: 1962 2081 2                                     ! trailer.
: 1963 2082 2     BLIVALSPEC      : BLOCK[8, BYTE]          ! Value spec buffer for
: 1964 2083 2                                     ! Bliss special cases DST.
: 1965 2084 2                                     !
: 1966 2085 2                                     !
: 1967 2086 2
: 1968 2087 2
: 1969 2088 2 ADDRESS = .ADDR[DBG$L_ADDRESS_BYTE_ADDR];
: 1970 2089 2 BIT_OFFSET = .ADDR[DBG$L_ADDRESS_BIT_OFFSET];
: 1971 2090 2
: 1972 2091 2
: 1973 2092 2 ! See if the address is a register address.
```



```

: 1974      2093      2
: 1975      2094      2
: 1976      2095      2
: 1977      2096      2
: 1978      2097      2
: 1979      2098      2
: 1980      2099      2
: 1981      2100      2
: 1982      2101      2
: 1983      2102      2
: 1984      2103      2
: 1985      2104      2
: 1986      2105      2
: 1987      2106      2
: 1988      2107      2
: 1989      2108      2
: 1990      2109      2
: 1991      2110      2
: 1992      2111      2
: 1993      2112      2
: 1994      2113      2
: 1995      2114      2
: 1996      2115      2
: 1997      2116      2
: 1998      2117      2
: 1999      2118      2
: 2000      2119      2
: 2001      2120      2
: 2002      2121      2
: 2003      2122      2
: 2004      2123      2
: 2005      2124      2
: 2006      2125      2
: 2007      2126      2
: 2008      2127      2
: 2009      2128      2
: 2010      2129      2
: 2011      2130      2
: 2012      2131      2
: 2013      2132      2
: 2014      2133      2
: 2015      2134      2
: 2016      2135      2
: 2017      2136      2
: 2018      2137      2
: 2019      2138      2
: 2020      2139      2
: 2021      2140      2
: 2022      2141      2
: 2023      2142      2
: 2024      2143      2
: 2025      2144      2
: 2026      2145      2
: 2027      2146      2
: 2028      2147      2
: 2029      2148      2
: 2030      2149      2

!
IF (.ADDRESS LSSA DBG$REG_VALUES[0]) OR
   (.ADDRESS GEQA DBG$REG_VALUES[17])
THEN
    RETURN FALSE;

! Set up to look through all set modules for symbols bound to REGNUM.
!
IF (MODRSTPTR = .RST$START_ADDR) EQL 0
THEN
    SIGNAL (DBG$_NOLOCALS);

! Calculate register number and offset from register address.
!
REGNUM = (.ADDRESS - DBG$REG_VALUES[0]) / 4;
BYTE_OFFSET = .ADDRESS - DBG$REG_VALUES[.REGNUM];

! Initialize.
!
SYMID = 0;
SYMBOL_FOUND = FALSE;

! Print register name.
!
IF .PRINT_FLAG
THEN
    BEGIN
        DBG$PRINT (UPLIT BYTE (%ASCIC 'address '), 0);
        PRINT_REGNAME (.ADDRESS);
        PRINT_BIT_OFFSET(BIT_OFFSET);
        DBG$PRINT (UPLIT BYTE (%ASCIC ': '), 0);
        DBG$NEWLINE();
        DBG$PRINT_CONTROL (DBG$K_PRTSET_LMARGIN, TAB);
    END;

! Loop through all set modules.
!
WHILE .MODRSTPTR NEQ 0 DO
    BEGIN

        $ABORT_ON_CONTROL_Y;

        ! Check to see if the module is set.
        !
        IF .MODRSTPTR[RST$V_MODSET]
        THEN
            BEGIN

                ! The module was set, so get the first RST entry in the chain.
```



```
2031 2150 4      !
2032 2151 4      ! RSTPTR = .MODRSTPTR[RST$L_SYMCHNPTR];
2033 2152 4
2034 2153 4
2035 2154 4      ! Loop until the module's symbol chain is exhausted.
2036 2155 4
2037 2156 4      WHILE .RSTPTR NEQ 0 DO
2038 2157 5          BEGIN
2039 2158 5
2040 2159 5
2041 2160 5      ! See if symbol is bound to the register in question.
2042 2161 5
2043 2162 5      VALSPEC = IS_SYMBOL_BOUND_TO_REG (.RSTPTR, .REGNUM);
2044 2163 5
2045 2164 5
2046 2165 5      ! If it is, then use the value spec. to print the
2047 2166 5      ! symbolization.
2048 2167 5
2049 2168 5      IF .VALSPEC NEQ 0
2050 2169 5      THEN
2051 2170 5          IF .VALSPEC[DST$V_VS_VALKIND] EQL DST$K_VALKIND_REG
2052 2171 5          THEN
2053 2172 6              BEGIN
2054 2173 6                  IF .PRINT_FLAG
2055 2174 6                  THEN
2056 2175 7                      BEGIN
2057 2176 7                          DBG$PRINT SYMBOL PATHNAME (.RSTPTR);
2058 2177 7                          IF .BYTE_OFFSET NEQ 0
2059 2178 7                          THEN
2060 2179 7                              DBG$PRINT OFFSET (.BYTE_OFFSET);
2061 2180 7                              PRINT BIT_OFFSET(BIT_OFFSET);
2062 2181 7                              DBG$NEWLINE();
2063 2182 6                          END;
2064 2183 6
2065 2184 6
2066 2185 6      ! The best symbolization is the first symbol found that
2067 2186 6      ! is bound to this register. (An arbitrary decision.)
2068 2187 6      ! (More work needs to be done to associate the RST with
2069 2188 6      ! the scope)
2070 2189 6
2071 2190 6      IF NOT .SYMBOL_FOUND
2072 2191 6      THEN
2073 2192 6          SYMID = .RSTPTR;
2074 2193 6          SYMBOL_FOUND = TRUE;
2075 2194 5      END;
2076 2195 5
2077 2196 5
2078 2197 5      ! Get the next symid in the symbol chain.
2079 2198 5
2080 2199 5      RSTPTR = .RSTPTR[RST$L_SYMCHNPTR];
2081 2200 4      END;
2082 2201 4
2083 2202 5      END;
2084 2203 5
2085 2204 5
2086 2205 5      ! Get a pointer to the next module in the chain.
2087 2206 5
```



```
2088 2207 3      MODRSTPTR = .MODRSTPTR[RST$L_NXTMODPTR];
2089 2208      END;
2090 2209
2091 2210
2092 2211      ! If NOT .SYMBOL_FOUND, then print out a message saying that there are no
2093 2212      ! symbols bound to this register address.
2094 2213
2095 2214      IF .PRINT_FLAG
2096 2215      THEN
2097 2216          BEGIN
2098 2217              IF NOT .SYMBOL_FOUND
2099 2218              THEN
2100 2219                  BEGIN
2101 2220                      DBG$PRINT (UPLIT BYTE (%ASCII 'no symbols are bound to '), 0);
2102 2221                      PRINT_REGNAME(.ADDRESS);
2103 2222                      PRINT_BIT_OFFSET(BIT_OFFSET);
2104 2223                      DBG$NEWLINE();
2105 2224                  END;
2106 2225          END;
2107 2226
2108 2227
2109 2228      ! End of routine.
2110 2229
2111 2230      IF .SYMID EQL 0 THEN BYTE_OFFSET = 0;
2112 2231      BIT_OFFSET = .BYTE_OFFSET * 8 + .BIT_OFFSET;
2113 2232      RETURN TRUE;
2114 2233
2115 2234      END;
```

```
                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
20 73 73 65 72 64 64 61 08 002FE P.ADC: .ASCII <8>\address \
                                3C 01 00307 P.ADD: .ASCII <1>\<\
                                21 03 00309 P.ADE: .ASCII <3>\!UL\
3E 30 20 2C 4C 55 2C 07 0030D P.ADF: .ASCII <7>\, 0, 0>\
                                3A 03 00315 P.ADG: .ASCII <3>\: \
                                3C 01 00319 P.ADH: .ASCII <1>\<\
                                4C 55 21 03 0031B P.ADI: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 0031F P.ADJ: .ASCII <7>\, 0, 0>\
65 72 61 20 73 6C 6F 62 6D 79 73 20 6F 6E 18 00327 P.ADK: .ASCII <24>\no symbols are bound to \
20 6F 74 20 64 6E 75 6F 62 20 00336
                                3C 01 00340 P.ADL: .ASCII <1>\<\
                                4C 55 21 03 00342 P.ADM: .ASCII <3>\!UL\
3E 30 20 2C 30 20 2C 07 00346 P.ADN: .ASCII <7>\, 0, 0>\
```

```
                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
                                OFFC 00000
                                .ENTRY  DBG$SYMBOLIZE_REG, Save R2,R3,R4,R5,R6,R7,- ; 2036
                                5B 00000000G 00 9E 00002 MOVAB DBG$PRINT, R11
                                5A 00000000' EF 9E 00009 MOVAB P.ADC, R10
                                5E          08 C2 00010 SUBL2 #8, SP
                                56          0C AC D0 00013 MOVL P_BIT_OFFSET, R6 ; 2068
```


50	04	AC	DO	00017	MOVL	ADDR, R0	2088	
52		60	DO	0001B	MOVL	(R0), ADDRESS		
66	04	A0	DO	0001E	MOVL	4(R0), (R6)	2089	
50	00000000G	0C	9E	00022	MOVAB	DBG\$REG_VALUES, R0	2094	
50		52	D1	00029	CMPL	ADDRESS, R0		
		03	1E	0002C	BGEQU	2\$		
		0153	31	0002E	BRW	19\$		
50	00000000G	00	9E	00031	MOVAB	DBG\$REG_VALUES+68, R0	2095	
50		52	D1	00038	CMPL	ADDRESS, R0		
		F1	1E	0003B	BGEQU	1\$		
53	00000000G	00	DO	0003D	MOVL	RST\$START_ADDR, MODRSTPTR	2102	
	00028053	0D	12	00044	BNEQ	3\$		
		8F	DD	00046	PUSHL	#163923	2104	
00000000G		01	FB	0004C	CALLS	#1, LIB\$SIGNAL		
50	00000000G	00	9E	00053	MOVAB	DBG\$REG_VALUES, R0	2109	
52		50	C3	0005A	SUBL3	R0, ADDRESS, R0		
50		04	C7	0005E	DIVL3	#4, R0, REGNUM		
50	00000000G	0045	DE	00062	MOVAL	DBG\$REG_VALUES[REGNUM], R0	2110	
52		50	C3	0006A	SUBL3	R0, ADDRESS, BYTE_OFFSET		
	08	BC	D4	0006E	CLRL	@P SYMID	2115	
		58	D4	00071	CLRL	SYMBOL FOUND	2116	
40	10	AC	E9	00073	BLBC	PRINT_FLAG, 5\$	2121	
		7E	D4	00077	CLRL	-(SP)	2124	
		5A	DD	00079	PUSHL	R10		
6B		02	FB	0007B	CALLS	#2, DBG\$PRINT		
		52	DD	0007E	PUSHL	ADDRESS	2125	
0000V	CF	01	FB	00080	CALLS	#1, PRINT_REGNAME		
		66	D5	00085	TSTL	(R6)	2126	
		14	13	00087	BEQL	4\$		
	09	AA	9F	00089	PUSHAB	P.ADD		
6B		01	FB	0008C	CALLS	#1, DBG\$PRINT		
		66	DD	0008F	PUSHL	(R6)		
	0B	AA	9F	00091	PUSHAB	P.ADE		
6B		02	FB	00094	CALLS	#2, DBG\$PRINT		
	0F	AA	9F	00097	PUSHAB	P.ADF		
6B		01	FB	0009A	CALLS	#1, DBG\$PRINT		
		7E	D4	0009D	CLRL	-(SP)	2127	
	17	AA	9F	0009F	PUSHAB	P.ADG		
6B		02	FB	000A2	CALLS	#2, DBG\$PRINT		
00000000G	00	00	FB	000A5	CALLS	#0, DBG\$NEWLINE	2128	
		04	DD	000AC	PUSHL	#4	2129	
		01	DD	000AE	PUSHL	#1		
00000000G	00	02	FB	000B0	CALLS	#2, DBG\$PRINT_CONTROL		
		53	D5	000B7	TSTL	MODRSTPTR	2135	
		03	12	000B9	BNEQ	6\$		
		0082	31	000BB	BRW	15\$		
0D	00000000G	00	01	E1	000BE	BBC	#1, DBG\$GV_CONTROL+1, 7\$	2136
		8F	DD	000C6	PUSHL	#164072		
00000000G	00	01	FB	000CC	CALLS	#1, LIB\$SIGNAL		
62	28	A3	E9	000D3	BLBC	40(MODRSTPTR), 14\$	2144	
54	08	A3	DO	000D7	MOVL	8(MODRSTPTR), RSTPTR	2151	
		5C	13	000DB	BEQL	14\$	2156	
		30	BB	000DD	PUSHR	#^M<R4,R5>	2162	
0000V	CF	02	FB	000DF	CALLS	#2, IS_SYMBOL_BOUND_TO_REG		
59		50	DO	000E4	MOVL	R0, VALSPEC		
		4A	13	000E7	BEQL	13\$	2168	
03	69	02	00	ED	000E9	CMPZV	#0, #2, (VALSPEC), #3	2170

	35	10	43	12	000EE	BNEQ	13\$	:	
			AC	E9	000F0	BLBC	PRINT FLAG, 11\$	:	2173
			54	DD	000F4	PUSHL	RSTPTR	:	2176
00000000G	00		01	FB	000F6	CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME	:	
			57	D5	000FD	TSTL	BYTE_OFFSET	:	2177
			09	13	000FF	BEQL	9\$	:	
			57	DD	00101	PUSHL	BYTE_OFFSET	:	2179
00000000G	00		01	FB	00103	CALLS	#1, DBG\$PRINT_OFFSET	:	
			66	D5	0010A	TSTL	(R6)	:	2180
			14	13	0010C	BEQL	10\$	:	
		1B	AA	9F	0010E	PUSHAB	P.ADH	:	
	6B		01	FB	00111	CALLS	#1, DBG\$PRINT	:	
			66	DD	00114	PUSHL	(R6)	:	
		1D	AA	9F	00116	PUSHAB	P.ADI	:	
	6B		02	FB	00119	CALLS	#2, DBG\$PRINT	:	
		21	AA	9F	0011C	PUSHAB	P.ADJ	:	
	6B		01	FB	0011F	CALLS	#1, DBG\$PRINT	:	
00000000G	00		00	FB	00122	CALLS	#0, DBG\$NEWLINE	:	2181
	04		58	E8	00129	BLBS	SYMBOL_FOUND, 12\$	:	2190
08	BC		54	D0	0012C	MOVL	RSTPTR, @P_SYMID	:	2192
	58		01	D0	00130	MOVL	#1, SYMBOL_FOUND	:	2193
	54	08	A4	D0	00133	MOVL	8(RSTPTR), RSTPTR	:	2199
			A2	11	00137	BRB	8\$	:	2156
	53	10	A3	D0	00139	MOVL	16(MODRSTPTR), MODRSTPTR	:	2207
			FF77	31	0013D	BRW	5\$	:	2135
	31	10	AC	E9	00140	BLBC	PRINT FLAG, 17\$	:	2214
	2E		58	E8	00144	BLBS	SYMBOL_FOUND, 17\$	:	2217
			7E	D4	00147	CLRL	-(SP)	:	2220
		29	AA	9F	00149	PUSHAB	P.ADK	:	
	6B		02	FB	0014C	CALLS	#2, DBG\$PRINT	:	
			52	DD	0014F	PUSHL	ADDRESS	:	2221
0000V	CF		01	FB	00151	CALLS	#1, PRINT_REGNAME	:	
			66	D5	00156	TSTL	(R6)	:	2222
			14	13	00158	BEQL	16\$	:	
		42	AA	9F	0015A	PUSHAB	P.ADL	:	
	6B		01	FB	0015D	CALLS	#1, DBG\$PRINT	:	
			66	DD	00160	PUSHL	(R6)	:	
		44	AA	9F	00162	PUSHAB	P.ADM	:	
	6B		02	FB	00165	CALLS	#2, DBG\$PRINT	:	
		48	AA	9F	00168	PUSHAB	P.ADN	:	
	6B		01	FB	0016B	CALLS	#1, DBG\$PRINT	:	
00000000G	00		00	FB	0016E	CALLS	#0, DBG\$NEWLINE	:	2223
		08	BC	D5	00175	TSTL	@P_SYMID	:	2230
			02	12	00178	BNEQ	18\$	:	
			57	D4	0017A	CLRL	BYTE_OFFSET	:	
	76	9647	7E	0017C	MOVAQ	@(R6)+[BYTE_OFFSET], -(R6)	:	2231	
	50		01	D0	00180	MOVL	#1, R0	:	2232
				04	00183	RET		:	
		50	D4	00184	CLRL	R0	:	2234	
			04	00186	RET		:		

; Routine Size: 391 bytes, Routine Base: DBG\$CODE + 0CF8


```
2117 2235 1 GLOBAL ROUTINE DBG$SYMID_TO_PRIMARY (SYMID, OFFSET) =
2118 2236 1
2119 2237 1 FUNCTION
2120 2238 1     This routine takes a SYMID and an offset
2121 2239 1     and builds a Primary Descriptor for the object.
2122 2240 1     The routine must first do symbol table lookups to determine
2123 2241 1     the TYPEID, the KIND, and the FCODE.
2124 2242 1     It then can build a Primary root node and call DBG$BUILD_PRIMARY_SUBNODE
2125 2243 1     to create the Primary sub-node.
2126 2244 1
2127 2245 1     Then, if the offset is not zero, the Primary must be
2128 2246 1     modified to include the offset information. For example, if
2129 2247 1     the Primary is an array, the subscripts can be filled in with
2130 2248 1     the appropriate values.
2131 2249 1
2132 2250 1 INPUTS
2133 2251 1     SYMID - SYMID for the object
2134 2252 1     OFFSET - offset from the object to the desired address
2135 2253 1
2136 2254 1 OUTPUTS
2137 2255 1     A Primary Descriptor is built out of temporary memory and a
2138 2256 1     pointer to this descriptor and an offset to the primary
2139 2257 1     are returned.
2140 2258 1
2141 2259 2 BEGIN
2142 2260 2
2143 2261 2 MAP
2144 2262 2     OFFSET: REF VECTOR[0];           ! Offset value
2145 2263 2
2146 2264 2 LOCAL
2147 2265 2     PRIMPTR: REF DBG$PRIMARY;         ! Pointer to a Primary Descriptor
2148 2266 2
2149 2267 2
2150 2268 2
2151 2269 2 ! SYMID better not be zero.
2152 2270 2
2153 2271 2 IF .SYMID EQL 0
2154 2272 2 THEN
2155 2273 2     $DBG_ERROR ('DBGSYMBLZ\DBG$SYMID_TO_PRIMARY zero symid');
2156 2274 2
2157 2275 2
2158 2276 2 ! Allocate space for the Primary Root node and fill in some of the header
2159 2277 2 ! fields.
2160 2278 2
2161 2279 2 PRIMPTR = DBG$GET_TEMPMEM (DBG$K_PRIMARY_SIZE);
2162 2280 2 PRIMPTR[DBG$B_DHDR_LANG] = .DBG$B_LANGUAGE;
2163 2281 2 PRIMPTR[DBG$B_DHDR_TYPE] = DBG$K_PRIMARY_DESC;
2164 2282 2 PRIMPTR[DBG$W_DHDR_LENGTH] = DBG$K_PRIMARY_SIZE * %UPVAL;
2165 2283 2 PRIMPTR[DBG$L_DHDR_SYMID] = .SYMID;
2166 2284 2 PRIMPTR[DBG$L_PRIM_FLINK] = PRIMPTR[DBG$A_PRIM_FLINK];
2167 2285 2 PRIMPTR[DBG$L_PRIM_BLINK] = PRIMPTR[DBG$A_PRIM_FLINK];
2168 2286 2
2169 2287 2
2170 2288 2 ! Build the rest Primary subnode(s) for given SYMID.
2171 2289 2
2172 2290 2 RETURN BUILD_PRIMARY_SUBNODE(.PRIMPTR, .SYMID, .OFFSET);
2173 2291 1 END;
```


24	47	42	44	5C	5A	4C	42	4D	59	53	47	42	44	29	0034E	P.ADO:	.PSECT	DBG\$PLIT,NOWRT, SHR, PIC,0		
52	41	4D	49	52	50	5F	4F	54	5F	44	49	4D	59	53	0035D		.ASCII	\)DBGSYMBLZ\<92>\DBG\$SYMD_ID_TO_PRIMARY ze\	:	
											65	7A	20	59	0036C				:	
							64	69	6D	79	73	20	6F	72	00370		.ASCII	\ro symid\	:	
																	.PSECT	DBG\$CODE,NOWRT, SHR, PIC,0		
																	.ENTRY	DBG\$SYMD_ID_TO_PRIMARY, Save R2	:	2235
							52		04	AC	DO	00002					MOVL	SYMD, R2	:	2271
										15	12	00006					BNEQ	1\$	:	
										EF	9F	00008					PUSHAB	P.ADO	:	2273
										01	DD	0000E					PUSHL	#1	:	
										8F	DD	00010					PUSHL	#164706	:	
										03	FB	00016					CALLS	#3, LIB\$SIGNAL	:	
										09	DD	0001D					PUSHL	#9	:	2279
										01	FB	0001F					CALLS	#1, DBG\$GET TEMPMEM	:	
										00	90	00026					MOVB	DBG\$GB_LANGUAGE, 3(PRIMPTR)	:	2280
										8F	90	0002E					MOVB	#121, 2(PRIMPTR)	:	2281
										24	80	00033					MOVW	#36, (PRIMPTR)	:	2282
										52	DO	00036					MOVL	R2, 12(PRIMPTR)	:	2283
										A0	9E	0003A					MOVAB	20(PRIMPTR), 20(PRIMPTR)	:	2284
										A0	9E	0003F					MOVAB	20(PRIMPTR), 24(PRIMPTR)	:	2285
										AC	DD	00044					PUSHL	OFFSET	:	2290
										05	BB	00047					PUSHR	#*M<R0,R2>	:	
										03	FB	00049					CALLS	#3, BUILD_PRIMARY_SUBNODE	:	
										04	0004E						RET		:	2291

; Routine Size: 79 bytes, Routine Base: DBG\$CODE + 0E7F


```
2175 2292 1 GLOBAL ROUTINE DBG$NEW_SYMBOLIZE (ADDR, P_SYMID, P_BIT_OFFSET) =
2176 2293 1
2177 2294 1 FUNCTION:
2178 2295 1     This routine accepts a address descriptor, and attempts to symbolize
2179 2296 1     it as a symbol name plus offset.
2180 2297 1     It always returns the best possible symbolization; if the address can
2181 2298 1     be symbolized by more than one symbol name with the same offset, then
2182 2299 1     the first is chosen to be the best.
2183 2300 1
2184 2301 1 INPUTS:
2185 2302 1     ADDR          - The address of an address descriptor (byte and bit offset).
2186 2303 1
2187 2304 1     P_SYMID       - The address of a longword location where the "symbol identi-
2188 2305 1                   fier" should be returned. The "symbol identifier" is a value
2189 2306 1                   which uniquely identifies the returned symbol. This value is
2190 2307 1                   not directly understood outside the symbol table access rou-
2191 2308 1                   tines, but can be passed to various other symbol table access
2192 2309 1                   routines to extract information about the symbol.
2193 2310 1
2194 2311 1     P_BIT_OFFSET  - The address of a longword location where the bit offset from
2195 2312 1                   the SYMID symbol should be returned.
2196 2313 1
2197 2314 1 OUTPUTS:
2198 2315 1     SYMID         - A symbol identifier which uniquely identifies the symbol
2199 2316 1                   which best symbolizes ADDR is returned to SYMID. This symbol
2200 2317 1                   identifier can then be passed to any symbol table access rou-
2201 2318 1                   tine which accepts a SYMID parameter. If no suitable symbol
2202 2319 1                   can be found, a zero is returned to SYMID.
2203 2320 1
2204 2321 1     OFFSET       - The bit offset of ADDR relative to the SYMID symbol is
2205 2322 1                   returned to OFFSET. If (SYMID) is zero, this
2206 2323 1                   offset is simply the original address descriptor.
2207 2324 1
2208 2325 1     The routine returns true if symbolization was possible; otherwise it
2209 2326 1     returns false.
2210 2327 1
2211 2328 1
2212 2329 2 BEGIN
2213 2330 2
2214 2331 2 BIND
2215 2332 2     SYMID = .P_SYMID: REF RST$ENTRY,
2216 2333 2     BIT_OFFSET = .P_BIT_OFFSET;
2217 2334 2
2218 2335 2 MAP
2219 2336 2     ADDR: REF DBG$ADDRESS_DESC;      ! Pointer to address descriptor
2220 2337 2
2221 2338 2 LOCAL
2222 2339 2     BYTE_OFFSET,
2223 2340 2     SATPTR: REF SAT$ENTRY;
2224 2341 2
2225 2342 2
2226 2343 2     ! See if the address is a register address.
2227 2344 2
2228 2345 2     IF DBG$SYMBOLIZE_REG (.ADDR, SYMID, BIT_OFFSET, FALSE)
2229 2346 2     THEN
2230 2347 2         RETURN TRUE;
2231 2348 2
```



```
2232 2349 2
2233 2350 2
2234 2351 2
2235 2352 2
2236 2353 2
2237 2354 2
2238 2355 2
2239 2356 2
2240 2357 3
2241 2358 3
2242 2359 3
2243 2360 3
2244 2361 3
2245 2362 3
2246 2363 3
2247 2364 3
2248 2365 3
2249 2366 3
2250 2367 4
2251 2368 3
2252 2369 4
2253 2370 4
2254 2371 4
2255 2372 4
2256 2373 4
2257 2374 4
2258 2375 4
2259 2376 4
2260 2377 4
2261 2378 4
2262 2379 4
2263 2380 4
2264 2381 4
2265 2382 4
2266 2383 4
2267 2384 4
2268 2385 4
2269 2386 4
2270 2387 4
2271 2388 5
2272 2389 5
2273 2390 5
2274 2391 4
2275 2392 5
2276 2393 4
2277 2394 5
2278 2395 5
2279 2396 5
2280 2397 5
2281 2398 4
2282 2399 3
2283 2400 3
2284 2401 3
2285 2402 2
2286 2403 2
2287 2404 2
2288 2405 2
```

```
! See if the address is a static address.
SATPTR = DBG$SEARCH_PROG_SAT (.SAT$START_ADDR,
                              .ADDR[DBG$L_ADDRESS_BYTE_ADDR],
                              BYTE_OFFSET);
IF .SATPTR NEQ 0
THEN
  BEGIN
    SYMID = .SATPTR[SAT$L_RSTPTR];

    ! If the address is a code address, then print the symbol
    ! name and corresponding line number. If there is no line
    ! number, then just print the routine name and offset.
    IF (.SYMID[RST$B_KIND] EQL RST$K_ROUTINE) OR
       (.SYMID[RST$B_KIND] EQL RST$K_BLOCK) OR
       (.SYMID[RST$B_KIND] EQL RST$K_LABEL) OR
       (.SYMID[RST$B_KIND] EQL RST$K_ENTRY)
    THEN
      BEGIN
        LOCAL
          ADDRESS,
          LINE_NO,
          STMT_NO,
          START_PC,
          END_PC,
          MOD_SYMID : REF RST$ENTRY,
          TMP_OFFSET,
          TMP_SYMID : REF RST$ENTRY;

          ! Line no. corresponding to code address.
          ! statement no.
          ! Beginning pc address for line.
          ! Dummy.
          ! Dummy.
          ! Temporay offset
          ! Temporay symid for %line

          ! Lookup the line number; if found, then print the symbol name
          ! and line number.
          MOD_SYMID = .SYMID;
          WHILE .MOD_SYMID[RST$B_KIND] NEQ RST$K_MODULE DO
            MOD_SYMID = .MOD_SYMID[RST$L_UPSCOPEPTR];
          ADDRESS = .ADDR[DBG$L_ADDRESS_BYTE_ADDR];
          IF (DBG$PC_TO_LINE_LOOKUP (.ADDRESS, LINE_NO, STMT_NO,
                                     START_PC, END_PC, MOD_SYMID))
          AND ((.LINE_NO NEQ 0) OR (.STMT_NO NEQ 0))
          THEN
            IF (.BYTE_OFFSET NEQ 0) OR (.SYMID[RST$B_KIND] EQL RST$K_BLOCK)
            THEN
              BEGIN
                SYMID = DBG$STA_LINE_NUM_RST(.SYMID, .LINE_NO, .STMT_NO,
                                              .START_PC, .END_PC);
                BYTE_OFFSET = .ADDRESS - .START_PC;
              END;
            END;
          BIT_OFFSET = .ADDR[DBG$L_ADDRESS_BIT_OFFSET] + (8 * .BYTE_OFFSET);
          RETURN TRUE;
        END;
      ! See if the address is a global symbol.
```



```

: 2289      2406      2
: 2290      2407      2
: 2291      2408      2
: 2292      2409      2
: 2293      2410      2
: 2294      2411      2
: 2295      2412      2
: 2296      2413      2
: 2297      2414      2
: 2298      2415      2
: 2299      2416      2
: 2300      2417      2
: 2301      2418      2
: 2302      2419      2
: 2303      2420      2
: 2304      2421      2
: 2305      2422      2
: 2306      2423      2
: 2307      2424      1

```

```

!
IF DBG$SEARCH_GLOBAL (.ADDR, SYMID, BIT_OFFSET, FALSE)
THEN
    RETURN TRUE;

! See if the address is on the call stack.
!
IF DBG$SEARCH_VAX_CALL_STACK (.ADDR, SYMID, BIT_OFFSET, FALSE)
THEN
    RETURN TRUE;

! The address was not found, and symbolization was thus impossible.
!
SYMID = 0;
BIT_OFFSET = .ADDR;
RETURN FALSE;
END;

```

			00FC 00000	.ENTRY	DBG\$NEW_SYMBOLIZE, Save R2,R3,R4,R5,R6,R7	2292
	SE		18 C2 00002	SUBL2	#24, SP	
	55	08	AC D0 00005	MOVL	P_SYMID, R5	2332
	57	0C	AC D0 00009	MOVL	P_BIT_OFFSET, R7	2333
			7E D4 0000D	CLRL	-(SP)	2345
		00A0	8F BB 0000F	PUSHR	#*M<R5,R7>	
	52	04	AC D0 00013	MOVL	ADDR, R2	
			52 DD 00017	PUSHL	R2	
FEOC	CF		04 FB 00019	CALLS	#4, DBG\$SYMBOLIZE_REG	
	03		50 E9 0001E	BLBC	R0, 1\$	
		00BE	31 00021	BRW	10\$	
			5E DD 00024 1\$:	PUSHL	SP	2352
			62 DD 00026	PUSHL	(R2)	2353
F2CD	CF	00000000G	00 DD 00028	PUSHL	SAT\$START_ADDR	2352
			03 FB 0002E	CALLS	#3, DBG\$SEARCH_PROG_SAT	
			50 D5 00033	TSTL	SATPTR	2355
			03 12 00035	BNEQ	2\$	
		008C	31 00037	BRW	9\$	
	65	0C	A0 D0 0003A 2\$:	MOVL	12(SATPTR), (R5)	2358
	53		65 D0 0003E	MOVL	(R5), R3	2364
	54	14	A3 9A 00041	MOVZBL	20(R3), R4	
	02		54 91 00045	CMPB	R4, #2	
			0F 13 00048	BEQL	3\$	
	03		54 91 0004A	CMPB	R4, #3	2365
			0A 13 0004D	BEQL	3\$	
	04		54 91 0004F	CMPB	R4, #4	2366
			05 13 00052	BEQL	3\$	
	08		54 91 00054	CMPB	R4, #8	2367
			63 12 00057	BNEQ	8\$	
04	AE		53 D0 00059 3\$:	MOVL	R3, MOD SYMID	2384
	50	04	AE D0 0005D 4\$:	MOVL	MOD SYMID, R0	2385
	01	14	A0 91 00061	CMPB	20(R0), #1	
			07 13 00065	BEQL	5\$	

04	AE	10	A0	D0	00067	MOVL	16(R0), MOD_SYMID	2386
			EF	11	0006C	BRB	4\$	
	56		62	D0	0006E	5\$: MOVL	(R2), ADDRESS	2387
		04	AE	9F	00071	PUSHAB	MOD_SYMID	2388
		0C	AE	9F	00074	PUSHAB	END_PC	
		14	AE	9F	00077	PUSHAB	START_PC	
		1C	AE	9F	0007A	PUSHAB	STMT_NO	
		24	AE	9F	0007D	PUSHAB	LINE_NO	
			56	DD	00080	PUSHL	ADDRESS	
00000000G	00		06	FB	00082	CALLS	#6, DBG\$PC_TO_LINE_LOOKUP	
	30		50	E9	00089	BLBC	R0, 8\$	
		14	AE	D5	0008C	TSTL	LINE_NO	2390
			05	12	0008F	BNEQ	6\$	
		10	AE	D5	00091	TSTL	STMT_NO	
			26	13	00094	BEQL	8\$	
			6E	D5	00096	6\$: TSTL	BYTE_OFFSET	2392
			05	12	00098	BNEQ	7\$	
	03		54	91	0009A	CMPB	R4, #3	
			1D	12	0009D	BNEQ	8\$	
		08	AE	DD	0009F	7\$: PUSHL	END_PC	2396
		10	AE	DD	000A2	PUSHL	START_PC	
		18	AE	DD	000A5	PUSHL	STMT_NO	2395
		20	AE	DD	000A8	PUSHL	LINE_NO	
			53	DD	000AB	PUSHL	R3	
00000000G	00		05	FB	000AD	CALLS	#5, DBG\$STA_LINE_NUM_RST	
	65		50	D0	000B4	MOVL	R0, (R5)	
6E	56	0C	AE	C3	000B7	SUBL3	START_PC, ADDRESS, BYTE_OFFSET	2397
	50		6E	D0	000BC	8\$: MOVL	BYTE_OFFSET, R0	2400
	67	04	B240	7E	000BF	MOVAQ	@4(R2)(R0), (R7)	
			1C	11	000C4	BRB	10\$	2401
			7E	D4	000C6	9\$: CLRL	-(SP)	2407
		00A4	8F	BB	000C8	PUSHR	#^M<R2,R5,R7>	
F38E	CF		04	FB	000CC	CALLS	#4, DBG\$SEARCH_GLOBAL	
	0E		50	E8	000D1	BLBS	R0, 10\$	
			7E	D4	000D4	CLRL	-(SP)	2414
		00A4	8F	BB	000D6	PUSHR	#^M<R2,R5,R7>	
F7A6	CF		04	FB	000DA	CALLS	#4, DBG\$SEARCH_VAX_CALL_STACK	
	04		50	E9	000DF	BLBC	R0, 11\$	
	50		01	D0	000E2	10\$: MOVL	#1, R0	2416
				04	000E5	RET		
			65	D4	000E6	11\$: CLRL	(R5)	2421
	67		52	D0	000E8	MOVL	R2, (R7)	2422
			50	D4	000EB	CLRL	R0	2423
				04	000ED	RET		2424

; Routine Size: 238 bytes, Routine Base: DBG\$CODE + 0ECE


```
2309 2425 1 GLOBAL ROUTINE DBG$PC_TO_SYMID(PC, RSTPTR) =
2310 2426 1
2311 2427 1 FUNCITON
2312 2428 1 This routine goes through the Module SAT table and/or Global symbol
2313 2429 1 chain to locate RST for the best match for a address. If the 3rd
2314 2430 1 parameter is present, that means if the above
2315 2431 1 search failed, this routine goes on searching global symbol chain.
2316 2432 1 If the 3rd parameter is not present, that means to search Module SAT
2317 2433 1 only.
2318 2434 1
2319 2435 1 INPUTS
2320 2436 1
2321 2437 1 ADDRESS - The address for which a match is desired.
2322 2438 1
2323 2439 1 RSTPTR - A pointer to RST entry.
2324 2440 1
2325 2441 1 3rd Parameter - optional.
2326 2442 1
2327 2443 1 OUTPUTS
2328 2444 1
2329 2445 1 TRUE - If a match is found.
2330 2446 1 FALSE - otherwise.
2331 2447 1
2332 2448 1
2333 2449 2 BEGIN
2334 2450 2
2335 2451 2 BUILTIN
2336 2452 2 ACTUALCOUNT;
2337 2453 2
2338 2454 2 LOCAL
2339 2455 2 ADDR: DBG$ADDRESS_DESC, ! Address Descriptor
2340 2456 2 OFFSET, ! Bit offset
2341 2457 2 SATPTR: REF SAT$ENTRY;
2342 2458 2
2343 2459 2 SATPTR = DBG$SEARCH_PROG_SAT (.SAT$START_ADDR, .PC, OFFSET);
2344 2460 2 IF .SATPTR NEQ 0
2345 2461 2 THEN
2346 2462 3 BEGIN
2347 2463 3 .RSTPTR = .SATPTR[SAT$L_RSTPTR];
2348 2464 3 RETURN TRUE;
2349 2465 2 END;
2350 2466 2 IF ACTUALCOUNT() EQL 2 THEN RETURN FALSE;
2351 2467 2 ADDR[DBG$L_ADDRESS_BYTE_ADDR] = .PC;
2352 2468 2 ADDR[DBG$L_ADDRESS_BIT_OFFSET] = 0;
2353 2469 2 RETURN DBG$SEARCH_GLOBAL(ADDR, .RSTPTR, OFFSET);
2354 2470 1 END;
```

```
SE 0000 00000
OC C2 00002
SE DD 00005
AC DD 00007
00 DD 0000A
F1FD CF 00000000G 03 FB 00010
```

```
.ENTRY DBG$PC_TO_SYMID, Save nothing
SUBL2 #12, SP
PUSHL SP
PUSHL PC
PUSHL SAT$START_ADDR
CALLS #3, DBG$SEARCH_PROG_SAT
```

```
: 2425
:
: 2459
:
:
```


DBGSYMBLZ
V04-000

G 8
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 75
(18)

			50	D5	00015	TSTL	SATPTR	:	2460
			09	13	00017	BEQL	1\$	:	
08	BC	0C	A0	D0	00019	MOVL	12(SATPTR), @RSTPTR	:	2463
	50		01	D0	0001E	MOVL	#1, R0	:	2464
				04	00021	RET		:	
	02		6C	91	00022	CMPB	(AP), #2	:	2466
			16	13	00025	BEQL	2\$	:	
04	AE	04	AC	D0	00027	MOVL	PC, ADDR	:	2467
		08	AE	D4	0002C	CLRL	ADDR+4	:	2468
			5E	DD	0002F	PUSHL	SP	:	2469
		08	AC	DD	00031	PUSHL	RSTPTR	:	
		0C	AE	9F	00034	PUSHAB	ADDR	:	
F335	CF		03	FB	00037	CALLS	#3, DBG\$SEARCH_GLOBAL	:	
				04	0003C	RET		:	
			50	D4	0003D	CLRL	R0	:	2470
				04	0003F	RET		:	

; Routine Size: 64 bytes, Routine Base: DBG\$CODE + 0FBC


```
2356 2471 1 ROUTINE FIND_ROUTINE (PCVAL, PRINT_FLAG) =
2357 2472 1
2358 2473 1 FUNCTION
2359 2474 1     This routine looks through the SAT to find the most nested routine
2360 2475 1     containing the PC value passed to the routine.
2361 2476 1
2362 2477 1 INPUTS
2363 2478 1     PCVAL           - The PC.
2364 2479 1
2365 2480 1 OUTPUTS
2366 2481 1     If successful, and the module is set, the routine returns the routine
2367 2482 1     RST pointer.
2368 2483 1     If successful, and the module is not set, the routine returns the
2369 2484 1     module RST pointer.
2370 2485 1
2371 2486 1     If unsuccessful, the routine return 0.
2372 2487 1
2373 2488 1
2374 2489 2 BEGIN
2375 2490 2
2376 2491 2 LOCAL
2377 2492 2     MODRSTPTR:      REF RST$ENTRY,  ! Ptr to module's RST entry.
2378 2493 2     RSTPTR:         REF RST$ENTRY,  ! Ptr to symbol's RST entry.
2379 2494 2     BYTE_OFFSET,    ! Byte offset.
2380 2495 2     PROG_SATPTR:    REF SAT$ENTRY;  ! Ptr to program level SAT entry.
2381 2496 2
2382 2497 2
2383 2498 2     ! Initialize program level SAT pointer.
2384 2499 2
2385 2500 2     PROG_SATPTR = .SAT$START_ADDR;
2386 2501 2
2387 2502 2
2388 2503 2     ! Search through the program level SAT.
2389 2504 2
2390 2505 2     WHILE .PROG_SATPTR NEQ 0 DO
2391 2506 2         BEGIN
2392 2507 2
2393 2508 2
2394 2509 2         ! If current SAT entry is past the address we are looking for, then
2395 2510 2         ! exitloop. (SAT is sorted on start address.)
2396 2511 2
2397 2512 2         IF .PROG_SATPTR[SAT$L_START] GTRA .PCVAL
2398 2513 2         THEN
2399 2514 2             EXITLOOP;
2400 2515 2
2401 2516 2
2402 2517 3         IF (.PROG_SATPTR[SAT$L_START] LEQA .PCVAL) AND
2403 2518 4         (.PROG_SATPTR[SAT$L_END] GEQA .PCVAL)
2404 2519 3         THEN
2405 2520 4             BEGIN
2406 2521 4
2407 2522 4
2408 2523 4             ! Search the module level SAT looking for a routine whose address
2409 2524 4             ! range includes .PCVAL. (The most nested routine is returned.)
2410 2525 4
2411 2526 4             SEARCH_MODULE_SAT (.PROG_SATPTR, .PCVAL, MODRSTPTR, RSTPTR,
2412 2527 4             BYTE_OFFSET, .PRINT_FLAG);
```



```

: 2413
: 2414
: 2415
: 2416
: 2417
: 2418
: 2419
: 2420
: 2421
: 2422
: 2423
: 2424
: 2425
: 2426
: 2427
: 2428
: 2429
: 2430
: 2431
: 2432
: 2433
: 2434
: 2435
: 2436
: 2437
: 2438
: 2439
: 2440
: 2441
: 2442
: 2443
: 2444
: 2445
: 2446
: 2447
: 2448
: 2449
: 2450
: 2451
: 2452
: 2453

```

```

2528 4
2529 4
2530 4
2531 4
2532 4
2533 4
2534 4
2535 4
2536 4
2537 4
2538 4
2539 4
2540 4
2541 4
2542 4
2543 4
2544 4
2545 4
2546 4
2547 4
2548 5
2549 5
2550 5
2551 5
2552 5
2553 4
2554 4
2555 4
2556 3
2557 3
2558 3
2559 3
2560 3
2561 3
2562 2
2563 2
2564 2
2565 2
2566 2
2567 2
2568 1

```

```

: If the module is not set, can only return at most the module
: RST pointer.
IF NOT .MODRSTPTR[RST$V_MODSET]
THEN
    RETURN .MODRSTPTR;

: If the module was set, make sure RSTPTR is not zero.
IF .RSTPTR EQL 0 THEN RETURN .RSTPTR;

: If the module was set, then find out if the RST entry is a
: routine (the pcval could have more closely symbolized to a
: label or block within the routine).
WHILE .RSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
    BEGIN
        IF .RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE
        THEN
            RETURN .RSTPTR;
            RSTPTR = .RSTPTR[RST$L_UPSCOPEPTR];
        END;
    END;

RETURN .MODRSTPTR;
END;

: Nothing found yet. Get next SAT entry.
:
PROG_SATPTR = .PROG_SATPTR[SAT$L_FLINK];
END;

: We did not find a routine containing the PC.
:
RETURN 0;
END;

```

				0004 00000 FIND_ROUTINE:			
	5E		0C	C2 00002	.WORD	Save R2	: 2471
	52	00000000G	00	D0 00005	SUBL2	#12, SP	: 2500
			56	13 0000C	1\$: MOVL	SAT\$START_ADDR, PROG_SATPTR	: 2505
04	AC	04	A2	D1 0000E	BEQL	5\$	: 2512
			4F	1A 00013	CMPL	4(PROG_SATPTR), PCVAL	
04	AC	08	A2	D1 00015	BGTRU	5\$	: 2518
			43	1F 0001A	CMPL	8(PROG_SATPTR), PCVAL	
		08	AC	DD 0001C	BLSSU	4\$	: 2527
		04	AE	9F 0001F	PUSHL	PRINT_FLAG	: 2526
					PUSHAB	BYTE_OFFSET	

		0C	AE	9F	00022	PUSHAB	RSTPTR	:	
		14	AE	9F	00025	PUSHAB	MODRSTPTR	:	
		04	AC	DD	00028	PUSHL	PCVAL	:	
			52	DD	0002B	PUSHL	PROG_SATPTR	:	
0000V	CF		06	FB	0002D	CALLS	#6, SEARCH_MODULE_SAT	:	
	51	08	AE	D0	00032	MOVL	MODRSTPTR, R1	:	2533
	21	28	A1	E9	00036	BLBC	40(R1), 3\$	:	
		04	AE	D5	0003A	TSTL	RSTPTR	:	2540
			05	12	0003D	BNEQ	2\$	:	
	50	04	AE	D0	0003F	MOVL	RSTPTR, R0	:	
			04	00043	RET			:	
	50	04	AE	D0	00044	MOVL	RSTPTR, R0	:	2547
	01	14	A0	91	00048	CMPB	20(R0), #1	:	
			0D	13	0004C	BEQL	3\$	:	
	02	14	A0	91	0004E	CMPB	20(R0), #2	:	2549
			12	13	00052	BEQL	6\$	:	
04	AE	10	A0	D0	00054	MOVL	16(R0), RSTPTR	:	2552
			E9	11	00059	BRB	2\$	:	2547
	50		51	D0	0005B	MOVL	R1, R0	:	2555
			04	0005E	RET			:	
	52		62	D0	0005F	MOVL	(PROG_SATPTR), PROG_SATPTR	:	2561
			A8	11	00062	BRB	1\$	:	2505
			50	D4	00064	CLRL	R0	:	2567
			04	00066	6\$:	RET		:	2568

; Routine Size: 103 bytes, Routine Base: DBG\$CODE + 0FFC


```
: 2455      2569 1 ROUTINE GET_ADDRESS_DESC (ADDRPTR, ADDRESS_DESC, MESSAGE_VECT) =
: 2456      2570 1
: 2457      2571 1 FUNCTION
: 2458      2572 1     This routine interprets an address representation (AED for level 2
: 2459      2573 1     support, Primary or Value Descriptor for level 3 support), and an
: 2460      2574 1     address descriptor is returned.
: 2461      2575 1
: 2462      2576 1 INPUTS
: 2463      2577 1     ADDRPTR      - The address representation.
: 2464      2578 1
: 2465      2579 1     ADDRESS_DESC  - The address descriptor.
: 2466      2580 1
: 2467      2581 1     MESSAGE_VECT - The address of a longword to contain the address
: 2468      2582 1     of a standard message argument vector.
: 2469      2583 1
: 2470      2584 1 OUTPUTS
: 2471      2585 1     If the ADDRPTR type is a permanent descriptor of AED, then the register
: 2472      2586 1     address is returned in the byte address field of the address descriptor.
: 2473      2587 1     The bit offset is set to 0. Otherwise, a regular address descriptor is
: 2474      2588 1     constructed and returned.
: 2475      2589 1
: 2476      2590 1     On success, STS$K_SUCCESS is returned.
: 2477      2591 1
: 2478      2592 1     On failure, either the routine signals its way out, or returns a
: 2479      2593 1     message vector and STS$K_SEVERE.
: 2480      2594 1
: 2481      2595 1
: 2482      2596 2 BEGIN
: 2483      2597 2
: 2484      2598 2 MAP
: 2485      2599 2     ADDRPTR      : REF DBG$AED,      ! Ptr to Address Exp. Desc.
: 2486      2600 2     ADDRESS_DESC : REF DBG$ADDRESS_DESC; ! Ptr to Address Desc.
: 2487      2601 2
: 2488      2602 2 LOCAL
: 2489      2603 2 TYPE;
: 2490      2604 2
: 2491      2605 2
: 2492      2606 2 ! See if we have a level 2 address expression descriptor. If we do
: 2493      2607 2 ! check for the permanent symbol AED type.
: 2494      2608 2
: 2495      2609 2 IF .ADDRPTR[DBG$B_AED_SIGNATURE] EQL DBG$K_AED
: 2496      2610 2 THEN
: 2497      2611 3 BEGIN
: 2498      2612 3 IF .ADDRPTR[DBG$B_AED_TYPE] EQL DBG$K_PERM_DESC
: 2499      2613 3 THEN
: 2500      2614 4 BEGIN
: 2501      2615 4 LOCAL
: 2502      2616 4     REGNUM,      ! Register Number
: 2503      2617 4     PERM_DESC: REF DBG$PERMSD; ! Ptr to the permanent symbol desc.
: 2504      2618 4
: 2505      2619 4
: 2506      2620 4 ! We do, so initialize perm_desc to the AED value field,
: 2507      2621 4 ! which points to a permanent symbol descriptor.
: 2508      2622 4
: 2509      2623 4 PERM_DESC = .ADDRPTR[DBG$L_AED_VALUE];
: 2510      2624 4
: 2511      2625 4
```



```
: 2512      2626  4      ! Calculate the register number from the perm_desc ID field.
: 2513      2627  4      !
: 2514      2628  4      REGNUM = .PERM_DESC[DBG$B_PERMSD_ID] - DBG$K_R0;
: 2515      2629  4
: 2516      2630  4
: 2517      2631  4      ! Get the address of the register from DBG$REG_VALUES.
: 2518      2632  4      !
: 2519      2633  4      ADDRESS_DESC[DBG$L_ADDRESS_BYTE_ADDR] = DBG$REG_VALUES[.REGNUM];
: 2520      2634  4      ADDRESS_DESC[DBG$L_ADDRESS_BIT_OFFSET] = 0;
: 2521      2635  4      RETURN ST$K_SUCCESS;
: 2522      2636  3      END;
: 2523      2637  3
: 2524      2638  2      END;
: 2525      2639  2
: 2526      2640  2
: 2527      2641  2      ! Any other kinds call DBG$NGET_ADDRESS to get the addr. desc.
: 2528      2642  2      !
: 2529      2643  2      IF NOT DBG$NGET_ADDRESS (.ADDRPTR, .ADDRESS_DESC, TYPE, FALSE, .MESSAGE_VECT)
: 2530      2644  2      THEN
: 2531      2645  2          RETURN ST$K_SEVERE;
: 2532      2646  2
: 2533      2647  2      RETURN ST$K_SUCCESS;
: 2534      2648  1      END;
```

```
0004 00000 GET_ADDRESS_DESC:
      5E      04  C2 00002      .WORD      Save R2      : 2569
      52      04  AC D0 00005      SUBL2      #4, SP
      84 8F      02  A2 91 00009      MOVL      ADDRPTR, R2      : 2609
      7B 8F      23 12 0000E      CMPB      2(R2), #132
      50      04  62 91 00010      BNEQ      1$
      51      60 1D 12 00014      CMPB      (R2), #123
      51      FF38 C1 9E 0001D      BNEQ      1$
      50      08  AC D0 00022      MOVL      4(R2), PERM_DESC
      60 00000000G00 41 DE 00026      MOVZBL   (PERM_DESC), REGNUM
      04      A0 D4 0002E      MOVAB    -200(R1), REGNUM
      0C      AC DD 00033 1$:      MOVL     ADDRESS_DESC, R0
      08      7E D4 00036      MOVAL    DBG$REG_VALUES[REGNUM], (R0)
      08      AE 9F 00038      CLRL     4(R0)
      00      52 DD 0003E      BRB      2$
      04      05 FB 00040      PUSHL   MESSAGE_VECT
      50      50 E8 00047      CLRL     -(SP)
      50      04 D0 0004A      PUSHAB  TYPE
      50      01 D0 0004E 2$:      PUSHL   ADDRESS_DESC
      04      04 00051      PUSHL   R2
      00      00      05 FB 00040      CALLS   #5, DBG$NGET_ADDRESS
      04      50 E8 00047      BLBS    R0, 2$
      50      04 D0 0004A      MOVL     #4, R0
      50      01 D0 0004E      RET
      04      04 00051      MOVL     #1, R0
      00      00      05 FB 00040      RET
```

; Routine Size: 82 bytes, Routine Base: DBG\$CODE + 1063

DBGSYMBLZ
V04-000

M 8
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 81
(20)


```

: 2536      2649 1 ROUTINE GSTADDRESS(RSTPTR) =
: 2537      2650 1
: 2538      2651 1 FUNCTION:
: 2539      2652 1     This routine get the address of GST from GST's RST entry.
: 2540      2653 1
: 2541      2654 1 INPUTS:
: 2542      2655 1     RSTPTR - Pointer to GST's RST entry.
: 2543      2656 1
: 2544      2657 1 OUTPUT:
: 2545      2658 1     The address of GST.
: 2546      2659 1
: 2547      2660 1
: 2548      2661 2 BEGIN
: 2549      2662 2
: 2550      2663 2 MAP
: 2551      2664 2     RSTPTR: REF RST$ENTRY;           ! Pointer to GST's RST entry
: 2552      2665 2
: 2553      2666 2 LOCAL
: 2554      2667 2     DSTPTR: REF DST$RECORD,         ! Pointer to DST record
: 2555      2668 2     GST_VALUE;                     ! Address of Global symbol
: 2556      2669 2
: 2557      2670 2
: 2558      2671 2 CASE .RSTPTR[RST$B_KIND] FROM RST$K_KIND_MINIMUM TO
: 2559      2672 2     RST$K_KIND_MAXIMUM OF
: 2560      2673 2 SET
: 2561      2674 2
: 2562      2675 2 [RST$K_ROUTINE]:
: 2563      2676 2     BEGIN
: 2564      2677 2     GST_VALUE = .RSTPTR[RST$L_STARTADDR];
: 2565      2678 2     END;
: 2566      2679 2
: 2567      2680 2 [RST$K_DATA]:
: 2568      2681 2     BEGIN
: 2569      2682 2     DSTPTR = .RSTPTR[RST$L_DSTPTR];
: 2570      2683 2     GST_VALUE = .DSTPTR[DST$L_VALUE];
: 2571      2684 2     END;
: 2572      2685 2
: 2573      2686 2 [INRANGE, OVRANGE]:
: 2574      2687 2     $DBG_ERROR('DBGSYMBLZ\GSTADDRESS');
: 2575      2688 2
: 2576      2689 2 TES;
: 2577      2690 2
: 2578      2691 2 RETURN .GST_VALUE;
: 2579      2692 2
: 2580      2693 1 END;
```

```

: 41 54 53 47 5C 5A 4C 42 4D 59 53 47 42 44 14 00378 P.ADP: .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
: 53 53 45 52 44 44 00387 .ASCII <20>\DBGSYMBLZ\<92>\GSTADDRESS\
:
```

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


0000 00000 GSTADDRESS:							
001C 001C 001C	0D 0033 0039 001C	50 00 001C 001C 001C 001C	04 14	AC AO 001C 001C 001C 001C	D0 8F 0000B 00013 0001B 00023	1\$:	Save nothing
							RSTPTR, R0
							20(R0), #0, #13
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$,-
							2\$-1\$
							PUSHAB
							PUSHL
							PUSHL
							CALLS
							BRB
							MOVL
							BRB
							MOVL
							MOVL
							MOVL
							RET

; Routine Size: 80 bytes, Routine Base: DBG\$CODE + 10B5

; 2581 2694 1


```
2583 2695 1 ROUTINE IS_SYMBOL_BOUND_TO_REG (SYMID, REGNUM) =
2584 2696 1
2585 2697 1 FUNCTION
2586 2698 1     This function accepts a symid as input, and determines if the symbol
2587 2699 1     associated with this symid is bound to the register regnum.
2588 2700 1
2589 2701 1 INPUTS
2590 2702 1     SYMID   - Pointer to symbol's RST entry.
2591 2703 1
2592 2704 1     REGNUM  - The register number in question.
2593 2705 1
2594 2706 1 OUTPUTS
2595 2707 1     The routine returns a DST value spec. if the symbol is bound to regnum
2596 2708 1     in some way; otherwise, it returns false.
2597 2709 1
2598 2710 1
2599 2711 2 BEGIN
2600 2712 2
2601 2713 2 MAP
2602 2714 2     SYMID           : REF RST$ENTRY;           ! Ptr. to symbol's RST entry.
2603 2715 2
2604 2716 2 LOCAL
2605 2717 2     VALSPEC         : REF DST$VAL_SPEC,         ! Ptr to DST record's
2606 2718 2                                     ! value spec.
2607 2719 2     BLITRLR         : REF DST$BLI_TRAILER1,      ! Ptr to Bliss DST record
2608 2720 2                                     ! trailer.
2609 2721 2     BLIVALSPEC      : BLOCK[8, BYTE]             ! Value spec buffer for
2610 2722 2     FIELD (DST$VS_HDR_FIELDS),                 ! Bliss special cases DST.
2611 2723 2     DSTPTR          : REF DST$RECORD;           ! Ptr. to DST record value spec.
2612 2724 2
2613 2725 2
2614 2726 2 $ABORT_ON_CONTROL_Y;
2615 2727 2
2616 2728 2 ! See if the entry is a data type.
2617 2729 2 !
2618 2730 2 IF (.SYMID[RST$B_KIND] EQL RST$K_DATA) OR
2619 2731 2     (.SYMID[RST$B_KIND] EQL RST$K_TYPCOMP)
2620 2732 2 THEN
2621 2733 2     BEGIN
2622 2734 2
2623 2735 2         ! It was, so get the corresponding DST entry.
2624 2736 2         !
2625 2737 2         !
2626 2738 2         !
2627 2739 2         !
2628 2740 2         !
2629 2741 2         !
2630 2742 2         ! Now get the value spec.
2631 2743 2         !
2632 2744 2         !
2633 2745 2         !
2634 2746 2         !
2635 2747 2         !
2636 2748 2         ! Get the value spec for those DST records
2637 2749 2         ! that can be represented by the standard
2638 2750 2         ! format DST.
2639 2751 2         !
```



```
: 2640      2752      [DSC$K_DTYPE_LOWEST TO DSC$K_DTYPE_HIGHEST,  
: 2641      2753      DST$K_BOOL, DST$K_SEPTYP, DST$K_LBLORLIT,  
: 2642      2754      DST$K_ENTRY, DST$K_RECBEQ, DST$K_ENUMELT]:  
: 2643      2755      BEGIN  
: 2644      2756      VALSPEC = DSTPTR[DST$B_VFLAGS];  
: 2645      2757      END;  
: 2646      2758  
: 2647      2759  
: 2648      2760      ! Handle the Bliss Special Cases DST Record.  
: 2649      2761  
: 2650      2762      [DST$K_BLI]:  
: 2651      2763      BEGIN  
: 2652      2764      BLIVALSPEC[DST$B_VS_VFLAGS] = .DSTPTR[DST$B_BLI_VFLAGS];  
: 2653      2765      BLITRLR = DSTPTR[DST$A_BLI_TRLR1] + .DSTPTR[DST$B_BLI_LNG];  
: 2654      2766      BLIVALSPEC[DST$L_VS_VALUE] = .BLITRLR[DST$L_BLI_VALUE];  
: 2655      2767      VALSPEC = BLIVALSPEC[DST$B_VS_VFLAGS];  
: 2656      2768      END;  
: 2657      2769  
: 2658      2770  
: 2659      2771      ! Other types of symbols cannot be bound to a  
: 2660      2772      ! register, so ignore them.  
: 2661      2773  
: 2662      2774      [INRANGE]:  
: 2663      2775      RETURN FALSE;  
: 2664      2776  
: 2665      2777      ! Anything else is an illegal DST type.  
: 2666      2778  
: 2667      2779      [OUTRANGE]:  
: 2668      2780      SIGNAL (DBG$_INVDSTREC);  
: 2669      2781  
: 2670      2782      TES;  
: 2671      2783  
: 2672      2784  
: 2673      2785      ! Loop through trailing value spec if present to get to actual  
: 2674      2786      ! value spec.  
: 2675      2787  
: 2676      2788      WHILE .VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_TVS DO  
: 2677      2789      VALSPEC = VALSPEC[DST$A_VS_TVS_BASE] + .VALSPEC[DST$L_VS_TVS_OFFSET];  
: 2678      2790  
: 2679      2791      SELECTONE TRUE OF  
: 2680      2792      SET  
: 2681      2793  
: 2682      2794  
: 2683      2795      ! Now, look at the VFLAGS field.  
: 2684      2796  
: 2685      2797      [.VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_NOVAL]:  
: 2686      2798      RETURN FALSE;  
: 2687      2799  
: 2688      2800  
: 2689      2801      ! If valspec gives the offset to a descriptor, just return.  
: 2690      2802  
: 2691      2803      [.VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_DSC]:  
: 2692      2804      RETURN FALSE;  
: 2693      2805  
: 2694      2806  
: 2695      2807      ! If valspec gives a bit offset, return.  
: 2696      2808
```



```

: 2697      2809      3
: 2698      2810
: 2699      2811
: 2700      2812
: 2701      2813
: 2702      2814
: 2703      2815
: 2704      2816
: 2705      2817
: 2706      2818
: 2707      2819
: 2708      2820
: 2709      2821
: 2710      2822
: 2711      2823
: 2712      2824
: 2713      2825
: 2714      2826
: 2715      2827
: 2716      2828
: 2717      2829
: 2718      2830
: 2719      2831
: 2720      2832
: 2721      2833
: 2722      2834
: 2723      2835
: 2724      2836
: 2725      2837
: 2726      2838
: 2727      2839
: 2728      2840
: 2729      2841
: 2730      2842
: 2731      2843
: 2732      2844
: 2733      2845
: 2734      2846
: 2735      2847
: 2736      2848
: 2737      2849
: 2738      2850
: 2739      2851
: 2740      2852
: 2741      2853
: 2742      2854
: 2743      2855
: 2744      2856
: 2745      2857
: 2746      2858
: 2747      2859
: 2748      2860
: 2749      2861
: 2750      2862
: 2751      2863
: 2752      2864
: 2753      2865      2

[.VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_BITOFFS]:
    RETURN FALSE;

! If valspec is a value-spec-follows, is a complex address.
[.VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VS_FOLLOWS]:
    RETURN FALSE;

! Now look at the VALKIND field for VFLAG non-above.
[.VALSPEC[DST$V_VS_VALKIND] EQL DST$K_VALKIND_LITERAL]:
    RETURN FALSE;

[.VALSPEC[DST$V_VS_VALKIND] EQL DST$K_VALKIND_DESC]:
    RETURN FALSE;

! If valkind is valkind_reg, then check and see if the value
! field is equal to regnum. If so, then return the valspec, as
! the symbol is bound to this register.
[.VALSPEC[DST$V_VS_VALKIND] EQL DST$K_VALKIND_REG]:
    BEGIN
    IF .VALSPEC[DST$L_VS_VALUE] NEQ .REGNUM
    THEN
        RETURN FALSE;

    RETURN .VALSPEC;
    END;

[.VALSPEC[DST$V_VS_VALKIND] EQL DST$K_VALKIND_ADDR]:

    ! If there is a register displacement, check to see if the regnum
    ! field is equal to regnum. If so, return valspec.
    BEGIN
    IF .VALSPEC[DST$V_VS_DISP]
    THEN
        BEGIN
        IF .VALSPEC[DST$V_VS_REGNUM] NEQ .REGNUM
        THEN
            RETURN FALSE;

        RETURN .VALSPEC;
        END;
    END;

    END;

    TES;
END;

! Otherwise, just return false.
```



```

: 2754
: 2755
: 2756
2866 2
2867 2
2868 1
! RETURN FALSE;
END;

```

		000C 00000 IS_SYMBOL_BOUND_TO_REG:					
		53	00000000G	00	9E 00002	.WORD Save R2,R3	2695
		5E		08	C2 00009	MOVAB LIB\$SIGNAL, R3	
	09 00000000G	00		01	E1 0000C	SUBL2 #8, SP	2723
			000280E8	8F	DD 00014	BBC #1, DBG\$GV_CONTROL+1, 1\$	
		63		01	FB 0001A	PUSHL #164072	
		50	04	AC	D0 0001D	CALLS #1, LIB\$SIGNAL	2731
		06	14	A0	91 00021	MOVL SYMID, R0	
				09	13 00025	CMPB 20(R0), #6	
		0A	14	A0	91 00027	BEQL 2\$	2732
				03	13 0002B	CMPB 20(R0), #10	
				0285	31 0002D	BEQL 2\$	
		52	0C	A0	D0 00030	BRW 10\$	2739
	FF 8F	00	01	A2	8F 00034	MOVL 12(R0), DSTPTR	2744
020B				0211	0003A	CASEB 1(DSTPTR), #0, #255	
020B				020B	00042	.WORD 5\$-3\$,-	
020B				020B	0004A	4\$-3\$,-	
020B				020B	00052	4\$-3\$,-	
020B				020B	0005A	4\$-3\$,-	
020B				020B	00062	4\$-3\$,-	
020B				020B	0006A	4\$-3\$,-	
020B				020B	00072	4\$-3\$,-	
020B				020B	0007A	4\$-3\$,-	
027B				020B	00082	4\$-3\$,-	
027B				027B	0008A	4\$-3\$,-	
027B				027B	00092	4\$-3\$,-	
027B				027B	0009A	4\$-3\$,-	
027B				027B	000A2	4\$-3\$,-	
027B				027B	000AA	4\$-3\$,-	
027B				027B	000B2	4\$-3\$,-	
027B				027B	000BA	4\$-3\$,-	
027B				027B	000C2	4\$-3\$,-	
027B				027B	000CA	4\$-3\$,-	
027B				027B	000D2	4\$-3\$,-	
027B				027B	000DA	4\$-3\$,-	
027B				027B	000E2	4\$-3\$,-	
027B				027B	000EA	4\$-3\$,-	
027B				027B	000F2	4\$-3\$,-	
027B				027B	000FA	4\$-3\$,-	
027B				027B	00102	4\$-3\$,-	
027B				027B	0010A	4\$-3\$,-	
027B				027B	00112	4\$-3\$,-	
027B				027B	0011A	4\$-3\$,-	
027B				027B	00122	4\$-3\$,-	
027B				027B	0012A	4\$-3\$,-	
027B				027B	00132	4\$-3\$,-	
027B				027B	0013A	4\$-3\$,-	
027B				027B	00142	4\$-3\$,-	

H 9
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

Page 89
(22)

108-38,-

DBGSYMBLZ
V04-000

16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 90
(22)[illegible]

.....

		51	02	A2	9A	0024F	MOVZBL	2(DSTPTR), R1	: 2765
		51	03	A142	9E	00253	MOVAB	3(R1)[DSTPTR], BLITRLR	: 2766
	01	AE		61	D0	00258	MOVL	(BLITRLR), BLIVALSPEC+1	: 2767
		50		6E	9E	0025C	MOVAB	BLIVALSPEC, VALSPEC	: 2788
	FB	8F		60	91	0025F	6\$: CMPB	(VALSPEC), #251	: 2789
				0B	12	00263	BNEQ	7\$	: 2797
	51	50	01	A0	C1	00265	ADDL3	1(VALSPEC), VALSPEC, R1	: 2803
		50	05	A1	9E	0026A	MOVAB	5(R1), VALSPEC	: 2809
				EF	11	0026E	BRB	6\$	: 2815
	80	8F		60	91	00270	7\$: CMPB	(VALSPEC), #128	: 2821
				3F	13	00274	BEQL	10\$	: 2825
	FA	8F		60	91	00276	CMPB	(VALSPEC), #250	: 2833
				39	13	0027A	BEQL	10\$	: 2835
	FF	8F		60	91	0027C	CMPB	(VALSPEC), #255	: 2843
				33	13	00280	BEQL	10\$	: 2850
	FD	8F		60	91	00282	CMPB	(VALSPEC), #253	: 2853
				2D	13	00286	BEQL	10\$	: 2868
		03		60	93	00288	BITB	(VALSPEC), #3	: 2868
				28	13	0028B	BEQL	10\$	: 2868
02	60	02		00	ED	0028D	CMPZV	#0, #2, (VALSPEC), #2	: 2868
03	60	02		21	13	00292	BEQL	10\$	: 2868
				00	ED	00294	CMPZV	#0, #2, (VALSPEC), #3	: 2868
				07	12	00299	BNEQ	8\$	: 2868
	08	AC	01	A0	D1	0029B	CPL	1(VALSPEC), REGNUM	: 2868
				11	11	002A0	BRB	9\$	: 2868
01	60	02		00	ED	002A2	8\$: CMPZV	#0, #2, (VALSPEC), #1	: 2868
				0C	12	002A7	BNEQ	10\$	: 2868
	08	60		03	E1	002A9	BBC	#3, (VALSPEC), 10\$	: 2868
08	AC	04		04	ED	002AD	CMPZV	#4, #4, (VALSPEC), REGNUM	: 2868
				02	13	002B3	9\$: BEQL	11\$	: 2868
				50	D4	002B5	10\$: CLRL	R0	: 2868
				04	002B7	11\$: RET			: 2868

; Routine Size: 696 bytes, Routine Base: DBG\$CODE + 1105


```
2758 2869 1 ROUTINE IS_SYMBOL_PSECT(RSTPTR) =
2759 2870 1
2760 2871 1 FUNCTION
2761 2872 1 This routine determines whether a given symbol is a PSECT or not.
2762 2873 1 It accepts as input a SYMID (the given symbol's RST pointer), and it
2763 2874 1 returns TRUE if that SYMID is the SYMID of a PSECT; it returns FALSE
2764 2875 1 otherwise. PSECTs are represented by Label RST Entries whose DST
2765 2876 1 pointer points to a PSECT DST Record.
2766 2877 1
2767 2878 1 PSECT names are less desirable symbolizations of addresses than any
2768 2879 1 other kind of label.
2769 2880 1
2770 2881 1 INPUTS
2771 2882 1 RSTPTR - A pointer to the RST entry of the symbol whose identity as
2772 2883 1 a PSECT is to be established.
2773 2884 1
2774 2885 1 OUTPUTS
2775 2886 1 The routine returns TRUE if the RSTPTR symbol is a PSECT. It returns
2776 2887 1 FALSE otherwise.
2777 2888 1
2778 2889 1
2779 2890 2 BEGIN
2780 2891 2
2781 2892 2 MAP
2782 2893 2 RSTPTR: REF RST$ENTRY; ! Pointer to symbol's RST entry
2783 2894 2
2784 2895 2 LOCAL
2785 2896 2 DSTPTR: REF DST$RECORD; ! Pointer to symbol's DST record
2786 2897 2
2787 2898 2
2788 2899 2
2789 2900 2 ! If this is a Label RST Entry, see if its DST pointer points to a PSECT
2790 2901 2 ! DST record. If so, return TRUE.
2791 2902 2
2792 2903 2 IF .RSTPTR[RST$B_KIND] EQL RST$K_LABEL
2793 2904 2 THEN
2794 2905 3 BEGIN
2795 2906 3 DSTPTR = .RSTPTR[RST$L_DSTPTR];
2796 2907 3 IF .DSTPTR[DST$B_TYPE] EQL DST$K_PSECT THEN RETURN TRUE;
2797 2908 2 END;
2798 2909 2
2799 2910 2
2800 2911 2 ! It is not a PSECT--return FALSE.
2801 2912 2
2802 2913 2 RETURN FALSE;
2803 2914 2
2804 2915 1 END;
```

```
0000 00000 IS_SYMBOL_PSECT:
50 04 AC D0 00002 .WORD Save nothing
04 14 AO 91 00006 MOVL RSTPTR, R0
OF 12 0000A CMPB 20(R0), #4
BNEQ 1$
```

```
: 2869
: 2903
:
```


DBGSYMBLZ
V04-000

M 9
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 94
(23)

B8	50	0C	A0	D0	0000C	MOVL	12(R0), DSTPTR
	8F	01	A0	91	00010	CMPB	1(DSTPTR), #184
			04	12	00015	BNEQ	1\$
	50		01	D0	00017	MOVL	#1, R0
				04	0001A	RET	
			50	D4	0001B	CLRL	R0
				04	0001D	RET	

: 2906
: 2907
:
:
: 2913
: 2915

; Routine Size: 30 bytes, Routine Base: DBG\$CODE + 138D

; 2805 2916 1


```

: 2807 2917 1 ROUTINE PRINT_REGNAME (ADDRESS): NOVALUE =
: 2808 2918 1
: 2809 2919 1 FUNCTION
: 2810 2920 1 This routine accepts an address in register address area and gets a
: 2811 2921 1 register descriptor by calling DBG$STA_ADDRESS_TO_REGDESCR. Then
: 2812 2922 1 converts register descriptor into a counted ASCII name.
: 2813 2923 1
: 2814 2924 1
: 2815 2925 1 INPUTS
: 2816 2926 1 ADDRESS - The input address which is a register address.
: 2817 2927 1
: 2818 2928 1 OUTPUTS
: 2819 2929 1 None.
: 2820 2930 1
: 2821 2931 1
: 2822 2932 2 BEGIN
: 2823 2933 2
: 2824 2934 2 LOCAL
: 2825 2935 2 NAMEPTR: REF VECTOR[BYTE], ! Pointer to register name
: 2826 2936 2 REGDESCR: DBG$REGDESCR; ! Register Descriptor
: 2827 2937 2
: 2828 2938 2
: 2829 2939 2 REGDESCR = DBG$STA_ADDRESS_TO_REGDESCR(.ADDRESS);
: 2830 2940 2 NAMEPTR = DBG$STA_REGISTER_NAME(.REGDESCR);
: 2831 2941 2 DBG$PRINT(UPBIT BYTE(%ASCII '!AC'), .NAMEPTR);
: 2832 2942 2 RETURN 0;
: 2833 2943 1 END;

```

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

43 41 21 03 0038D P.ADQ: .ASCII <3>\!AC\

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

```

0000 00000 PRINT_REGNAME:
00000000G 00 04 AC DD 00002 .WORD Save nothing
00000000G 00 01 FB 00005 PUSHL ADDRESS
00000000G 00 50 DD 0000C CALLS #1, DBG$STA_ADDRESS_TO_REGDESCR
00000000G 00 01 FB 0000E PUSHL REGDESCR
00000000G 00 50 DD 00015 CALLS #1, DBG$STA_REGISTER_NAME
00000000G 00 00000000' EF 9F 00017 PUSHL NAMEPTR
00000000G 00 02 FB 0001D PUSHAB P.ADQ
00000000G 00 04 00024 CALLS #2, DBG$PRINT
RET

```

```

: 2917
: 2939
: 2940
: 2941
: 2943

```

; Routine Size: 37 bytes, Routine Base: DBG\$CODE + 13DB


```
2835 2944 1 ROUTINE SEARCH_MODULE_SAT (PROG_SATPTR, ADDRESS, P_MODRSTPTR, P_RSTPTR,  
2836 2945 1 P_BYTE_OFFSET, PRINT_FLAG): NOVALUE =  
2837 2946 1  
2838 2947 1 FUNCTION  
2839 2948 1 This routine searches a set module's SAT to determine what static  
2840 2949 1 symbol most immediately precedes the given address. The routine  
2841 2950 1 returns the module RST pointer, the symbol RST pointer, and the byte  
2842 2951 1 offset.  
2843 2952 1  
2844 2953 1 INPUTS  
2845 2954 1 ADDRESS - The 32 bit address we want to symbolize.  
2846 2955 1  
2847 2956 1 PROG_SATPTR - The pointer into the program level SAT.  
2848 2957 1  
2849 2958 1 PRINT_FLAG - Flag to indicate the symbol to be outputed, in  
2850 2959 1 here we print the symbols were not picked up  
2851 2960 1 by this routine which all has the same address.  
2852 2961 1  
2853 2962 1 OUTPUTS  
2854 2963 1 P_MODRSTPTR - The module RST entry for the symbol if one is found.  
2855 2964 1  
2856 2965 1 P_RSTPTR - The RST entry for the symbol if a symbolization is  
2857 2966 1 possible.  
2858 2967 1  
2859 2968 1 P_BYTE_OFFSET - The byte offset.  
2860 2969 1  
2861 2970 1  
2862 2971 2 BEGIN  
2863 2972 2  
2864 2973 2 BIND  
2865 2974 2 BYTE_OFFSET = .P_BYTE_OFFSET, ! Byte offset.  
2866 2975 2 RSTPTR = .P_RSTPTR : REF RST$ENTRY, ! The symbol's RST pointer.  
2867 2976 2 MODRSTPTR = .P_MODRSTPTR : REF RST$ENTRY; ! The module RST pointer.  
2868 2977 2  
2869 2978 2 MAP  
2870 2979 2 PROG_SATPTR : REF SAT$ENTRY; ! The program's SAT pointer.  
2871 2980 2  
2872 2981 2 LOCAL  
2873 2982 2 BEST_SAT : REF SAT$ENTRY,  
2874 2983 2 NAME : REF VECTOR[BYTE], ! Pointer to ASCII string  
2875 2984 2 NEXT_SATPTR : REF SAT$ENTRY, ! The next SAT ptr in the chain.  
2876 2985 2 SATPTR : REF SAT$ENTRY, ! The module's SAT pointer.  
2877 2986 2 SATRST : REF RST$ENTRY, ! Pointer RST entry in SAT  
2878 2987 2 TMPST : REF SAT$ENTRY; ! Temp. SAT.  
2879 2988 2  
2880 2989 2  
2881 2990 2 ! Initialize module RST pointer, as this is used to indicate the success  
2882 2991 2 of this routine.  
2883 2992 2  
2884 2993 2 $ABORT ON_CONTROL_Y;  
2885 2994 2 MODRSTPTR = 0;  
2886 2995 2 RSTPTR = 0;  
2887 2996 2  
2888 2997 2  
2889 2998 2 MODRSTPTR = .PROG_SATPTR[SAT$RSTPTR];  
2890 2999 2 OWN_MODULE_BEGIN = .PROG_SATPTR[SAT$START];  
2891 3000 2 OWN_MODULE_END = .PROG_SATPTR[SAT$END];
```



```

: 2892      3001  2      IF NOT .MODRSTPTR[RST$V_MODSET] THEN RETURN 0;
: 2893      3002  2
: 2894      3003  2
: 2895      3004  2      ! If the module is set, search through the module's SAT looking for
: 2896      3005  2      ! a symbol whose address matches the input.
: 2897      3006  2
: 2898      3007  2      SATPTR = .MODRSTPTR[RST$L_SAT_PTR];
: 2899      3008  2      IF .SATPTR[SAT$L_START] GEQA .ADDRESS THEN RETURN 0;
: 2900      3009  2
: 2901      3010  2
: 2902      3011  2      ! Determine the address range.
: 2903      3012  2
: 2904      3013  2      BEST_SAT = 0;
: 2905      3014  2      TMP$AT = 0;
: 2906      3015  2      NEXT_SATPTR = .SATPTR[SAT$L_FLINK];
: 2907      3016  2      WHILE .NEXT_SATPTR NEQ 0 DO
: 2908      3017  3          BEGIN
: 2909      3018  3              $ABORT ON CONTROL Y;
: 2910      3019  3              IF .SATPTR[SAT$L_END] GEQA .ADDRESS
: 2911      3020  3              THEN
: 2912      3021  4                  BEGIN
: 2913      3022  4                      IF NOT IS_SYMBOL_PSECT(.SATPTR[SAT$L_RSTPTR])
: 2914      3023  4                      THEN
: 2915      3024  5                          BEGIN
: 2916      3025  5                              IF .TMP$AT NEQ 0 AND
: 2917      3026  5                              .ADDRESS LEQ .TMP$AT[SAT$L_END]
: 2918      3027  5                              THEN
: 2919      3028  6                                  BEGIN
: 2920      3029  6                                      BEST_SAT = .TMP$AT;
: 2921      3030  6                                      TMP$AT = 0;
: 2922      3031  6
: 2923      3032  6                      ! If the print_flag is set, output the symbol we did not
: 2924      3033  6                      ! choose.
: 2925      3034  6                      IF .PRINT_FLAG
: 2926      3035  6                      THEN
: 2927      3036  6                          BEGIN
: 2928      3037  6
: 2929      3038  7                              ! Build the ASCII string for the symbol.
: 2930      3039  7
: 2931      3040  7                              DBG$PRINT (UPLIT BYTE (%ASCIC 'address !XL'), .ADDRESS);
: 2932      3041  7                              DBG$PRINT (UPLIT BYTE (%ASCIC ': '));
: 2933      3042  7                              DBG$NEWLINE();
: 2934      3043  7                              DBG$PRINT_CONTROL (DBG$K_PRTSET LMARGIN, TAB);
: 2935      3044  7                              DBG$PRINT_SYMBOL_PATHNAME(.SATPTR[SAT$L_RSTPTR]);
: 2936      3045  7                              BYTE_OFFSET = .ADDRESS - .SATPTR[SAT$L_START];
: 2937      3046  7                              IF .BYTE_OFFSET NEQ 0
: 2938      3047  7                              THEN
: 2939      3048  7                                  DBG$PRINT_OFFSET (.BYTE_OFFSET);
: 2940      3049  7                                  DBG$NEWLINE();
: 2941      3050  7                                  DBG$PRINT_CONTROL (DBG$K_PRT_RESET);
: 2942      3051  7                                  END;
: 2943      3052  7
: 2944      3053  7                              END;
: 2945      3054  6
: 2946      3055  6      END
: 2947      3056  6
: 2948      3057  5      ELSE
```



```

: 2949      3058 5
: 2950      3059 4
: 2951      3060 3
: 2952      3061 3
: 2953      3062 3
: 2954      3063 3
: 2955      3064 3
: 2956      3065 3
: 2957      3066 3
: 2958      3067 3
: 2959      3068 3
: 2960      3069 3
: 2961      3070 3
: 2962      3071 3
: 2963      3072 3
: 2964      3073 4
: 2965      3074 3
: 2966      3075 4
: 2967      3076 4
: 2968      3077 4
: 2969      3078 4
: 2970      3079 4
: 2971      3080 5
: 2972      3081 6
: 2973      3082 5
: 2974      3083 5
: 2975      3084 5
: 2976      3085 5
: 2977      3086 5
: 2978      3087 5
: 2979      3088 5
: 2980      3089 6
: 2981      3090 6
: 2982      3091 6
: 2983      3092 7
: 2984      3093 7
: 2985      3094 7
: 2986      3095 7
: 2987      3096 7
: 2988      3097 7
: 2989      3098 7
: 2990      3099 7
: 2991      3100 7
: 2992      3101 7
: 2993      3102 7
: 2994      3103 7
: 2995      3104 7
: 2996      3105 7
: 2997      3106 7
: 2998      3107 7
: 2999      3108 6
: 3000      3109 6
: 3001      3110 5
: 3002      3111 5
: 3003      3112 4
: 3004      3113 4
: 3005      3114 3

      BEST_SAT = .SATPTR;
      END;
    END;
  IF .NEXT_SATPTR[SAT$L_START] GTRA .ADDRESS
  THEN
    EXITLOOP;

    ! In bliss, BIND user_meaningful_name = PLIT (....), in this case
    ! user_meaningful_name == P.AAA. So the choice should not be P.AAA.
    ! Here is the code to check for this kind of cases. Print both
    ! name, the best symbolization one will be printed from the caller.
    IF (.SATPTR[SAT$L_START] EQL .NEXT_SATPTR[SAT$L_START]) AND
      (.SATPTR[SAT$L_END] EQL .NEXT_SATPTR[SAT$L_END])
    THEN
      BEGIN
        SATRST = .SATPTR[SAT$L_RSTPTR];
        NAME = DBG$GET_DST_NAME(.SATRST[RST$L_DSTPTR]);
        IF .NAME[0] NEQ 0
        THEN
          BEGIN
            IF NOT (.NAME[1] EQL %C'P' AND .NAME[2] EQL %C'.')
            THEN
              TMPSAT = .SATPTR

              ! Print P.AAA, which we did not choose.
            ELSE
              BEGIN
                IF .PRINT_FLAG
                THEN
                  BEGIN

                    ! Build the ASCII string for the symbol.
                    !
                    DBG$PRINT (UPLIT BYTE (%ASCII 'address !XL'), .ADDRESS);
                    DBG$PRINT (UPLIT BYTE (%ASCII ': '));
                    DBG$NEWLINE();
                    DBG$PRINT_CONTROL (DBG$K_PRTSET LMARGIN, TAB);
                    DBG$PRINT_SYMBOL_PATHNAME(.SATPTR[SAT$L_RSTPTR]);
                    BYTE_OFFSET = .SATPTR[SAT$L_START];
                    IF .BYTE_OFFSET NEQ 0
                    THEN
                      DBG$PRINT_OFFSET (.BYTE_OFFSET);
                    DBG$NEWLINE();
                    DBG$PRINT_CONTROL (DBG$K_PRT_RESET);
                  END;
                END;
              END;
            END;
          END;
        END;
      END;
    END;
  END;
END;
```



```

3006      3115 3
3007      3116 3
3008      3117 3
3009      3118 3
3010      3119 3
3011      3120 3
3012      3121 3
3013      3122 3
3014      3123 3
3015      3124 3
3016      3125 3
3017      3126 3
3018      3127 4
3019      3128 3
3020      3129 4
3021      3130 4
3022      3131 4
3023      3132 4
3024      3133 4
3025      3134 4
3026      3135 4
3027      3136 3
3028      3137 3
3029      3138 3
3030      3139 3
3031      3140 3
3032      3141 3
3033      3142 3
3034      3143 3
3035      3144 3
3036      3145 3
3037      3146 3
3038      3147 3
3039      3148 3
3040      3149 3
3041      3150 3
3042      3151 3
3043      3152 3
3044      3153 3
3045      3154 3
3046      3155 3
3047      3156 3
3048      3157 3
3049      3158 3
3050      3159 3
3051      3160 3
3052      3161 3
3053      3162 3
3054      3163 3
3055      3164 4
3056      3165 4
3057      3166 4
3058      3167 4
3059      3168 4
3060      3169 4
3061      3170 5
3062      3171 5

      SATPTR = .NEXT_SATPTR;
      NEXT_SATPTR = .SATPTR[SAT$L_FLINK];
      END;

      ! Check for special cases.
      !
      IF .NEXT_SATPTR EQL 0
      THEN
      BEGIN
      IF (.ADDRESS GEQA .SATPTR[SAT$L_START]) AND
      (.ADDRESS LEQA .SATPTR[SAT$L_END])
      THEN
      BEGIN
      IF .TMPSAT NEQ 0 AND
      .ADDRESS LEQ .TMPSAT[SAT$L_END]
      THEN
      BEST_SAT = .TMPSAT
      ELSE
      BEST_SAT = .SATPTR;
      END;
      END;

      IF .BEST_SAT EQL 0 THEN BEST_SAT = .SATPTR;

      ! We have found the best sat, get the symbol's RST pointer and determine
      ! the maximum byte offset.
      !
      RSTPTR = .BEST_SAT[SAT$L_RSTPTR];
      BYTE_OFFSET = .ADDRESS - .BEST_SAT[SAT$L_START];
      CASE .RSTPTR[RST$B_KIND] FROM RST$K_KIND_MINIMUM
      TO RST$K_KIND_MAXIMUM OF
      SET
      [RST$K_ROUTINE, RST$K_BLOCK, RST$K_LABEL, RST$K_ENTRY]:
      BEGIN

      ! See if the address is a label address. It much match the
      ! the label address exactly, or else we find the containing
      ! entity and print that plus line number as the better
      ! symbolization.
      !
      IF .RSTPTR[RST$B_KIND] EQL RST$K_LABEL OR
      .RSTPTR[RST$B_KIND] EQL RST$K_ENTRY
      THEN
      IF .BYTE_OFFSET NEQ 0
      THEN
      BEGIN
      LOCAL
      RSTPTR_TMP;

      RSTPTR_TMP = .RSTPTR;
      WHILE .RSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
      BEGIN
      RSTPTR = .RSTPTR[RST$L_UPSCOPEPTR];

```



```
RETURN 0;
END;
```

.PSECT DBGSPLIT,NOWRT, SHR, PIC,0

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

: 2944

56	14	AC	D0	00009	MOVL	P_BYTE_OFFSET, R6	2974	
5A	10	AC	D0	0000D	MOVL	P_RSTPTR, R10	2975	
0D 00000000G	00	01	E1	00011	BBC	#1, DBG\$GV_CONTROL+1, 1\$	2987	
00000000G	00	8F	DD	00019	PUSHL	#164072		
		01	FB	0001F	CALLS	#1, LIB\$SIGNAL		
	0C	BC	D4	00026	CLRL	@P_MODRSTPTR	2994	
		6A	D4	00029	CLRL	(R10)	2995	
	50	AC	D0	0002B	MOVL	PROG_SATPTR, R0	2998	
0C	04	A0	D0	0002F	MOVL	12(R0), @P_MODRSTPTR		
00000000*	BC	0C	A0	7D	MOVQ	4(R0), OWN_MODULE_BEGIN	2999	
	EF	04	A0	00034	MOVL	@P_MODRSTPTR, R0	3001	
	50	0C	BC	D0	0003C	BLBS	40(R0), 2\$	
	01	28	A0	E8	00040	RET		
				04	00044			
	52	18	A0	D0	00045	MOVL	24(R0), SATPTR	3007
	57	08	AC	D0	00049	MOVL	ADDRESS, R7	3008
	57	04	A2	D1	0004D	CMPL	4(SATPTR), R7	
			01	1B	00051	BLEQU	3\$	
				04	00053	RET		
			54	7C	00054	CLRQ	TMPSAT	3014
	53		62	D0	00056	MOVL	(SATPTR), NEXT_SATPTR	3015
			03	12	00059	BNEQ	5\$	3016
			0128	31	0005B	BRW	16\$	
0D 00000000G	00	01	E1	0005E	BBC	#1, DBG\$GV_CONTROL+1, 6\$	3017	
00000000G	00	8F	DD	00066	PUSHL	#164072		
	57	08	01	FB	0006C	CALLS	#1, LIB\$SIGNAL	
			A2	D1	00073	CMPL	8(SATPTR), R7	3019
			74	1F	00077	BLSSU	9\$	
	0C		A2	DD	00079	PUSHL	12(SATPTR)	3022
FF3C	CF		01	FB	0007C	CALLS	#1, IS_SYMBOL_PSECT	
	69		50	E8	00081	BLBS	R0, 9\$	
			54	D5	00084	TSTL	TMPSAT	3025
			62	13	00086	BEQL	8\$	
08	A4		57	D1	00088	CMPL	R7, 8(TMPSAT)	3026
			5C	14	0008C	BGTR	8\$	
	55		54	D0	0008E	MOVL	TMPSAT, BEST_SAT	3029
			54	D4	00091	CLRL	TMPSAT	3030
	56	18	AC	E9	00093	BLBC	PRINT_FLAG, 9\$	3036
			57	DD	00097	PUSHL	R7	3043
			5B	DD	00099	PUSHL	R11	
00000000G	00		02	FB	0009B	CALLS	#2, DBG\$PRINT	
		0C	AB	9F	000A2	PUSHAB	P.ADS	3044
00000000G	00		01	FB	000A5	CALLS	#1, DBG\$PRINT	
00000000G	00		00	FB	000AC	CALLS	#0, DBG\$NEWLINE	3045
			04	DD	000B3	PUSHL	#4	3046
			01	DD	000B5	PUSHL	#1	
00000000G	00		02	FB	000B7	CALLS	#2, DBG\$PRINT_CONTROL	
		0C	A2	DD	000BE	PUSHL	12(SATPTR)	3047
00000000G	00		01	FB	000C1	CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME	
66	57	04	A2	C3	000C8	SUBL3	4(SATPTR), R7, (R6)	3048
			09	13	000CD	BEQL	7\$	3049
			66	DD	000CF	PUSHL	(R6)	3051
00000000G	00		01	FB	000D1	CALLS	#1, DBG\$PRINT_OFFSET	
00000000G	00		00	FB	000D8	CALLS	#0, DBG\$NEWLINE	3052
			05	DD	000DF	PUSHL	#5	3053
00000000G	00		01	FB	000E1	CALLS	#1, DBG\$PRINT_CONTROL	
			03	11	000E8	BRB	9\$	3025
	55		52	D0	000EA	MOVL	SATPTR, BEST_SAT	3058

57	04	A3	D1	000ED	9\$:	CMPL	4(NEXT_SATPTR), R7	3062	
		03	1B	000F1		BLEQU	10\$		
		0090	31	000F3		BRW	16\$		
04	A3	04	A2	D1	000F6	10\$:	CMPL	4(SATPTR), 4(NEXT_SATPTR)	3072
			2C	12	000FB		BNEQ	12\$	
08	A3	08	A2	D1	000FD		CMPL	8(SATPTR), 8(NEXT_SATPTR)	3073
			7C	12	00102		BNEQ	15\$	
59		0C	A2	D0	00104		MOVL	12(SATPTR), SATRST	3076
		0C	A9	DD	00108		PUSHL	12(SATRST)	3077
00000000G	00		01	FB	0010B		CALLS	#1, DBG\$GET_DST_NAME	
	58		50	D0	00112		MOVL	R0, NAME	
			68	95	00115		TSTB	(NAME)	3078
			67	13	00117		BEQL	15\$	
50	8F	01	A8	91	00119		CMPB	1(NAME), #80	3081
			06	12	0011E		BNEQ	11\$	
	2E	02	A8	91	00120		CMPB	2(NAME), #46	
			05	13	00124		BEQL	13\$	
	54		52	D0	00126	11\$:	MOVL	SATPTR, TMPSAT	3083
			55	11	00129	12\$:	BRB	15\$	
	51	18	AC	E9	0012B	13\$:	BLBC	PRINT_FLAG, 15\$	3090
			57	DD	0012F		PUSHL	R7	3097
		0F	AB	9F	00131		PUSHAB	P.ADT	
00000000G	00		02	FB	00134		CALLS	#2, DBG\$PRINT	
		1B	AB	9F	0013B		PUSHAB	P.ADU	3098
00000000G	00		01	FB	0013E		CALLS	#1, DBG\$PRINT	
00000000G	00		00	FB	00145		CALLS	#0, DBG\$NEWLINE	3099
			04	DD	0014C		PUSHL	#4	3100
			01	DD	0014E		PUSHL	#1	
00000000G	00		02	FB	00150		CALLS	#2, DBG\$PRINT_CONTROL	
		0C	A2	DD	00157		PUSHL	12(SATPTR)	3101
00000000G	00		01	FB	0015A		CALLS	#1, DBG\$PRINT_SYMBOL_PATHNAME	
	66	04	A2	D0	00161		MOVL	4(SATPTR), (R6)	3102
			09	13	00165		BEQL	14\$	3103
			66	DD	00167		PUSHL	(R6)	3105
00000000G	00		01	FB	00169		CALLS	#1, DBG\$PRINT_OFFSET	
00000000G	00		00	FB	00170	14\$:	CALLS	#0, DBG\$NEWLINE	3106
			05	DD	00177		PUSHL	#5	3107
00000000G	00		01	FB	00179		CALLS	#1, DBG\$PRINT_CONTROL	
	52		53	D0	00180	15\$:	MOVL	NEXT_SATPTR, SATPTR	3116
		FED0	31	00183		BRW	4\$		3117
			53	D5	00186	16\$:	TSTL	NEXT_SATPTR	3123
			1E	12	00188		BNEQ	18\$	
04	A2		57	D1	0018A		CMPL	R7, 4(SATPTR)	3126
			18	1F	0018E		BLSSU	18\$	
08	A2		57	D1	00190		CMPL	R7, 8(SATPTR)	3127
			12	1A	00194		BGTRU	18\$	
			54	D5	00196		TSTL	TMPSAT	3130
			0B	13	00198		BEQL	17\$	
08	A4		57	D1	0019A		CMPL	R7, 8(TMPSAT)	3131
			05	14	0019E		BGTR	17\$	
	55		54	D0	001A0		MOVL	TMPSAT, BEST_SAT	3133
			03	11	001A3		BRB	18\$	
	55		52	D0	001A5	17\$:	MOVL	SATPTR, BEST_SAT	3135
			55	D5	001A8	18\$:	TSTL	BEST_SAT	3139
			03	12	001AA		BNEQ	19\$	
	55		52	D0	001AC		MOVL	SATPTR, BEST_SAT	
6A		0C	A5	D0	001AF	19\$:	MOVL	12(BEST_SAT), (R10)	3145

	66	57	04	A5	C3	001B3	SUBL3	4(BEST_SAT), R7, (R6)	: 3146
	OD	52		6A	DO	001B8	MOVL	(R10), R2	: 3147
0021	0021	00	14	A2	8F	001BB	CASEB	20(R2), #0, #13	
001C	006E	001C		001C		001C0	.WORD	21\$-20\$,-	
001C	001C	001C		0021		001C8		21\$-20\$,-	
		001C		0021		001D0		22\$-20\$,-	
		001C		001C		001D8		22\$-20\$,-	
								22\$-20\$,-	
								21\$-20\$,-	
								28\$-20\$,-	
								21\$-20\$,-	
								22\$-20\$,-	
								21\$-20\$,-	
								21\$-20\$,-	
								21\$-20\$,-	
								21\$-20\$,-	
								21\$-20\$,-	
								21\$-20\$,-	
			4E	AB	9F	001DC	21\$:	PUSHAB	P.ADW
				3E	11	001DF		BRB	27\$
		04	14	A2	91	001E1	22\$:	CMPB	20(R2), #4
				06	13	001E5		BEQL	23\$
		08	14	A2	91	001E7		CMPB	20(R2), #8
				41	12	001EB		BNEQ	28\$
				66	D5	001ED	23\$:	TSTL	(R6)
				3D	13	001EF		BEQL	28\$
		51		52	DO	001F1		MOVL	R2, RSTPTR_TMP
		50		6A	DO	001F4	24\$:	MOVL	(R10), R0
		01	14	A0	91	001F7		CMPB	20(R0), #1
				0D	13	001FB		BEQL	25\$
		6A	10	A0	DO	001FD		MOVL	16(R0), (R10)
		50		6A	DO	00201		MOVL	(R10), R0
		02	14	A0	91	00204		CMPB	20(R0), #2
				EA	12	00208		BNEQ	24\$
		50		6A	DO	0020A	25\$:	MOVL	(R10), R0
		01	14	A0	91	0020D		CMPB	20(R0), #1
				1B	12	00211		BNEQ	28\$
			29	A0	95	00213		TSTB	41(R0)
				04	12	00216		BNEQ	26\$
		6A		51	DO	00218		MOVL	RSTPTR_TMP, (R10)
					04	0021B		RET	
			1E	AB	9F	0021C	26\$:	PUSHAB	P.ADV
				01	DD	0021F	27\$:	PUSHL	#1
				8F	DD	00221		PUSHL	#164706
				03	FB	00227		CALLS	#3, LIB\$SIGNAL
				04	0022E	28\$:	RET		: 3203

; Routine Size: 559 bytes, Routine Base: DBG\$CODE + 1400

; 3095 3204 1


```
3097 3205 1 ROUTINE BUILD_PRIMARY_SUBNODE(PRIMPTR, SYMID, BIT_OFFSET) =
3098 3206 1
3099 3207 1 FUNCTION
3100 3208 1     This routine is a recursive routine to be called to build the
3101 3209 1     Primary subnode. For Array or Record type, we need to do
3102 3210 1     more work to symbolize down to record component or array element
3103 3211 1     level. For other type, return the primary from this routine.
3104 3212 1
3105 3213 1 INPUTS
3106 3214 1     PRIMPTR - Pointer to Primary.
3107 3215 1
3108 3216 1     SYMID   - Pointer to SYMID.
3109 3217 1
3110 3218 1     BIT_OFFSET - Bit offset to the given object.
3111 3219 1
3112 3220 1 OUTPUTS
3113 3221 1     Return value is the pointer to Primary. And Bit offsets to the
3114 3222 1     primary is passed back to the caller. (BIT_OFFSET updated accordingly).
3115 3223 1
3116 3224 1
3117 3225 2 BEGIN
3118 3226 2
3119 3227 2 MAP
3120 3228 2     BIT_OFFSET: REF VECTOR[.LONG], ! Bit offset to Primary
3121 3229 2     PRIMPTR: REF DBG$PRIMARY,      ! Pointer to Primary
3122 3230 2     SYMID: REF RST$ENTRY;          ! Pointer to Symid
3123 3231 2
3124 3232 2 LOCAL
3125 3233 2     FCODE,                        ! Fcode of the symid to be build
3126 3234 2     KIND,                        ! Kind of this symid to be build
3127 3235 2     TYPEID: REF RST$ENTRY;        ! Pointer to TYPEID for the SYMID
3128 3236 2
3129 3237 2     DBG$GL_CURRENT_PRIMARY = .PRIMPTR; ! Update the current primary
3130 3238 2
3131 3239 2     FCODE = 0;
3132 3240 2     TYPEID = 0;
3133 3241 2     DBG$STA_SYMKIND (.SYMID, KIND);
3134 3242 2
3135 3243 2
3136 3244 2     ! Check to see if this is a data item, if it is get more inforamtion.
3137 3245 2
3138 3246 2     IF .KIND EQL RST$K_DATA OR
3139 3247 2         .KIND EQL RST$K_TYPCOMP OR
3140 3248 2         .KIND EQL RST$K_TYPE
3141 3249 2     THEN
3142 3250 2         DBG$STA_SYMTYPE(.SYMID, FCODE, TYPEID);
3143 3251 2
3144 3252 2
3145 3253 2     ! Call DBG$BUILD_PRIMARY_SUBNODE to build a subnode and fill in all of
3146 3254 2     ! the subnode information.
3147 3255 2
3148 3256 2     IF .KIND EQL RST$K_TYPE
3149 3257 2     THEN
3150 3258 2         BEGIN
3151 3259 2             KIND = RST$K_DATA;
3152 3260 2             SYMID = 0;
3153 3261 2         END;
```



```

: 3154      3262 2
: 3155      3263 2
: 3156      3264 2
: 3157      3265 2
: 3158      3266 2
: 3159      3267 2
: 3160      3268 2
: 3161      3269 2
: 3162      3270 2
: 3163      3271 2
: 3164      3272 2
: 3165      3273 2
: 3166      3274 2
: 3167      3275 2
: 3168      3276 2
: 3169      3277 2
: 3170      3278 2
: 3171      3279 2
: 3172      3280 2
: 3173      3281 2
: 3174      3282 2
: 3175      3283 2
: 3176      3284 2
: 3177      3285 2
: 3178      3286 2
: 3179      3287 2
: 3180      3288 2
: 3181      3289 2
: 3182      3290 2
: 3183      3291 1

DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, .KIND, .SYMID, .FCODE, .TYPEID, 0);

! Do more work for array and record component -- Symbolize down to
! the components + offset. For example, Array+15 --> Array(3)+3.
CASE .FCODE FROM RST$K_TYPE_MINIMUM TO RST$K_TYPE_MAXIMUM OF
SET
[RST$K_TYPE_ARRAY]:
IF .DBG$GL_ARRSUB_FLAG
THEN
PRIMPTR = SYMBOLIZE_ARRAY_ELEMENT(.PRIMPTR, .BIT_OFFSET)
ELSE
PRIMPTR[DBG$V_DHDR_AGGR] = FALSE;
[RST$K_TYPE_RECORD]:
IF .DBG$GL_RECCEMP_FLAG
THEN
PRIMPTR = SYMBOLIZE_RECORD_COMPONENT(.PRIMPTR, .BIT_OFFSET)
ELSE
PRIMPTR[DBG$V_DHDR_AGGR] = FALSE;
[INRANGE, OUTRANGE]:
0;
TES;
RETURN .PRIMPTR;
END;
```

```

                                0004 00000 BUILD_PRIMARY_SUBNODE:
                                .WORD  Save R2
                                5E      0C C2 00002      SUBL2  #12, SP
                                52      04 AC D0 00005      MOVL  PRIMPTR, R2
00000000G 00      04 52 D0 00009      MOVL  R2, DBG$GL_CURRENT_PRIMARY
                                AE 7C 00010      CLRQ  TYPEID
                                SE DD 00013      PUSHL  SP
                                08      AC DD 00015      PUSHL  SYMID
00000000G 00      02 FB 00018      CALLS  #2, DBG$STA_SYMKIND
06      6E D1 0001F      CMPL  KIND, #6
                                0A 13 00022      BEQL  1$
                                0A      6E D1 00024      CMPL  KIND, #10
                                07      05 13 00027      BEQL  1$
                                07      6E D1 00029      CMPL  KIND, #7
                                10 12 0002C      BNEQ  2$
                                04      AE 9F 0002E 1$:      PUSHAB TYPEID
                                0C      AE 9F 00031      PUSHAB FCODE
                                08      AC DD 00034      PUSHL  SYMID
00000000G 00      03 FB 00037      CALLS  #3, DBG$STA_SYMTYPE
07      6E D1 0003E 2$:      CMPL  KIND, #7
                                06 12 00041      BNEQ  3$
                                6E      06 D0 00043      MOVL  #6, KIND
                                : 3205
                                : 3237
                                : 3240
                                : 3241
                                : 3246
                                : 3247
                                : 3248
                                : 3250
                                : 3256
                                : 3259
```


[illegible]

; Routine Size: 198 bytes, Routine Base: DBG\$CODE + 162F

; 3184 3292 1


```
3186 3293 1 ROUTINE GET_RECORD_COMPONENT(PRIMPTR, COMPTR, BIT_OFFSET, SYMID, OFFSET, IDX) =
3187 3294 1
3188 3295 1 FUNCTION
3189 3296 1 This routine calculates the record component offsets to the record
3190 3297 1 address and then compares the value with the offset to the primary
3191 3298 1 we got so far. If the value we just calculated is better than the
3192 3299 1 one we got so far, choice this one instead. This routine is
3193 3300 1 recursive in a sense that if one of the record component is a
3194 3301 1 variant, we need to get a list of the variant components. Then
3195 3302 1 figure out the offsets for each variant and choose the better
3196 3303 1 variant for the given record component.
3197 3304 1
3198 3305 1 INPUTS
3199 3306 1 PRIMPTR - Pointer to Primary descriptor.
3200 3307 1
3201 3308 1 COMPTR - Pointer to the record component or variant component.
3202 3309 1
3203 3310 1 BIT_OFFSET - Offset value to the Primary.
3204 3311 1
3205 3312 1 SYMID - The symid we choose for the record component, or variant.
3206 3313 1
3207 3314 1 OFFSET - Offset value of the component.
3208 3315 1
3209 3316 1 IDX - Ith component index.
3210 3317 1
3211 3318 1 OUTPUTS
3212 3319 1 Return value is the pointer to the primary.
3213 3320 1 Note: The primary returned from this routine may not be in a stable
3214 3321 1 state. (The information in the root node may not be updated properly).
3215 3322 1 When next time around to build primary subnode, it will be updated
3216 3323 1 properly.
3217 3324 1
3218 3325 1
3219 3326 1 BEGIN
3220 3327 1
3221 3328 1 MAP
3222 3329 1 BIT_OFFSET: REF VECTOR[.LONG], | Desired offset value to Begin of
3223 3330 1 | the Record
3224 3331 1 COMPTR: REF RST$ENTRY, | Pointer to component RST
3225 3332 1 OFFSET: REF VECTOR[.LONG], | Offset Value to Begin of the Record
3226 3333 1 | for SYMID
3227 3334 1 PRIMPTR: REF DBG$PRIMARY, | Pointer to Primary Root Node
3228 3335 1 SYMID: REF VECTOR[.LONG]; | Pointer to desired RST
3229 3336 1
3230 3337 1 BUILTIN
3231 3338 1 REMQUE;
3232 3339 1
3233 3340 1 LOCAL
3234 3341 1 COMPVECPTR: REF VECTOR[.LONG], | Pointer to a list of variants
3235 3342 1 DUMMY, | Dummy as it is
3236 3343 1 KIND, | Data kind
3237 3344 1 SUBNODE: REF DBG$PRIM_NODE, | Last primary subnode
3238 3345 1 TAGID, | Variant Tag variable
3239 3346 1 TAG_NAME: REF VECTOR[.BYTE], | Variant Tag name
3240 3347 1 TAG_VAL, | Tag value
3241 3348 1 TMP_OFFSET, | Temporary offset
3242 3349 1 TMP_SYMID, | Temporary symid
```



```

3243 3350 2 VAL_DESC: REF DBG$VALDESC, ! Tag Value descriptor
3244 3351 2 VAL_KIND, ! Data value kind
3245 3352 2 VAL_PTR: VECTOR[3, LONG], ! Data value
3246 3353 2 VARIANT: REF RST$VAR_ENTRY; ! Variant
3247 3354 2
3248 3355 2
3249 3356 2
3250 3357 2 DBG$GL_CURRENT_PRIMARY = .PRIMPTR; ! Update the current primary
3251 3358 2 ! Set the context for the given RST entry.
3252 3359 2 !
3253 3360 2 DBG$STA_SETCONTEXT(.COMPTR);
3254 3361 2 DBG$STA_SYMKIND(.COMPTR, KIND);
3255 3362 2
3256 3363 2
3257 3364 2
3258 3365 2 ! Check to see if this is a variant.
3259 3366 2 !
3260 3367 2 IF .KIND EQL RST$K_VARIANT
3261 3368 2 THEN
3262 3369 2 BEGIN
3263 3370 2 VARIANT = 0;
3264 3371 2
3265 3372 2
3266 3373 2 ! Check to see if this variant has a tag variable. Also check to
3267 3374 2 ! see if the tag variable has a name.
3268 3375 2 !
3269 3376 2 IF (TAGID = .COMPTR[RST$L_VARTAGPTR]) NEQ 0
3270 3377 2 THEN
3271 3378 2 BEGIN
3272 3379 2 DBG$STA_SYMNAME(.TAGID, TAG_NAME);
3273 3380 2 IF .TAG_NAME[0] NEQ 0
3274 3381 2 THEN
3275 3382 2 BEGIN
3276 3383 2
3277 3384 2
3278 3385 2 ! Build a subnode for the tag variable.
3279 3386 2 !
3280 3387 2 PRIMPTR = BUILD_PRIMARY_SUBNODE(.PRIMPTR, .TAGID, .BIT_OFFSET);
3281 3388 2
3282 3389 2
3283 3390 2 ! Get the value for the tag variable.
3284 3391 2 !
3285 3392 2 DBG$PRIM_TO_VAL(.PRIMPTR, DBG$K_VALUE_DESC, VAL_DESC);
3286 3393 2 TAG_VAL = .VAL_DESC[DBG$L_VALUE_VALUE0];
3287 3394 2
3288 3395 2
3289 3396 2 ! Call variant select routine to select the variant components
3290 3397 2 ! given tag value and the pointer to the variant.
3291 3398 2 !
3292 3399 2 VARIANT = DBG$STA_VARIANT_SELECT(.TAG_VAL, .COMPTR);
3293 3400 2
3294 3401 2
3295 3402 2 ! We have no use for the tag variable, take it off from the
3296 3403 2 ! subnode.
3297 3404 2 !
3298 3405 2 REMQUE(.PRIMPTR[DBG$L_PRIM_BLINK], DUMMY);
3299 3406 2 END;
```


3300 3407 4
3301 3408 3
3302 3409 3
3303 3410 3
3304 3411 3
3305 3412 3
3306 3413 3
3307 3414 3
3308 3415 4
3309 3416 4
3310 3417 4
3311 3418 4
3312 3419 4
3313 3420 4
3314 3421 4
3315 3422 4
3316 3423 4
3317 3424 4
3318 3425 4
3319 3426 4
3320 3427 4
3321 3428 4
3322 3429 4
3323 3430 4
3324 3431 4
3325 3432 4
3326 3433 4
3327 3434 4
3328 3435 4
3329 3436 4
3330 3437 4
3331 3438 4
3332 3439 4
3333 3440 5
3334 3441 5
3335 3442 5
3336 3443 4
3337 3444 4
3338 3445 4
3339 3446 4
3340 3447 4
3341 3448 4
3342 3449 4
3343 3450 4
3344 3451 4
3345 3452 4
3346 3453 4
3347 3454 4
3348 3455 4
3349 3456 4
3350 3457 5
3351 3458 5
3352 3459 5
3353 3460 5
3354 3461 4
3355 3462 4
3356 3463 3

```
END;

! We have got variant components.
IF .VARIANT NEQ 0
THEN
  BEGIN

    ! Build subnode for this variant, fill in information.
    ! Note: Remember to take this node away, if this is not what we
    ! want.
    DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, RST$K_VARIANT, 0,
      RST$K_TYPE_VARIANT, 0, 0);
    SUBNODE = .PRIMPTR[DBG$L_PRIM_BLINK];
    SUBNODE[DBG$W_PNVAR_TAGID] = .TAGID;
    SUBNODE[DBG$W_PNVAR_INDEX] = 1;
    SUBNODE[DBG$V_PNVAR_VALID] = TRUE;
    SUBNODE[DBG$W_PNVAR_NCOMPS] = .VARIANT[RST$L_VAR_COMPCNT];
    SUBNODE[DBG$W_PNVAR_COMPLST] = .VARIANT[RST$L_VAR_COMPLST];
    SUBNODE[DBG$W_PNVAR_DSTPTR] = .VARIANT[RST$L_VAR_DSTPTR];

    ! Calculate the offsets of the variant components. Call
    ! GET_RECORD_COMPONENT to get the better variant component.
    COMVECPTR = .VARIANT[RST$L_VAR_COMPLST];
    TMP_SYMID = .SYMID[0];
    TMP_OFFSET = .OFFSET[0];
    INCR I FROM 0 TO .VARIANT[RST$L_VAR_COMPCNT] - 1 DO
      BEGIN
        PRIMPTR = GET_RECORD_COMPONENT(.PRIMPTR, .COMVECPTR[I],
          .BIT_OFFSET, TMP_SYMID, TMP_OFFSET, .I);
      END;

    ! If we did not find one, take the subnode off.
    IF .TMP_SYMID EQL .SYMID[0] AND
      .TMP_OFFSET EQL .OFFSET[0]
    THEN
      REMQUE(.PRIMPTR[DBG$L_PRIM_BLINK], DUMMY)

    ! We got one variant component.
    ELSE
      BEGIN
        SYMID[0] = .TMP_SYMID;
        OFFSET[0] = .TMP_OFFSET;
        SUBNODE[DBG$W_PNVAR_INDEX] = .IDX + 1;
      END;

  END;
```



```
3357 3464 3
3358 3465
3359 3466
3360 3467
3361 3468
3362 3469
3363 3470
3364 3471
3365 3472
3366 3473
3367 3474
3368 3475
3369 3476
3370 3477
3371 3478
3372 3479
3373 3480
3374 3481
3375 3482
3376 3483
3377 3484
3378 3485
3379 3486
3380 3487
3381 3488
3382 3489
3383 3490
3384 3491
3385 3492
3386 3493
3387 3494
3388 3495
3389 3496
3390 3497
3391 3498
3392 3499
3393 3500
3394 3501 1

END

! This is the normal case, get the component offset value.
! Select the smallest positive offset.
ELSE
BEGIN
DBG$STA SYMVALUE(.COMPTR, VALPTR, VALKIND);
IF .VALKIND EQL DBG$K_VAL_ADDR
THEN
BEGIN
LOCAL
VAL_OFFSET;

VAL_OFFSET = .VALPTR[0] * 8 + .VALPTR[1];
IF .VAL_OFFSET LEQ .BIT_OFFSET[0]
THEN
BEGIN
IF (.BIT_OFFSET[0] - .VAL_OFFSET) LEQ (.BIT_OFFSET[0] - .OFFSET[0])
THEN
BEGIN
SYMID[0] = .COMPTR;
OFFSET[0] = .VAL_OFFSET;
SUBNODE = .PRIMPTR[DBG$L_PRIM_BLINK];
SUBNODE[DBG$W_PNREC_INDEX] = .IDX + 1;
END;
END;
END;

END;

END;

! End of ELSE clause for non-variant type of
! record component.

RETURN .PRIMPTR;
END;
```

```
007C 0000 GET_RECORD COMPONENT:
      5E      24 C2 00002      .WORD      Save R2,R3,R4,R5,R6      : 3293
      52      04 AC 7D 00005      SUBL2      #36, SP      : 3357
00000000G 00      52 D0 00009      MOVQ      PRIMPTR, R2
      53      01 DD 00010      MOVL       R2, DBG$GL_CURRENT_PRIMARY
00000000G 00      8F BB 00019      PUSHL      R3      : 3361
      02      01 FB 00012      CALLS      #1, DBG$STA_SETCONTEXT
00000000G 00      02 FB 0001D      PUSHR      #M<R3,SP>      : 3362
      0B      6E D1 00024      CALLS      #2, DBG$STA_SYMKIND
      03      13 00027      CMPL       KIND, #11      : 3367
      00E1     31 00029      BEQL       1$
      54      D4 0002C 1$:      BRW        7$
      55      10 A3 D0 0002E      CLRL       VARIANT      : 3370
      55      10 A3 D0 0002E      MOVL       16(R3), TAGID      : 3376
```


			10	BC	10	52	11	000FF	5\$:	BRB	9\$		
			14	BC	0C	AE	D0	00101	6\$:	MOVL	TMP_SYMID, @SYMID		3458
						40	11	0010B		MOVL	TMP_OFFSET, @OFFSET		3459
					14	AE	9F	0010D	7\$:	BRB	8\$		3460
					1C	AE	9F	00110		PUSHAB	VALKIND		3473
						53	DD	00113		PUSHAB	VALPTR		
						03	FB	00115		PUSHL	R3		
		00000000G		00						CALLS	#3, DBG\$STA_SYMVALUE		
				02	14	AE	D1	0011C		CMPL	VALKIND, #2		3474
						31	12	00120		BNEQ	9\$		
				50	18	AE	D0	00122		MOVL	VALPTR, R0		3480
				51	1C	BE	40	7E	00126	MOVAQ	@VALPTR+4[R0], VAL_OFFSET		
			0C	BC		51	D1	0012B		CMPL	VAL_OFFSET, @BIT_OFFSET		3481
						22	14	0012F		BGTR	9\$		
		54	0C	BC		51	C3	00131		SUBL3	VAL_OFFSET, @BIT_OFFSET, R4		3484
		50	0C	BC	14	BC	C3	00136		SUBL3	@OFFSET, @BIT_OFFSET, R0		
				50		54	D1	0013C		CMPL	R4, R0		
						12	14	0013F		BGTR	9\$		
			10	BC		53	D0	00141		MOVL	R3, @SYMID		3487
			14	BC		51	D0	00145		MOVL	VAL_OFFSET, @OFFSET		3488
				52	18	A2	D0	00149		MOVL	24(R2), SUBNODE		3489
18	A2		18	AC		01	A1	0014D	8\$:	ADDW3	#1, IDX, 24(SUBNODE)		3490
				50	04	AC	D0	00153	9\$:	MOVL	PRIMPTR, R0		3500
						04	00157			RET			3501

; Routine Size: 344 bytes, Routine Base: DBG\$CODE + 16F5

; 3395 3502 1


```
3397 3503 1 ROUTINE SYMBOLIZE_ARRAY_ELEMENT(PRIMPTR, BIT_OFFSET) =
3398 3504 1
3399 3505 1 FUNCTION
3400 3506 1     This routine figures out the array element from offset value to
3401 3507 1     the start of the array. And build primary subnode for the
3402 3508 1     array element.
3403 3509 1
3404 3510 1 INPUTS
3405 3511 1     PRIMPTR - Pointer to Primary descriptor.
3406 3512 1
3407 3513 1     BIT_OFFSET - Array element offset value in bits to the start of
3408 3514 1     the array.
3409 3515 1
3410 3516 1 OUTPUTS
3411 3517 1     Return value is the pointer to the primary descriptor for the
3412 3518 1     array element. BIT_OFFSET is the offset value in bits to
3413 3519 1     the primary.
3414 3520 1
3415 3521 1
3416 3522 2 BEGIN
3417 3523 2
3418 3524 2 MAP
3419 3525 2     BIT_OFFSET: REF VECTOR[.LONG], ! Array cell Bit offset to array
3420 3526 2     PRIMPTR: REF DBG$PRIMARY; ! Primary descriptor for the object
3421 3527 2
3422 3528 2 LOCAL
3423 3529 2     ARRAY_SUBNODE: REF DBG$PRIM_NODE, ! Pointer to Array subnode
3424 3530 2     ARRAY_SUBVEC: REF DBG$PRIM_NODE_SUBS, ! Pointer to array subscript info
3425 3531 2     BOUND_FLAG, ! Flag to indicate the bound is exceeded
3426 3532 2     OFFSET, ! Offset value
3427 3533 2     STRIDE; ! Stride along each dimension in bits
3428 3534 2
3429 3535 2
3430 3536 2 ARRAY_SUBNODE = .PRIMPTR[DBG$L_PNARR_CELLTYPE];
3431 3537 2 IF .ARRAY_SUBNODE[DBG$B_PNODE_FCODE] NEQ RST$K_TYPE_ARRAY
3432 3538 2 THEN
3433 3539 2     $DBG_ERROR('DBGSYMBLZ\SYMBOLIZE_ARRAY_ELEMENT');
3434 3540 2
3435 3541 2
3436 3542 2 ! We treat the start of the array is the first array element.
3437 3543 2
3438 3544 2 IF .BIT_OFFSET[0] EQL 0
3439 3545 2 THEN
3440 3546 2     BEGIN
3441 3547 2     ARRAY_SUBNODE[DBG$V_PNODE_EVAL] = TRUE;
3442 3548 2     RETURN BUILD_PRIMARY_SUBNODE(.PRIMPTR, .ARRAY_SUBNODE[DBG$L_PNARR_CELLTYPE],
3443 3549 2     .BIT_OFFSET);
3444 3550 2     END;
3445 3551 2
3446 3552 2
3447 3553 2 ! Divide the offset by largest stride and then adjusted by the lower bound,
3448 3554 2 ! this is the subscript we want.
3449 3555 2
3450 3556 2 OFFSET = .BIT_OFFSET[0];
3451 3557 2 ARRAY_SUBVEC = ARRAY_SUBNODE[DBG$A_PNARR_SVECTOR];
3452 3558 2 BOUND_FLAG = FALSE;
3453 3559 2 IF .ARRAY_SUBNODE[DBG$V_PNARR_COLUMN]
```



```

3454 3560 2 THEN
3455 3561 3 BEGIN
3456 3562 3   DECR I FROM .ARRAY_SUBNODE[DBG$B_PNARR_DIMCNT] - 1 TO 0 DO
3457 3563 4   BEGIN
3458 3564 4   IF .ARRAY_SUBNODE[DBG$V_PNARR_BITREF]
3459 3565 4   THEN
3460 3566 4     STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE]
3461 3567 4
3462 3568 4   ELSE
3463 3569 4     STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE] * 8;
3464 3570 4
3465 3571 4   ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] =
3466 3572 4     .OFFSET / .STRIDE + .ARRAY_SUBVEC[.I, DBG$L_PNSUB_LBOUND];
3467 3573 4
3468 3574 4   IF .ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] GTR
3469 3575 4     .ARRAY_SUBVEC[.I, DBG$L_PNSUB_UBOUND]
3470 3576 4   THEN
3471 3577 4     BOUND_FLAG = TRUE;
3472 3578 4
3473 3579 4   OFFSET = .OFFSET MOD .STRIDE;
3474 3580 3   END;
3475 3581 3
3476 3582 3   END
3477 3583 3
3478 3584 2 ELSE
3479 3585 3 BEGIN
3480 3586 3   INCR I FROM 0 TO .ARRAY_SUBNODE[DBG$B_PNARR_DIMCNT] - 1 DO
3481 3587 4   BEGIN
3482 3588 4   IF .ARRAY_SUBNODE[DBG$V_PNARR_BITREF]
3483 3589 4   THEN
3484 3590 4     STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE]
3485 3591 4
3486 3592 4   ELSE
3487 3593 4     STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE] * 8;
3488 3594 4
3489 3595 4   ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] =
3490 3596 4     .OFFSET / .STRIDE + .ARRAY_SUBVEC[.I, DBG$L_PNSUB_LBOUND];
3491 3597 4
3492 3598 4   IF .ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] GTR
3493 3599 4     .ARRAY_SUBVEC[.I, DBG$L_PNSUB_UBOUND]
3494 3600 4   THEN
3495 3601 4     BOUND_FLAG = TRUE;
3496 3602 4
3497 3603 4   OFFSET = .OFFSET MOD .STRIDE;
3498 3604 3   END;
3499 3605 3
3500 3606 2   END;
3501 3607 2
3502 3608 2
3503 3609 2 ! Make sure the subscripts calculated from above are not exceed the bounds.
3504 3610 2 !
3505 3611 2 OFFSET = 0;
3506 3612 2 INCR I FROM 0 TO .ARRAY_SUBNODE[DBG$B_PNARR_DIMCNT] - 1 DO
3507 3613 3   BEGIN
3508 3614 3   IF .BOUND_FLAG
3509 3615 3   THEN
3510 3616 3     ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] =
```



```

3511      .ARRAY_SUBVEC[.I, DBG$L_PNSUB_UBOUND];
3512
3513      IF .ARRAY_SUBNODE[DBG$V_PNARR_BITREF]
3514      THEN
3515          STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE]
3516
3517      ELSE
3518          STRIDE = .ARRAY_SUBVEC[.I, DBG$L_PNSUB_STRIDE] * 8;
3519
3520      OFFSET = .OFFSET + .ARRAY_SUBVEC[.I, DBG$L_PNSUB_SVALUE] * .STRIDE;
3521      END;
3522
3523      ! Adjust the offset. Based on A(0,0,...).
3524
3525      IF .ARRAY_SUBNODE[DBG$V_PNARR_BITREF]
3526      THEN
3527          BIT_OFFSET[0] = .BIT_OFFSET[0] + (-1 * .ARRAY_SUBNODE[DBG$L_PNARR_OFFSET])
3528
3529      ELSE
3530          BIT_OFFSET[0] = .BIT_OFFSET[0] + (-8 * .ARRAY_SUBNODE[DBG$L_PNARR_OFFSET]);
3531
3532      BIT_OFFSET[0] = .BIT_OFFSET[0] - .OFFSET;
3533      ARRAY_SUBNODE[DBG$V_PNODE_EVAL] = TRUE;
3534      ARRAY_SUBNODE[DBG$B_PNARR_SUBCNT] = .ARRAY_SUBNODE[DBG$B_PNARR_DIMCNT];
3535      RETURN BUILD_PRIMARY_SUBNODE(.PRIMPTR, .ARRAY_SUBNODE[DBG$L_PNARR_CELLTYPE],
3536      .BIT_OFFSET);
3537
3538      END;
3539
3540
```

```

                                .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
42 4D 59 53 5C 5A 4C 42 4D 59 53 47 42 44 21 0040E P.ADX: .ASCII \!DBGSYMBLZ\<92>\SYMBOLIZE_ARRAY_ELEMENT\
45 4C 45 5F 59 41 52 52 41 5F 45 5A 49 4C 4F 0041D
                                54 4E 45 4D 0042C
```

```

                                .PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                OFFC 00000 SYMBOLIZE_ARRAY_ELEMENT:
                                .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
                                5A 04 AC D0 00002 MOVL PRIMPTR, R10
                                52 18 AA D0 00006 MOVL 24(R10), ARRAY_SUBNODE
                                57 08 A2 9E 0000A MOVAB 8(ARRAY_SUBNODE), R7
                                01 01 A7 91 0000E CMPB 1(R7), #1
                                BEQL 1$
                                00000000 EF 9F 00014 PUSHAB P.ADX
                                01 DD 0001A PUSHL #1
                                00028362 8F DD 0001C PUSHL #164706
                                00000000G 00 03 FB 00022 CALLS #3, LIB$SIGNAL
                                55 08 AC D0 00029 1$: MOVL BIT_OFFSET, R5
                                65 D5 0002D TSTL (R5)
                                07 12 0002F BNEQ 2$
```

```

: 3503
: 3536
: 3537
: 3539
: 3544
:
```


02	A7	01	88	00031	BISB2	#1, 2(R7)	3547		
		010E	31	00035	BRW	22\$	3549		
	53	65	D0	00038	2\$:	MOVL (R5), OFFSE	3556		
	50	A2	9E	0003B	MOVAB	40(R2), ARRAY_SUBVEC	3557		
		5B	D4	0003F	CLRL	BOUND_FLAG	3558		
52	67	11	E1	00041	BBC	#17, (R7), 8\$	3559		
	58	1B	A2	9A	00045	MOVZBL	27(ARRAY_SUBNODE), R8	3562	
		58	D7	00049	DECL	R8			
	51	01	A8	9E	0004B	MOVAB	1(R8), I		
		41	11	0004F	BRB	7\$			
54	51	14	C5	00051	3\$:	MULL3	#20, I, R4	3566	
09	67	12	E1	00055	BBC	#18, (R7), 4\$	3564		
		04	A440	9F	00059	PUSHAB	4(R4)[ARRAY_SUBVEC]	3566	
	56	9E	D0	0005D	MOVL	@(SP)+, STRIDE			
		08	11	00060	BRB	5\$			
		04	A440	9F	00062	4\$:	PUSHAB	4(R4)[ARRAY_SUBVEC]	3569
56	9E	03	78	00066	ASHL	#3, @(SP)+, STRIDE			
59	53	56	C7	0006A	5\$:	DIVL3	STRIDE, OFFSET, R9	3572	
		6440	9F	0006E	PUSHAB	(R4)[ARRAY_SUBVEC]			
		08	A440	9F	00071	PUSHAB	8(R4)[ARRAY_SUBVEC]		
	9E	59	9E	C1	00075	ADDL3	@(SP)+, R9, -(SP)+		
		0C	A440	9F	00079	PUSHAB	12(R4)[ARRAY_SUBVEC]	3575	
		6440	9F	0007D	PUSHAB	(R4)[ARRAY_SUBVEC]			
	9E	9E	D1	00080	CMPL	@(SP)+, @(SP)+			
		03	15	00083	BLEQ	6\$			
	5B	01	D0	00085	MOVL	#1, BOUND_FLAG	3577		
7E	00	53	01	7A	00088	6\$:	EMUL	#1, OFFSET, #0, -(SP)	3579
53	53	8E	56	7B	0008D	EDIV	STRIDE, (SP)+, OFFSET, OFFSET		
		BC	51	F4	00092	7\$:	SOBGEQ	I, 3\$	3562
			50	11	00095	BRB	14\$	3559	
	58	1B	A2	9A	00097	8\$:	MOVZBL	27(ARRAY_SUBNODE), R8	3586
			58	D7	0009B	DECL	R8		
	51	01	CE	0009D	MNEGL	#1, I			
		41	11	000A0	BRB	13\$			
54	51	14	C5	000A2	9\$:	MULL3	#20, I, R4	3590	
09	67	12	E1	000A6	BBC	#18, (R7), 10\$	3588		
		04	A440	9F	000AA	PUSHAB	4(R4)[ARRAY_SUBVEC]	3590	
	56	9E	D0	000AE	MOVL	@(SP)+, STRIDE			
		08	11	000B1	BRB	11\$			
		04	A440	9F	000B3	10\$:	PUSHAB	4(R4)[ARRAY_SUBVEC]	3593
56	9E	03	78	000B7	ASHL	#3, @(SP)+, STRIDE			
59	53	56	C7	000BB	11\$:	DIVL3	STRIDE, OFFSET, R9	3596	
		6440	9F	000BF	PUSHAB	(R4)[ARRAY_SUBVEC]			
		08	A440	9F	000C2	PUSHAB	8(R4)[ARRAY_SUBVEC]		
	9E	59	9E	C1	000C6	ADDL3	@(SP)+, R9, -(SP)+		
		0C	A440	9F	000CA	PUSHAB	12(R4)[ARRAY_SUBVEC]	3599	
		6440	9F	000CE	PUSHAB	(R4)[ARRAY_SUBVEC]			
	9E	9E	D1	000D1	CMPL	@(SP)+, @(SP)+			
		03	15	000D4	BLEQ	12\$			
	5B	01	D0	000D6	MOVL	#1, BOUND_FLAG	3601		
7E	00	53	01	7A	000D9	12\$:	EMUL	#1, OFFSET, #0, -(SP)	3603
53	53	8E	56	7B	000DE	EDIV	STRIDE, (SP)+, OFFSET, OFFSET		
		51	58	F3	000E3	13\$:	AOBLEQ	R8, I, 9\$	3586
			53	D4	000E7	14\$:	CLRL	OFFSEf	3611
	59	01	CE	000E9	MNEGL	#1, I		3614	
		34	11	000EC	BRB	19\$			
	0E	5B	E9	000EE	15\$:	BLBC	BOUND_FLAG, 16\$		

54	59	14	C5	000F1	MULL3	#20, 1, R4	3616
		6440	9F	000F5	PUSHAB	(R4)[ARRAY SUBVEC]	3617
		0C A440	9F	000F8	PUSHAB	12(R4)[ARRAY SUBVEC]	
	9E	9E	D0	000FC	MOVL	@(SP)+, @(SP)+	
54	59	14	C5	000FF	MULL3	#20, 1, R4	3621
09	67	12	E1	00103	BBC	#18, (R7), 17\$	3619
		04 A440	9F	00107	PUSHAB	4(R4)[ARRAY SUBVEC]	3621
	56	9E	D0	0010B	MOVL	@(SP)+, STRIDE	
		08	11	0010E	BRB	18\$	
		04 A440	9F	C0110	PUSHAB	4(R4)[ARRAY SUBVEC]	3624
56	9E	03	78	00114	ASHL	#3, @(SP)+, STRIDE	
		6440	9F	00118	PUSHAB	(R4)[ARRAY SUBVEC]	3626
54	9E	56	C5	0011B	MULL3	STRIDE, @(SP)+, R4	
	53	54	C0	0011F	ADDL2	R4, OFFSET	
C8	59	58	F3	00122	AOBLEQ	R8, 1, 15\$	3612
06	67	12	E1	00126	BBC	#18, (R7), 20\$	3632
	65	20	A2	C2	SUBL2	32(ARRAY_SUBNODE), (R5)	3634
		0A	11	0012E	BRB	21\$	
	50	20	A2	D0	MOVL	32(ARRAY_SUBNODE), R0	3637
	50	08	C4	00134	MULL2	#8, R0	
	65	50	C2	00137	SUBL2	R0, (R5)	
	65	53	C2	0013A	SUBL2	OFFSET, (R5)	3640
02	A7	01	88	0013D	BISB2	#1, 2(R7)	3641
1F	A2	1B	A2	90	MOVB	27(ARRAY_SUBNODE), 31(ARRAY_SUBNODE)	3642
		55	DD	00146	PUSHL	R5	3644
		24	A2	DD	PUSHL	36(ARRAY_SUBNODE)	3643
		5A	DD	0014B	PUSHL	R10	
FC90	CF	03	FB	0014D	CALLS	#3, BUILD_PRIMARY_SUBNODE	
		04	00152	RET			3646

; Routine Size: 339 bytes, Routine Base: DBG\$CODE + 184D

; 3541 3647 1


```
3543 1 ROUTINE SYMBOLIZE_RECORD_COMPONENT(PRIMPTR, BIT_OFFSET) =
3544 1
3545 1 FUNCTION
3546 1     This routine gets a list of record components, call GET_RECORD_COMPONENT
3547 1     to figure out THE component that is closest to the desired address.
3548 1
3549 1 INPUTS
3550 1     PRIMPTR - Pointer to primary descriptor.
3551 1
3552 1     BIT_OFFSET - Offset value to the Primary.
3553 1
3554 1 OUTPUTS
3555 1     Returned value is the Pointer to primary and updated offset value
3556 1     to the Primary.
3557 1
3558 1
3559 1 BEGIN
3560 1
3561 1 MAP
3562 1     BIT_OFFSET: REF VECTOR[.LONG], ! Offset to Primary
3563 1     PRIMPTR: REF DBG$PRIMARY; ! Pointer to Primary
3564 1
3565 1 LOCAL
3566 1     BITSIZE, ! Bit size of the record (not used)
3567 1     COMPVECPTR: REF VECTOR[.LONG], ! Pointer to vector of component
3568 1     ! SYMIDs for this record type
3569 1     NCOMPS, ! Number of components for this record
3570 1     OFFSET, ! Left over offsets to Primary
3571 1     REC_SUBNODE: REF DBG$PRIM_NODE, ! Pointer to Primary Subnode
3572 1     SYMID: REF RST$ENTRY; ! Symid pointer
3573 1
3574 1 ! Get the last subnode we just built. (This one must be a RST$K_TYPE_RECORD.
3575 1 ! type).
3576 1
3577 1 REC_SUBNODE = .PRIMPTR[DBG$L PRIM BLINK];
3578 1 IF .REC_SUBNODE[DBG$B_PNODE_FCODE] NEQ RST$K_TYPE_RECORD
3579 1 THEN
3580 1     $DBG_ERROR('DBGSYMBLZ\SYMBOLIZE_RECORD_COMPONENT');
3581 1
3582 1 ! Get a list of record components for this record..
3583 1
3584 1 !
3585 1 DBG$STA_TYP_RECORD(.REC_SUBNODE[DBG$L_PNODE_TYPEID], NCOMPS, COMPVECPTR,
3586 1 ! BITSIZE);
3587 1
3588 1 ! Find a SYMID from this list of record components has the closest
3589 1 ! offset value to the given offset value.
3590 1
3591 1 SYMID = 0;
3592 1 OFFSET = 0;
3593 1 INCR I FROM 0 TO .NCOMPS - 1 DO
3594 1     BEGIN
3595 1         PRIMPTR = GET_RECORD_COMPONENT(.PRIMPTR, .COMPVECPTR[I],
3596 1         ! .BIT_OFFSET, SYMID, OFFSET, .I);
3597 1
3598 1     END;
3599 1
```



```

: 3600      3705 2
: 3601      3706 2
: 3602      3707 2
: 3603      3708 2
: 3604      3709 2
: 3605      3710 2
: 3606      3711 2
: 3607      3712 2
: 3608      3713 2
: 3609      3714 2
: 3610      3715 2
: 3611      3716 2
: 3612      3717 2
: 3613      3718 2
: 3614      3719 2
: 3615      3720 2
: 3616      3721 2
: 3617      3722 2
: 3618      3723 1

! We did not find a record component just returned the primary we got
! so far.
IF .SYMID EQL 0 THEN RETURN .PRIMPTR;

! We have found a better symbol, re-figure the offset value to the
! record component.
BIT_OFFSET[0] = .BIT_OFFSET[0] - .OFFSET;

! Building the Primary subnode for record component.
REC_SUBNODE[DBG$V_PNODE_EVAL] = TRUE;
RETURN BUILD_PRIMARY_SUBNODE(.PRIMPTR, .SYMID, .BIT_OFFSET);
END;
```

```

                                .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
42 4D 59 53 5C 5A 4C 42 4D 59 53 47 42 44 24 00430 P.ADY: .ASCII \DBGSYMBLZ\<92>\SYMBOLIZE_RECORD_COMPONENT\
4F 43 5F 44 52 4F 43 45 52 5F 45 5A 49 4C 4F 0043F
                                4E 4F 50 4D 0044E
                                54 4E 45 00452 .ASCII \ENT\
```

```

                                .PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                000C 00000 SYMBOLIZE_RECORD_COMPONENT:
                                .WORD Save R2,R3
                                SUBL2 #20, SP
                                MOVL PRIMPTR, R0
                                MOVL 24(R0), REC_SUBNODE
                                CMPB 9(REC_SUBNODE), #7
                                BEQL 1$
                                PUSHAB P.ADY
                                PUSHL #1
                                PUSHL #164706
                                CALLS #3, LIB$SIGNAL
                                PUSHL SP
                                PUSHAB COMPVECPTR
                                PUSHAB NCOMPS
                                PUSHL 12(REC_SUBNODE)
                                CALLS #4, DBG$STA_TYP_RECORD
                                CLRQ OFFSET
                                MNEGL #1, I
                                BRB 3$
                                PUSHL I
                                PUSHAB OFFSET
                                PUSHAB SYMID
                                PUSHL BIT_OFFSET
                                PUSHL @COMPVECPTR[1]

                                5E      14 C2 00002
                                50      04 AC DD 00005
                                53      18 A0 DD 00009
                                07      09 A3 91 0000D
                                00000000' 15 13 00011
                                EF 9F 00013
                                01 DD 00019
                                00028362 8F DD 0001B
                                03 FB 00021
                                00000000G 00 5E DD 00028 1$:
                                08 AE 9F 0002A
                                10 AE 9F 0002D
                                0C A3 DD 00030
                                00000000G 00 04 FB 00033
                                0C AE 7C 0003A
                                52 01 CE 0003D
                                1B 11 00040
                                52 DD 00042 2$:
                                10 AE 9F 00044
                                18 AE 9F 00047
                                08 AC DD 0004A
                                14 BE42 DD 0004D
```


			04	AC	DD	00051		PUSHL	PRIMPTR		
				06	FB	00054		CALLS	#6, GET_RECORD_COMPONENT		
	FCFC	CF		50	D0	00059		MOVL	R0, PRIMPTR		
E0	04	AC		08	AE	F2 0005D	3\$:	AOBLSS	NCOMPS, 1, 2\$		3700
		52		10	AE	D5 00062		TSTL	SYMID		3710
					05	12 00065		BNEQ	4\$		
		50		04	AC	D0 00067		MOVL	PRIMPTR, R0		
						04 0006B		RET			
	08	BC		0C	AE	C2 0006C	4\$:	SUBL2	OFFSET, @BIT_OFFSET		3716
	0A	A3			01	88 00071		BISB2	#1, 10(REC_SUBNODE)		3721
				08	AC	DD 00075		PUSHL	BIT_OFFSET		3722
				14	AE	DD 00078		PUSHL	SYMID		
				04	AC	DD 0007B		PUSHL	PRIMPTR		
	FC0C	CF			03	FB 0007E		CALLS	#3, BUILD_PRIMARY_SUBNODE		
					04	00083		RET			3723

; Routine Size: 132 bytes, Routine Base: DBG\$CODE + 19A0

; 3619 3724 1
; 3620 3725 0 END ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$OWN	8	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$PLIT	1109	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$CODE	6692	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	11	0	1000	00:01.8
_\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	0	0	7	00:00.2
_\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	222	14	97	00:02.1
_\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1					
	418	143	34	31	00:00.3
_\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	9	2	22	00:00.3

COMMAND QUALIFIERS

DBGSYMBLZ
V04-000

N 11
16-Sep-1984 02:41:46
14-Sep-1984 12:17:51

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

Page 121
(29)

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

; Size: 6692 code + 1117 data bytes
; Run Time: 01:45.9
; Elapsed Time: 01:57.0
; Lines/CPU Min: 2110
; Lexemes/CPU-Min: 13086
; Memory Used: 479 pages
; Compilation Complete

0095

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY