


```

SSSSSSSS YY YY SSSSSSS FFFFFFFF AAAAAA 000000
SSSSSSSS YY YY SSSSSSS FFFFFFFF AAAAAA 000000
SS SS YY YY SS SS FF AA AA 00 00
SS SS YY YY SS SS FF AA AA 00 00
SS SS YY YY SS SS FF AA AA 00 00
SSSSSS YY YY SSSSSS FFFFFFFF AA AA 00 00
SSSSSS YY YY SSSSSS FFFFFFFF AA AA 00 00
SS YY YY SS AA AA 00 00
SS YY YY SS AA AA 00 00
SS YY YY SS AA AA 00 00
SSSSSS YY YY SSSSSSS FF AA AA 000000
SSSSSS YY YY SSSSSSS FF AA AA 000000

```

```

LL LL I I I I I SSSSSSS
LL LL I I I I I SSSSSSS
LL LL I I SS
LL LL I I SS
LL LL I I SS
LL LL I I SSSSSS
LL LL I I SSSSSS
LL LL I I SS
LL LL I I SS
LL LL I I SS
LLLLLLLLLL I I I I I SSSSSSS
LLLLLLLLLLLL I I I I I SSSSSSS

```

SYSFAO
Table of contents

(2)	98
(3)	206
(4)	411
(5)	459
(6)	530
(7)	678
(8)	890
(9)	1028
(10)	1345

DECLARATIONS
FAO - MAIN PROGRAM
GETCHAR - Routine to get next char from input string
GETCOUNT - Routine to get repeat-count or field-width
CVTASC - Insert ASCII string
CVTNUM - Convert numeric parameter to ASCII
QUICKSERVE - Small service routines
PERCENT - Time directives, plural 'S', and UIC
HANDLER - Condition handler

```
0000 1 .TITLE SYSFAO - FORMATTED ASCII OUTPUT SYSTEM SERVICE
0000 2 .IDENT 'V04-000'
0000 3
0000 4 :*****
0000 5 :*
0000 6 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8 :* ALL RIGHTS RESERVED.
0000 9 :*
0000 10 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15 :* TRANSFERRED.
0000 16 :*
0000 17 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19 :* CORPORATION.
0000 20 :*
0000 21 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23 :*
0000 24 :*
0000 25 :*****
0000 26
0000 27 :++
0000 28 : FACILITY: SYSTEM SERVICE
0000 29
0000 30 : ABSTRACT:
0000 31
0000 32 : This module provides general formatting services. It converts
0000 33 : binary values to octal, hexadecimal, and decimal ASCII
0000 34 : representations, and also inserts ASCII strings and converts
0000 35 : date and time to ASCII.
0000 36
0000 37 : ENVIRONMENT:
0000 38
0000 39 : FAO runs in the mode of the caller.
0000 40
0000 41 : AUTHOR: Henry M. Levy , CREATION DATE: 29-JAN-1977
0000 42
0000 43 : MODIFIED BY:
0000 44
0000 45 : V03-014 LJK0278 Lawrence J. Kenah 2-May-1984
0000 46 : Move this code to separate program section to reduce the
0000 47 : strain on the cursed word displacements.
0000 48
0000 49 : V03-013 LMP0201 L. Mark Pilant, 28-Feb-1984 13:22
0000 50 : Add support for formatting the match-all identifier.
0000 51
0000 52 : V03-012 LMP0169 L. Mark Pilant, 11-Nov-1983 15:07
0000 53 : Correctly handle member wildcards in the %I directive.
0000 54
0000 55 : V03-011 LMP0119 L. Mark Pilant, 16-Jun-1983 12:05
0000 56 : Make non-translating identifiers appear as hex numbers.
0000 57 :
```

```
0000 58 : V03-010 JLV0257 Jake VanNoy 23-MAY-1983
0000 59 : Change !AF to not make "." out of valid 8 bit characters.
0000 60 :
0000 61 : V03-009 LMP0111 L. Mark Pilant, 9-May-1983 9:45
0000 62 : Add a new directive, %I, to allow formatting of identifiers.
0000 63 :
0000 64 : V03-008 LMP0078 L. Mark Pilant, 10-Feb-1983 12:52
0000 65 : Modify the method used when checking for wildcard group
0000 66 : and member portions of the UIC.
0000 67 :
0000 68 : V03-007 LMP0056 L. Mark Pilant, 28-Oct-1982 20:50
0000 69 : Correct a problem introduced by LMP0052 which caused a
0000 70 : truncated search of the % directive table.
0000 71 :
0000 72 : V03-006 LMP0052 L. Mark Pilant, 14-Oct-1982 12:30
0000 73 : Add a new directive, !%U, to allow formatting of a UIC.
0000 74 :
0000 75 : V03-005 MSH0001 Maryann S. Hinden 20-NOV-1981
0000 76 : Use longword displacement to reference EXESSIGTORET.
0000 77 :
0000 78 : V03-004 DWT0001 David W. Thiel 06-Nov-1981
0000 79 : Fixed condition handler. Check argument to $ASCTIM to
0000 80 : prevent exception in $ASCTIM.
0000 81 :
0000 82 : V03-003 PCA0001 Paul C. Anagnostopoulos 22-Jul-1981
0000 83 : Fixed a bug wherein !AF did not replace unprintable
0000 84 : characters if it encountered result string overflow.
0000 85 : Now it replaces those characters that it does copy.
0000 86 :
0000 87 : V03-002 TCM0001 Trudy C. Matthews 10-Mar-1981
0000 88 : Change CALLS with word displacement to CALLS with longword
0000 89 : displacement.
0000 90 :
0000 91 : V03-001 TMH0001 Tim Halvorsen 24-Feb-1981
0000 92 : Add condition handler to catch access violations
0000 93 : and the like, so that services like $PUTMSG do
0000 94 : not cause an access violation in programs like DCL
0000 95 : simply because not enough arguments were supplied.
0000 96 :--
```

0000 98 .SBTTL DECLARATIONS

0000 99

0000 100

0000 101

0000 102

0000 103

0000 104

0000 105

0000 106

0000 107

0000 108

0000 109

0000 110

0000 111

0000 112

00000000

00000004

00000008

0000000C

00000010

0000

FFFFFFF0

FFFFFFF4

FFFFFFF8

FFFFFFFC

0000000D

0000000A

00000021

00000009

0000000C

0000 113

0000 114

0000 115

0000 116

0000 117

0000 118

0000 119

0000 120

0000 121

0000 122

0000 123

0000 124

0000 125

0000 126

0000 127

0000 128

0000 129

0000 130

0000 131

0000 132

0000 133

00000000 134

0000 135

0000 136

0000 137

MACROS:

\$\$\$DEF : define system status codes
\$CHFDEF : Condition handling facility
\$\$FDEF : Call frame definitions
\$UICDEF : UIC FIELD OFFSETS

EQUATED SYMBOLS:

ARGCOUNT = 0 : offset to argument count
INDSC = 4 : offset to input string descriptor
OUTLEN = 8 : offset to output length
OUTDSC = 12 : offset to output buffer descriptor
FIRSTARG = 16 : offset to first conversion param

INLEN = -16 : local offset to input length remaining
INPTR = -12 : local offset to input string pointer
LASTVAL = -8 : local offset to last value converted
FIELDEND = -4 : local offset to end of defined field

CR = 13 : carriage return
LF = 10 : line feed
EXCL = 33 : exclamation ('!')
TAB = 9 : horizontal tab
FF = 12 : form feed

OWN STORAGE:

.PSECT YF\$SYSFAO

ASC_NAMES:

.ASCII /0123456789ABCDEF/ ; ASCII digits

The following table contains the first character for all
FAO conversion directives. The first part of the table
contains the first character for two-character directives,
while the second half of the table contains the one-character
directives.

NOTE -- The ordering of this table must be preserved. The index
of the directives found in this table is used to dispatch
via a CASE statement in the main program (FAO).
Routine CVTNUM also uses the index to dispatch and to
compute the proper radix for the conversion.

CNTRL_TABLE:

42 41 39 38 37 36 35 34 33 32 31 30
46 45 44 43

```

0010 154 TWO_CHAR_CNTRLS:
4F 0010 155     .ASCII  /O/           ; octal conversions
58 0011 156     .ASCII  /X/           ; hex conversions
55 0012 157     .ASCII  /U/           ; unsigned decimal
53 0013 158     .ASCII  /S/           ; signed decimal
5A 0014 159     .ASCII  /Z/           ; unsigned decimal zero filled
41 0015 160     .ASCII  /A/           ; ascii insertion directives
25 0016 161     .ASCII  /%/           ; time conversion, plural indication, or UI
2A 0017 162     .ASCII  /*/           ; character repeater
0018 163 ONE_CHAR_CNTRLS:
2B 0018 164     .ASCII  /+/           ; skip argument
2D 0019 165     .ASCII  /-/           ; backup argument
3C 001A 166     .ASCII  /</          ; begin field definition
3E 001B 167     .ASCII  />/          ; end of field definition
001C 168 REPLACE_CHRS:
2F 001C 169     .ASCII  ./           ; these are one or two char replacements
5F 001D 170     .ASCII  /           ; newline
5E 001E 171     .ASCII  /#           ; tab
21 001F 172     .ASCII  /!           ; form feed
0G000010 0020 173 CNTRL_LENGTH = .-CNTRL_TABLE ; insert exclamation
00000008 0020 174     .-CNTRL_TABLE ; length of table
0000000C 0020 175 ONECHAR_INDEX = CNTRL_LENGTH - <ONE_CHAR_CNTRLS - CNTRL_TABLE>
0020 176
0020 177 REPL_OFFSET = REPLACE_CHRS - CNTRL_TABLE ; offset of replacement chars
0020 178
0020 179 STRING_TYPES:
46 44 53 43 0020 180     .ASCII  /CSDF/           ; ascii string types
0024 181 DATA_TYPES:
4C 57 42 0024 182     .ASCII  /BWL/           ; byte, word, or long
0027 183 PERCENT_STR:
54 44 53 49 55 0027 184     .ASCII  /UISDT/           ; subtypes for % directive
002C 185 FIELDS:
20 10 08 002C 186     .BYTE   8,16,32           ; field size for B,W,and L
002F 187 REPLACEMENT:
21 0C 09 0A 002F 188     .BYTE   LF,TAB,FF,EXCL ; simple replacement table
0033 189
0033 190 ;
0033 191 ; The following array contains the number of Octal and Hex digits in
0033 192 ; byte, word, and longword fields. The byte digits are first, the
0033 193 ; hex digits starting at the 4'th entry so that the array may be
0033 194 ; context indexed.
0033 195 ;
0033 196
0033 197 OCT_HEX_DIGITS:
00 08 06 03 0033 198     .BYTE   3,6,11,0
08 04 02 0037 199     .BYTE   2,4,8
003A 200
003A 201 RADIX:
0A 0A 0A 10 08 003A 202     .BYTE   8,16,10,10,10 ; radix for numeric conversions
003F 203
003F 204

```

```

003F 206 .SBTTL FAO - MAIN PROGRAM
003F 207 :++
003F 208 : FUNCTIONAL DESCRIPTION:
003F 209 :
003F 210 : This routine is the entry point for the FAO and FAOL system
003F 211 : services. The caller's control string is scanned for control
003F 212 : characters ('!'). All other information is simply passed to
003F 213 : the output buffer. If a control directive is found, it is parsed
003F 214 : and an action routine is dispatched.
003F 215 :
003F 216 : CALLING SEQUENCE:
003F 217 :
003F 218 : CALLS or CALLG to SYSSFAO or SYSSFAOL
003F 219 :
003F 220 : INPUT PARAMETERS:
003F 221 :
003F 222 : INDSC - The address of a string descriptor for the input
003F 223 : control string.
003F 224 : OUTLEN - The address of a word to receive the length of
003F 225 : the output string
003F 226 : OUTDSC - The address of a string descriptor for the output
003F 227 : buffer.
003F 228 : FIRSTARG - For FAOL, this is the address of a list of longword
003F 229 : parameters. For FAO, this is the first of a
003F 230 : variable number of parameters which
003F 231 : may have been passed on the call argument list.
003F 232 :
003F 233 : IMPLICIT INPUTS:
003F 234 :
003F 235 : none
003F 236 :
003F 237 : OUTPUT PARAMETERS:
003F 238 :
003F 239 : OUTLEN - Word pointed to will receive length of output buffer.
003F 240 :
003F 241 : IMPLICIT OUTPUTS:
003F 242 :
003F 243 : none
003F 244 :
003F 245 : COMPLETION CODES:
003F 246 :
003F 247 : SSS_NORMAL - success code, normal return
003F 248 : SSS_BUFFEROVF - output buffer overflow, attempt to write past end of outpu
003F 249 : SSS_BADPARAM - invalid directive specified
003F 250 : SSS_ACCVIO - unable to read argument list or address arguments
003F 251 :
003F 252 : SIDE EFFECTS:
003F 253 :
003F 254 : none
003F 255 :
003F 256 : --
003F 257 :
003F 258 :
003F 259 : Global register usage:
003F 260 :
003F 261 : R7,R8 - scratch registers
003F 262 : R9 - number of characters remaining in output buffer

```



```

003F 263 : R10 - current position in output buffer
003F 264 : R11 - pointer to next conversion parameter
003F 265 :
003F 266 : Locals
003F 267 :
003F 268 : INLEN(FP) - (word) length of input control string
003F 269 : INPTR(FP) - address of position in input control string
003F 270 :
003F 271 :
003F 272 :
003F 273 : Entry point for call with multiple arguments on stack
003F 274 :
003F 275 :
003F 276 EXESFAO::
003F 277
003F 278 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; save all registers
6D 05AD'CF OFFC 0041 279 MOVAB W^HANDLER,(FP) ; Establish condition handler
5B 10 AC DE 0046 280 MOVAL FIRSTARG(AP),R11 ; get address of first argument
004A 281 BRB FAO ; go to main routine
004C 282
004C 283 :
004C 284 : Entry point for FAOL call.
004C 285 :
004C 286
004C 287 EXESFAOL::
004C 288
004C 289 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
6D 05AD'CF OFFC 004E 290 . B W^HANDLER,(FP) ; Establish condition handler
5B 10 AC DO 0053 291 MO - FIRSTARG(AP),R11 ; address of first argument
0057 292 FAO:
0057 293 CLRQ -(SP) ; save space for LASTVAL and FIELDEND
7E 04 BC 7C 0059 294 MOVQ @INDSC(AP),-(SP) ; save locals on stack
59 0C BC 7D 005D 295 MOVQ @OUTDSC(AP),R9 ; load output descriptor into R9,R10
59 59 3C 0061 296 MOVZWL R9,R9 ; ensure word length
0064 297
0064 298 :
0064 299 : Look for a control character in the input string. Copy text
0064 300 : up to the control, if any, to the output buffer.
0064 301 :
0064 302
0064 303 MAIN_SCAN:
F4 BD F0 AD 7E D4 0064 304 CLRQ -(SP) ; indicate control not found
0066 305 LOCC #EXCL,INLEN(FP),@INPTR(FP) ; search for control char
006C 306 BEQL 10$ ; branch if not found
006E 307 INCL (SP) ; set indicator to show char. found
0070 308 10$:
56 F0 AD 50 A3 0070 309 SUBW3 R0,INLEN(FP),R6 ; calculate bytes to move
F0 AD 50 D0 0075 310 MOVL R0,INLEN(FP) ; update input length remaining
59 56 A2 0079 311 SUBW R6,R9 ; update and test output length
007C 312 BLSS OVERFLOW ; not enough room, error exit
6A F4 BD 56 28 007E 313 MOV3 R6,@INPTR(FP),(R10) ; move text part of input string
74 8E E9 0083 314 BLBC (SP)+,DONE ; leave if no controls left
F4 AD 51 D0 0086 315 MOVL R1,INPTR(FP) ; update input address pointer
5A 53 D0 008A 316 MOVL R3,R10 ; update output address pointer
7A 10 008D 317 BSBB GETCHAR ; skip control char
008F 318
008F 319 :

```

```

008F 320 : Parse the directive which has been found in the input string. Set
008F 321 : up: R0 = remaining count in CNTRL_TABLE
008F 322 : R4 = second char if two-char directive
008F 323 : R5 = repeat count
008F 324 : R6 = field width
008F 325 :
008F 326 :
008F 327 PARSE_DIRECTIVE:
008F 328
55 01 D0 008F 329 MOVL #1,R5 ; default repeat count is 1
52 D4 0092 330 CLRL R2 ; paren indicator ( not found yet )
007F 30 0094 331 BSBW GETCOUNT ; pull off count, if any
70 10 0097 332 BSBB GETCHAR ; get next char from input string
53 28 91 0099 333 CMPB #^A/(/,R3 ; was next char a paren?
0D 12 009C 334 BNEQ 20$ ; branch if not
52 D6 009E 335 INCL R2 ; set paren found indicator
56 D5 00A0 336 TSTL R6 ; was there a repeat count?
03 19 00A2 337 BLSS 10$ ; no..use default
55 56 D0 00A4 338 MOVL R6,R5 ; else get repeat count
00A7 339 10$:
6D 10 00A7 340 BSBB GETCOUNT ; look for field width
5E 10 00A9 341 BSBB GETCHAR ; get next char
00AB 342 20$:
FF5F CF 10 53 3A 00AB 343 LOCC R3,#CNTRL_LENGTH,CNTRL_TABLE ; check character in table
39 13 00B1 344 BEQL ILLEGAL ; illegal directive exit
08 50 D1 00B3 345 CMPL R0,#ONECHAR_INDEX ; is this a one char directive?
05 15 00B6 346 BLEQ 30$ ; yes, don't need any more
4F 10 00B8 347 BSBB GETCHAR ; get second control char
54 53 D0 00BA 348 MOVL R3,R4 ; move to R4 for return
00BD 349 30$:
02 52 E9 00BD 350 BLBC R2,40$ ; skip if no paren found
47 10 00C0 351 BSBB GETCHAR ; else skip paren char
00C2 352 40$:
53 10 50 C3 00C2 353
00C2 354 SUBL3 R0,#CNTRL_LENGTH,R3 ; compute offset for case table
00C6 355
00C6 356 :
00C6 357 : The following does a BSBB to the case dispatch
00C6 358 : table. The service routines do an RSB and return into CASE_LOOP.
00C6 359 :
00C6 360
02 11 00C6 361 BRB CASE_LOOP ; start processing loop
00C8 362 CASE_BSB:
05 10 00C8 363 BSBB FAO_CASE ; dispatch next directive
00CA 364 CASE_LOOP:
FB 55 F4 00CA 365 SOBGEQ R5,CASE_BSB ; repeat as specified
95 11 00CD 366 BRB MAIN_SCAN ; else continue string processing
00CF 367
00CF 368 :
00CF 369 : Here is the main dispatch table for dispatching FAO service
00CF 370 : routines. The case is entered via BSBB from CASE_BSB. The routines
00CF 371 : RSB to CASE_LOOP. Since the 5 numeric conversion directives all
00CF 372 : dispatch to the same routine, the case has a base of 5 and the
00CF 373 : numeric directives fall through to the statement following the CASE.
00CF 374 :
00CF 375 : Registers R0, R1, and R2 may be scratched by service routines.
00CF 376 :

```

```

00CF 377
00CF 378 FAO_CASE:
00CF 379 CASE R3,<- ; dispatch to service routine
00CF 380 CVTASC,- ; ascii string insertion
00CF 381 PERCENT,- ; insert ascii time, plural 'S', or UIC
00CF 382 REPEATI,- ; repeat character 'n' times
00CF 383 INCR_ARGPTR,- ; skip next parameter
00CF 384 DECR_ARGPTR,- ; backup to previous parameter
00CF 385 STARTFIELD,- ; define fixed length field
00CF 386 ENDFIELD,- ; terminate fixed length field
00CF 387 NEWLINE,- ; insert CR/LF
00CF 388 INSERT_CHAR,- ; insert TAB
00CF 389 INSERT_CHAR,- ; insert form feed
00CF 390 INSERT_CHAR,- ; insert '!'
00CF 391 >,B,#5 ; offset start by 5
00E9 392
00E0 31 00E9 393 BRW CVTNUM ; dispatch to numeric conversion
00EC 394
00EC 395
00EC 396 ILLEGAL:
50 14 3C 00EC 397 MOVZWL #SS$_BADPARAM,R0 ; error return code
0C 11 00EF 398 BRB FAO_EXIT
50 0601 8F 3C 00F1 399 OVERFLOW: MOVZWL #SS$_BUFFEROVF,R0 ; error return code
59 D4 00F6 401 CLRL R9 ; ensure correct return length
03 11 00F8 402 BRB FAO_EXIT
50 01 3C 00FA 403 DONE: MOVZWL #SS$_NORMAL,R0 ; no errors
08 AC D5 00FD 404 FAO_EXIT:
06 13 0100 406 TSTL OUTLEN(AP) ; was a return length required?
08 BC 0C BC 59 A3 0102 407 BEQL 10$ ; branch if not
04 0108 408 SUBW3 R9,@OUTDSC(AP),@OUTLEN(AP) ; compute and return output buffer length
04 0108 409 10$: RET
  
```

```

0109 411 .SBTTL GETCHAR - Routine to get next char from input string
0109 412
0109 413 :++
0109 414 :
0109 415 : FUNCTIONAL DESCRIPTION:
0109 416 :
0109 417 : This routine gets the next character from the input control
0109 418 : string, updating the length and address pointers. If the length
0109 419 : goes negative, an error exit is called.
0109 420 :
0109 421 : CALLING SEQUENCE:
0109 422 :
0109 423 : JSB (R8)
0109 424 :
0109 425 : INPUT PARAMETERS:
0109 426 :
0109 427 : none
0109 428 :
0109 429 : IMPLICIT INPUTS:
0109 430 :
0109 431 : INLEN(FP) - lower word has remaining length of input string
0109 432 : INPTR(FP) - is pointer to current string position
0109 433 :
0109 434 : OUTPUTS:
0109 435 :
0109 436 : R3 - next character in input string
0109 437 :
0109 438 : IMPLICIT OUTPUTS:
0109 439 :
0109 440 : none
0109 441 :
0109 442 : COMPLETION CODES:
0109 443 :
0109 444 : none
0109 445 :
0109 446 : SIDE EFFECTS:
0109 447 :
0109 448 : input pointers on stack are updated
0109 449 : error may cause jump to ILLEGAL
0109 450 :--
0109 451

```

```

53  F0 AD B7 0109 452 GETCHAR:
    DE 19 010C 453     DECW   INLEN(FP)           ; decr input length remaining
    F4 BD 9A 010E 454     BLSS  ILLEGAL           ; error if no more left
    F4 AD D6 0112 455     MOVZBL @INPTR(FP),R3      ; get next character
    05 0115 456     INCL  INPTR(FP)       ; update pointer
    457     RSB                ; return

```

```

0116 459 .SBTTL GETCOUNT - Routine to get repeat-count or field-width
0116 460
0116 461 :++
0116 462 :
0116 463 : FUNCTIONAL DESCRIPTION:
0116 464 :
0116 465 : This subroutine to PARSE DIRECTIVE scans for a repeat-count or
0116 466 : field-width in the directive in the input stream. If a numeric
0116 467 : count is found, it is converted to binary. If a '#' character
0116 468 : is found, the count is taken from the next parameter
0116 469 : in the parameter list.
0116 470 :
0116 471 : CALLING SEQUENCE:
0116 472 :
0116 473 : JSB or BSB
0116 474 :
0116 475 : INPUTS:
0116 476 :
0116 477 : R11 - parameter pointer
0116 478 :
0116 479 : IMPLICIT INPUTS:
0116 480 :
0116 481 : none
0116 482 :
0116 483 : OUTPUTS:
0116 484 :
0116 485 : R6 - value of count, if # or number found, else -1
0116 486 :
0116 487 : IMPLICIT OUTPUTS:
0116 488 :
0116 489 : R11 may be modified if a parameter is taken from the stack
0116 490 :
0116 491 : COMPLETION CODES:
0116 492 :
0116 493 : none
0116 494 :
0116 495 : SIDE EFFECTS:
0116 496 :
0116 497 : R1, R3, and R4 are destroyed
0116 498 :--
0116 499 :
0116 500 :
0116 501 :
0116 502 GETCOUNT:
0116 503 MNEGL #1,R6 ; not found indicator
F4 BD 23 91 0119 504 CMPB #^A/#/,@INPTR(FP) ; is this a param. count?
26 13 011D 505 BEQL 40$ ; yes .. pull next param
53 7C 011F 506 CLRQ R3 ; zero buffer for digit (R3)
0121 507 ; ... and accumulator for sum (R4)
51 F4 AD D0 0121 508 MOVL INPTR(FP),R1 ; remember where we were
0125 509 10$:
53 F4 BD 30 83 0125 510 SUBB3 #^A/0/,@INPTR(FP),R3 ; subtract ascii 0 from char
OF 19 012A 511 BLSS 20$ ; branch if not numeric
53 09 91 012C 512 CMPB #^A/9/-^A/0/,R3 ; still numeric?
OA 19 012F 513 BLSS 20$ ; no, branch
54 OA C4 0131 514 MULL2 #10,R4 ; shift for next digit
54 53 CO 0134 515 ADDL R3,R4 ; add in next digit

```

	D0	10	0137	516	BSBB	GETCHAR	:	skip digit we took
	EA	11	0139	517	BRB	10\$:	continue while numeric
			013B	518				
F4	AD	51	013B	519	CMPL	R1,INPTR(FP)	:	did we get any chars?
		03	013F	520	BEQL	30\$:	no, leave
56		54	D0	0141	MOVL	R4,R6	:	yes, return value
			0144	522				
			0144	523	RSB		:	return
			0145	524				
			0145	525				
56	8B	D0	0145	526	MOVL	(R11)+,R6	:	get value from next parameter
	BF	10	0148	527	BSBB	GETCHAR	:	skip '#'
		05	014A	528	RSB		:	return

```

0148 530 .SBTTL CVTASC - Insert ASCII string
0148 531
0148 532 :++
0148 533 :
0148 534 : FUNCTIONAL DESCRIPTION:
0148 535 :
0148 536 : Service routine to handle ASCII string insertions.
0148 537 : Strings are specified by several different methods. For
0148 538 : filled strings (AF) , non-printing characters are output
0148 539 : as dots ('.').
0148 540 :
0148 541 : CALLING SEQUENCE:
0148 542 :
0148 543 : JSB or BSB
0148 544 :
0148 545 : INPUTS:
0148 546 :
0148 547 : R3 - index of first control char in CNTRL_TABLE
0148 548 : R4 - second control character
0148 549 : R6 - output field width
0148 550 : R9 - output buffer length remaining
0148 551 : R10 - output buffer pointer
0148 552 : R11 - parameter pointer
0148 553 :
0148 554 : IMPLICIT INPUTS:
0148 555 :
0148 556 : none
0148 557 :
0148 558 : OUTPUTS:
0148 559 :
0148 560 : none
0148 561 :
0148 562 : IMPLICIT OUTPUTS:
0148 563 :
0148 564 : R9 and R10 are update to point to current position in output buffer
0148 565 : R11 is updated as parameters are taken from the stack
0148 566 :
0148 567 : ROUTINE VALUE:
0148 568 :
0148 569 : none
0148 570 :
0148 571 : SIDE EFFECTS:
0148 572 :
0148 573 : R7 and R8 are destroyed
0148 574 :--
0148 575 :
0148 576 :
0148 577 CVTASC:
0148 578
0148 579 PUSHR #*M<R3,R4,R5,R6> ; save registers
0148 580 CLRL R7 ; set filled indicator to not filled
0148 581 LOCC R4,#4,STRING_TYPES ; search for string subtype
0148 582 BEQL 110$ ; error if not found
0148 583
0148 584 :
0148 585 : R0 = 1 - filled , 2 - 2 arg desc. , 3 - str. desc. , 4 - cstring
0148 586 :

```

```

0078 8F BB
FEC9 CF 04 57 D4
54 3A
70 13

```

```

0159 587 CASE R0,<10$,20$,30$>,B,#2 ; case on descriptor type, base = 2
0163 588
0163 589 :
0163 590 : Case falls through here for filled ascii strings. Two argument
0163 591 : descriptor is used.
0163 592 :
0163 593 :
57 D6 0163 594 INCL R7 ; set filled indicator for filled ascii
51 8B 7D 0165 595 10$: MOVQ (R11)+,R1 ; get length and address
OE 11 0168 597 BRB 40$ ; continue
016A 598
016A 599 :
016A 600 : Standard system string descriptor
016A 601 :
016A 602 :
51 9B 7D 016A 603 20$: MOVQ @(R11)+,R1 ; move descriptor to R1,R2
51 51 3C 016D 605 MOVZWL R1,R1 ; make sure length is word
06 11 0170 606 BRB 40$ ; continue
0172 607
0172 608 :
0172 609 : Ascii counted string, first byte contains length
0172 610 :
0172 611 :
52 8B D0 0172 612 30$: MOVL (R11)+,R2 ; address of counted string
51 82 9A 0175 614 MOVZBL (R2)+,R1 ; get length and skip byte count
0178 615
0178 616 40$:
0178 617
0178 618 :
0178 619 : Here, R1 has string length, R2 has string address. Check length against
0178 620 : specified field width to decide how much string to move.
0178 621 :
0178 622 :
58 56 D0 0178 623 MOVL R6,R8 ; was a width specified?
03 18 0178 624 BGEQ 50$ ; branch if so
58 51 D0 017D 625 MOVL R1,R8 ; if not, use string length instead
0180 626 50$:
0180 627
0180 628 :
0180 629 : The string is moved to the output buffer with blank fill at the
0180 630 : end. The output pointers are then updated by the field width, so
0180 631 : that the string will be truncated if it was longer than the field
0180 632 : width. If the string is filled, a second pass is made to change
0180 633 : non-printing characters to dots.
0180 634 :
0180 635 :
56 59 D0 0180 636 MOVL R9,R6 ; copy remaining char count
0183 637 ; NOTE we have to use R6 here.
59 58 C2 0183 638 SUBL R8,R9 ; update length remaining
03 19 0186 639 BLSS 55$ ; Overflow, use remaining length.
56 58 D0 0188 640 MOVL R8,R6 ; else move only required length
6A 56 20 62 51 2C 018B 641 55$: MOVCS R1,(R2),#^A/ /,R6,(R10) ; move string, fill at end
0191 642
52 5A D0 0191 643 MOVL R10,R2 ; save output address

```



```

5A 56 C0 0194 644 ADDL R6,R10 ; update output pointer
 23 57 E9 0197 645 BLBC R7,90$ ; all done if not filled ASCII
54 62 9A 019A 646 60$: MOVZBL (R2),R4 ; R7 will now become loop counter.
      019D 648 ; Fetch character
      019D 649 ; Check for 7 bit printing (left half of DEC169)
      019D 650 ;
20 54 91 019D 651 CMPB R4,#^040 ; Less than space?
 12 1F 1F 01A0 652 BLSSU 70$ ; if yes,
7E 8F 54 91 01A2 653 CMPB R4,#^0176 ; Less than delete?
 0F 1B 1B 01A6 654 BLEQU 80$ ; yes, printing GL
      01A8 655 ;
      01A8 656 ; Check for 8 bit printing. note that space with 8th bit set is non-printing.
      01A8 657 ;
AO 8F 54 91 01A8 658 CMPB R4,#^X80+^040 ; delete or C1 control?
 06 1B 1B 01AC 659 BLEQU 70$ ; if yes,
FF 8F 54 91 01AE 660 CMPB R4,#^XFF ; 8 bit "delete" is non-printing
 03 12 12 01B2 661 BNEQU 80$ ; GR printing if not
62 2E 90 01B4 662 70$: MOVB #^A/./,(R2) ; Set character to "."
      01B7 664 80$:
DD 57 52 D6 01B7 665 INCL R2 ; point to next character
 56 F3 01B9 666 AOBLEQ R6,R7,60$ ; continue until done
      01BD 667 90$:
      59 D5 01BD 668 TSTL R9 ; Did we get result overflow above?
      05 19 01BF 669 BLSS 100$ ; Yes, branch to tell user.
0078 8F BA 01C1 670 POPR #^M<R3,R4,R5,R6>
      05 01C5 671 RSB ; return
      01C6 672
      01C6 673 100$:
      FF28 31 01C6 674 BRW OVERFLOW
      01C9 675 110$:
      FF20 31 01C9 676 BRW ILLEGAL

```

```

01CC 678      .SBTTL CVTNUM - Convert numeric parameter to ASCII
01CC 679
01CC 680      :++
01CC 681      :
01CC 682      : FUNCTIONAL DESCRIPTION:
01CC 683      :
01CC 684      :     This routine handles the various HEX, OCTAL, and DECIMAL
01CC 685      :     conversions. The proper field is extracted from the
01CC 686      :     parameter (byte, word, or long) and the needed output
01CC 687      :     width is determined. This is compared with the user
01CC 688      :     specified field width to determine if padding or filling
01CC 689      :     is needed. The entire field with fill is built on the
01CC 690      :     stack and then moved so that the result will be correct
01CC 691      :     on buffer overflow.
01CC 692      :
01CC 693      : CALLING SEQUENCE:
01CC 694      :
01CC 695      :     JSB or BSB
01CC 696      :
01CC 697      : INPUTS:
01CC 698      :
01CC 699      :     R3      - index of directive in CNTRL_TABLE.
01CC 700      :             0 = Octal
01CC 701      :             1 = heX
01CC 702      :             2 = Unsigned decimal
01CC 703      :             3 = Signed decimal
01CC 704      :             4 = Zero filled unsigned decimal
01CC 705      :     R4      - second char of directive (B,W, or L)
01CC 706      :     R6      - field width, or -1 if none
01CC 707      :     R9      - output length remaining
01CC 708      :     R10     - output position pointer
01CC 709      :     R11     - next parameter pointer
01CC 710      :
01CC 711      : IMPLICIT INPUTS:
01CC 712      :
01CC 713      :     none
01CC 714      :
01CC 715      : OUTPUTS:
01CC 716      :
01CC 717      :     none
01CC 718      :
01CC 719      : IMPLICIT OUTPUTS:
01CC 720      :
01CC 721      :     none
01CC 722      :
01CC 723      : ROUTINE VALUE:
01CC 724      :
01CC 725      :     none
01CC 726      :
01CC 727      : SIDE EFFECTS:
01CC 728      :
01CC 729      :     none
01CC 730      : --
01CC 731      :
01CC 732      :
01CC 733      : The registers will be set up as follows
01CC 734      :

```

```

01CC 735 : R0 - max digits to be output
01CC 736 : R1 - 0 -> byte, 1 -> word, 2 -> long
01CC 737 : R2 - value to be converted
01CC 738 : R4 - conversion radix
01CC 739 : R5 - sign indicator, 1 -> sign to be output, 0 otherwise
01CC 740 : R7 - fill character, (blank, zero for !Z, or * on width too small
01CC 741 : for decimal conversions)
01CC 742 : R8 - total width of field to be output
01CC 743 :
01CC 744 :
01CC 745 CVTNUM:
01CC 746
01CC 747 PUSHR #M<R3,R4,R5>
01CE 748
FE50 CF 03 54 3A 01CE 749 LOCC R4,#3,DATA_TYPES ; determine data type
03 12 01D4 750 BNEQ 10$ ; continue if legal directive
FF13 31 01D6 751 BRW !LEGAL ; else take error condition
01D9 752 10$:
51 03 50 C3 01D9 753 SUBL3 R0,#3,R1 ; convert to index
52 88 D0 01DD 754 MOVL (R1)+,R2 ; get next longword parameter
52 52 FE46 CF41 00 EF 01E0 755 EXTZV #0,FIELDS[R1],R2,R2 ; select proper field
55 D4 01E8 756 CLRL R5 ; note unsigned
57 20 90 01EA 757 MOVB #A/ /,R7 ; default fill char is blank
54 FE48 CF43 9A 01ED 758 MOVZBL RADIX[R3],R4 ; get conversion radix
01F3 759
01F3 760 :
01F3 761 : Case on the type of conversion. Note that base is set
01F3 762 : so that octal and hex conversions fall through case table.
01F3 763 :
01F3 764 :
01F3 765 CASE R3,<40$,30$,20$>,,#2 ; base index of 2
01FD 766
01FD 767 :
01FD 768 : Octal and Hex fall through here
01FD 769 :
01FD 770 :
50 50 6143 DE 01FD 771 MOVAL (R1)[R3],R0 ; compute index OCT_HEX_DIGITS
FE2D CF40 9A 0201 772 MOVZBL OCT_HEX_DIGITS[R0],R0 ; get number of digits to output
58 56 D0 0207 773 MOVL R6,R8 ; user specified width?
03 18 020A 774 BGEQ 15$ ; yes, use it as width
58 50 D0 020C 775 MOVL R0,R8 ; else take needed space
020F 776 15$:
50 58 D1 020F 777 CMLP R8,R0 ; width lss default digits?
45 18 0212 778 BGEQ 60$ ; no, fill to user specified width
50 58 D0 0214 779 MOVL R8,R0 ; else output only specified width
40 11 0217 780 BRB 60$
0219 781
0219 782 :
0219 783 : Unsigned decimal with zero fill
0219 784 :
0219 785 :
0219 786 20$:
57 30 90 0219 787 MOVB #A/0/,R7 ; insert new fill char
11 11 021C 788 BRB 40$ ; continue with normal dec. code
021E 789
021E 790 :
021E 791 : Signed decimal conversion

```

SY
Sy
AR
AS
CA
CA
CH
CH
CH
CH
CH
CH
CN
CN
CR
CV
CV
DA
DE
DO
EN
EX
EX
EX
EX
FA
FA
FA
FF
FI
FI
FI
GE
GE
HA
ID
IL
IL
IN
IN
IN
IN
IN
IN
IN
LA
LF
MA
NE
OC
ON
ON
OF
OF
OF
OF
OF
OF


```

F1 51 50 F2 027F 849 AOBLS R0,R1,10$ ; one more digit, done yet?
      05 55 E9 0283 850 BLBC R5,20$ ; branch if no sign to output
      7B 2D 90 0286 851 MOVB #^A/-/,-(R11) ; output sign
      51 D6 0289 852 INCL R1 ;
      028B 853 20$:
      028B 854
      028B 855 ;
      028B 856 ; If field (R8) is not full, then fill remainder with the fill character
      028B 857 ;
      028B 858
      03 11 028B 859 BRB 40$ ; start the loop
      7B 57 90 028D 860 30$:
      028D 861 MOVB R7,-(R11) ; insert fill character
      0290 862 40$:
      F9 51 58 F3 0290 863 AOBLEQ R8,R1,30$ ; fill until full
      0294 864
      0294 865 ;
      0294 866 ; Now copy stack back to buffer, checking for overflow
      0294 867 ;
      0294 868
      08 11 0294 869 BRB 70$ ; start loop
      0296 870 50$:
      02 59 F4 0296 871 SOBGEQ R9,60$ ; update length, check for overflow
      21 11 0299 872 BRB INSERT_OVF ; handle overflow
      8A 8B 90 029B 873 60$: MOVB (R11)+,(R10)+ ; move char to output buffer
      F5 58 F4 029E 874 70$: SOBGEQ R8,50$ ; move entire string
      02A1 875
      02A1 876 ;
      02A1 877 ; Now clean up mess on stack
      02A1 878 ;
      02A1 879
      SE 5B D0 02A1 880 MOVL R11,SP ; restore stack
      0841 8F BA 02A4 881 POPR #^M<R0,R6,R11> ; remove top of stack and restore regs
      02A8 882
      02A8 883 ;
      02A8 884 ; Restore registers and return from service routine.
      02A8 885 ;
      02A8 886
      38 BA 02A8 887 POPR #^M<R3,R4,R5>
      05 02AA 888 RSB

```

SY
Ps

PS
--

SA
YF

Ph

--

In

Co

Pa

Sy

Pa

Sy

Ps

Cr

As

Th

81

Th

41

14

Ma

--

-S

-S

TO

94

Th

MA

```

02AB 890      .SBTTL QUICKSERVE - Small service routines
02AB 891
02AB 892      :++
02AB 893      :
02AB 894      : FUNCTIONAL DESCRIPTION:
02AB 895      :
02AB 896      :     Following are a collection of short service routines for
02AB 897      :     FAO directives.
02AB 898      :
02AB 899      : CALLING SEQUENCE:
02AB 900      :
02AB 901      :     JSB or BSB
02AB 902      :
02AB 903      : INPUTS:
02AB 904      :
02AB 905      :     R3 - index in CNTRL_TABLE of the directive
02AB 906      :     R4 - second character of two-char directive, if any
02AB 907      :     R6 - user specified field width, if any (ignored for singal char
02AB 908      :           and argument directives)
02AB 909      :     R9 - output length remaining
02AB 910      :     R10 - output position pointer
02AB 911      :
02AB 912      : IMPLICIT INPUTS:
02AB 913      :
02AB 914      :     none
02AB 915      :
02AB 916      : OUTPUTS:
02AB 917      :
02AB 918      :     none
02AB 919      :
02AB 920      : IMPLICIT OUTPUTS:
02AB 921      :
02AB 922      :     R9 and R10 are modified
02AB 923      :
02AB 924      : COMPLETION CODES:
02AB 925      :
02AB 926      :     none
02AB 927      :
02AB 928      : SIDE EFFECTS:
02AB 929      :
02AB 930      :     none
02AB 931      : --
02AB 932      :
02AB 933      :
02AB 934      : INCR_ARGPTR:
02AB 935      :
02AB 936      :
02AB 937      : Directive to skip next parameter in parameter list
02AB 938      :
02AB 939      :
02AB 940      :     TSTL      (R11)+      ; skip next parameter
02AB 941      :     RSB              ; exit
02AB 942      :
02AB 943      : DECR_ARGPTR:
02AB 944      :
02AB 945      :
02AB 946      : Directive to back up and reuse last parameter in parameter list

```

8B D5
05

```

02AE 947 ;
02AE 948 ;
7B D5 02AE 949 TSTL -(R11) ; back up argument pointer
05 02B0 950 RSB ; exit
02B1 951
02B1 952 NEWLINE:
02B1 953
02B1 954 ;
02B1 955 ; Insert carriage return, line feed into output buffer
02B1 956 ;
02B1 957
02 59 F4 02B1 958 SOBGEQ R9,10$ ; room for CR?, branch if so
06 11 02B4 959 BRB INSERT_OVF ; no room in output buffer
02B6 960 10$:
8A 0D 90 02B6 961 MOVB #CR,(R10)+ ; insert CR in output buffer
02B9 962 ; continue for LF insertion
02B9 963
02B9 964 INSERT_CHAR:
02B9 965
02B9 966 ;
02B9 967 ; Make simple one character insertion in the output buffer.
02B9 968 ;
02B9 969
03 59 F4 02B9 970 SOBGEQ R9,INSERT_IT ; check length, branch if ok
FE32 31 02BC 971 INSERT_OVF:
02BC 972 BRW OVERFLOW ; error , no room in output buffer
02BF 973 INSERT_IT:
02BF 974
02BF 975 ;
02BF 976 ; Insert the character by computing the index into the replacement table
02BF 977 ;
02BF 978
8A FDSF CF43 90 02BF 979 MOVB REPLACEMENT-REPL_OFFSET[R3],(R10)+ ; insert the char
05 02C5 980 RSB
02C6 981
02C6 982 ;
02C6 983 ; Directive to repeat a particular character 'n' times, where 'n' is
02C6 984 ; specified by the field width in the directive.
02C6 985 ;
02C6 986
02C6 987 REPEATIT:
38 BB 02C6 988 PUSHR #*M<R3,R4,R5> ; save regs for MOVCS clobber
56 D5 02C8 989 TSTL R6 ; check if width was specified
15 19 02CA 990 BLSS ILLFIELD ; illegal if none specified
59 56 C2 02CC 991 SUBL R6,R9 ; compute remaining output length
EB 19 02CF 992 BLSS INSERT_OVF ; not enough room, error
6A 56 54 6E 00 2C 02D1 993 MOVCS #0,(SP),R4,R6,(R10) ; fill with specified character
5A 56 C0 02D7 994 ADDL R6,R10 ; update output pointer
38 BA 02DA 995 POPR #*M<R3,R4,R5> ; restore regs
05 02DC 996 RSB
02DD 997
02DD 998 ;
02DD 999 ; The following are the directives which define a fixed length field.
02DD 1000 ; The field width is specified with the define field directive. At the
02DD 1001 ; end field directive, any of the field remaining is blank filled, else
02DD 1002 ; the field is truncated to the specified length.
02DD 1003 ;

```

```

02DD 1004
02DD 1005 STARTFIELD:
56 03 D5 02DD 1006 TSTL R6 ; did user specify field (must be specified)
03 18 02DF 1007 BGEQ STARTOK ; yes, continue
FE08 31 02E1 1008 ILLFIELD:
FC AD 5A 56 C1 02E1 1009 BRW ILLEGAL ; illegal directive
59 56 D1 02E4 1010 STARTOK:ADDL3 R6,R10,FIELDEND(FP) ; compute and save ending address
CE 14 02E9 1011 CMPL R6,R9 ; was that much space remaining?
05 02EC 1012 BGTR INSERT_OVF ; no, take error here
02EE 1013 RSB ; return
02FF 1014
02EF 1015 ;
02EF 1016 ; Set up registers so that if fill is needed, a phony call is made
02EF 1017 ; to REPEATIT with the length in R6 and the 'blank' character in R4
02EF 1018 ;
02EF 1019 ;
02EF 1020 ENDFIELD:
56 54 20 9A 02EF 1021 MOVZBL #^A/ /,R4 ; generate blank fill character
FC AD 5A C3 02F2 1022 SUBL3 R10,FIELDEND(FP),R6 ; compute remaining field length
CD 14 02F7 1023 BGTR REPEATIT ; if any left, go fill with blanks
SA FC AD D0 02F9 1024 MOVL FIELDEND(FP),R10 ; else truncate by setting back pointer
59 56 C2 02FD 1025 SUBL R6,R9 ; subtract negative difference from counter
05 0300 1026 RSB ; return

```



```

0301 1028 .SBTTL PERCENT - Time directives, plural 'S', and UIC
0301 1029
0301 1030 :++
0301 1031 :
0301 1032 : FUNCTIONAL DESCRIPTION:
0301 1033 :
0301 1034 : These directives are for date and time conversion, for
0301 1035 : conditionally inserting a plural 'S' into messages, and UIC conversion.
0301 1036 : The time directives insert an ASCII time string into the output buffer.
0301 1037 : The user may supply a quadword binary time to be converted,
0301 1038 : or have the current date or time inserted.
0301 1039 :
0301 1040 : CALLING SEQUENCE:
0301 1041 :
0301 1042 : JSB/BSB
0301 1043 :
0301 1044 : INPUTS:
0301 1045 :
0301 1046 : R4 - second character of directive. D -> convert
0301 1047 : date and time, T -> convert time only
0301 1048 : S -> plural indicator, U -> convert UIC
0301 1049 : I -> identifier
0301 1050 : R6 - user specified field width, if any
0301 1051 : R9 - remaining length of output buffer
0301 1052 : R10 - current output buffer position
0301 1053 : R11 - next parameter address
0301 1054 :
0301 1055 : IMPLICIT INPUTS:
0301 1056 :
0301 1057 : none
0301 1058 :
0301 1059 : OUTPUTS:
0301 1060 :
0301 1061 : none
0301 1062 :
0301 1063 : IMPLICIT OUTPUTS:
0301 1064 :
0301 1065 : none
0301 1066 :
0301 1067 : ROUTINE VALUE:
0301 1068 :
0301 1069 : none
0301 1070 :
0301 1071 : SIDE EFFECTS:
0301 1072 :
0301 1073 : none
0301 1074 : --
0301 1075 :
000001FC 0301 1076 ID_REG_MASK= ^M<R2,R3,R4,R5,R6,R7,R8> ; %I & %U WORK REG MASK
0301 1077 :
FD20 CF 05 54 3A 0301 1078 PERCENT:
D8 13 0301 1079 LOCC R4,#5,PERCENT_STR ; find directive type
57 D4 0307 1080 BEQL ILLFIELD ; illegal directive if not found
0309 1081 CLRL R7 ; assume date and time
0308 1082 CASE R0,<5$,10$,30$,70$,210$>,B,#1 ; branch on directive type
0319 1083 :
0319 1084 :

```

```

0319 1085 ; Time only directive falls through here
0319 1086 ;
0319 1087 ;
0319 1088 5$:
57 D6 0319 1089 INCL R7 ; indicate time only
031B 1090 10$: ; time and date enters here
6A 59 20 6A 38 BB 031B 1091 PUSHR #M<R3,R4,R5> ; save registers
58 7E DE 031D 1092 MOVCS #0,(R10),#A/ /,R9,(R10) ; blank fill rest of output buffer
7E 59 7D 0323 1093 MOVAL -(SP),R8 ; space for return length
52 6E DE 0326 1094 MOVQ R9,-(SP) ; form descriptor for output buffer
51 8B D0 0329 1095 MOVAL (SP),R2 ; get address of buffer descriptor
04 A1 04 13 032C 1096 MOVL (R11)+,R1 ; get binary time address
61 D1 032F 1097 BEQL 12$ ; branch if no address
0331 1098 CMPL (R1),4(R1) ; let potential access violation
0335 1099 ; ...happen in this frame rather than
0335 1100 ; ...within $ASCTIM to help condition
0335 1101 ; ...handler
52 56 D0 0335 1102 12$: $ASCTIM_S (R8),(R2),(R1),R7 ; convert time to ascii
03 18 0344 1103 MOVL R6,R2 ; did user specify width?
52 68 3C 0347 1104 BGEQ 20$ ; yes, use it
59 52 C2 0349 1105 MOVZWL (R8),R2 ; else use returned length
12 19 034C 1106 20$:
5A 52 C0 034C 1107 SUBL R2,R9 ; update output length
5E 0C C0 034F 1108 BLSS 40$ ; error, not enough room
38 BA 0351 1109 ADDL R2,R10 ; update output buffer
05 05 0354 1110 ADDL #12,SP ; pop locals from stack
0357 1111 POPR #M<R3,R4,R5> ; restore registers
0359 1112 RSB ;
035A 1113 30$:
035A 1114 ;
035A 1115 ;
035A 1116 ; Check if the last value converted was equal to one. If so, then do
035A 1117 ; nothing, else output an 'S' into the output buffer.
035A 1118 ;
035A 1119 ;
F8 AD 01 D1 035A 1120 CMPL #1, LASTVAL(FP) ; was last value a one
13 13 035E 1121 BEQL 60$ ; yes, simply return
03 59 F4 0360 1122 SOBGEQ R9,50$ ; check if room in buffer
FD 8B 31 0363 1123 40$: BRW OVERFLOW ; no room, error
8A 53 8F 90 0366 1124 50$: MOVB #A/S/, (R10)+ ; plural, insert 'S'
04 FE AA 05 E1 036A 1125 BBC #5,-2(R10),60$ ; continue if previous character was
036F 1126 ; ...upper case
FF AA 20 88 036F 1127 BISB #X20,-1(R10) ; else convert upper 'S' to lower 's'
05 05 0373 1128 60$: RSB ; return
0374 1129 ;
0374 1130 ;
0374 1131 ; Convert a longword value to an identifier if possible. This identifier may
0374 1132 ; take one of two forms, a random identifier or an alphanumeric UIC. In the
0374 1133 ; case of an alphanumeric UIC, an attempt is first made to translate just the
0374 1134 ; group portion of the UIC. If this fails, an attempt is made to translate
0374 1135 ; the entire UIC. If this also fails, the UIC is formatted using the %U
0374 1136 ; directive.
0374 1137 ;
01FC 8F BB 0374 1138 70$: PUSHR #ID_REG_MASK ; SAVE WORK REGS
5E 20 C2 0378 1139 SUBL2 #32,SP ; GROUP IDENTIFIER STORAGE
7E 5E D0 037B 1140 MOVL SP,-(SP) ; GROUP IDENTIFIER
20 DD 037E 1141 PUSHL #32 ; DESCRIPTOR

```

```

57 5E D0 0380 1142      MOVL   SP,R7          ; SAVE DESCRIPTOR ADDRESS FOR LATER
5E 20 C2 0383 1143      SUBL2  #32,SP         ; USER IDENTIFIER STORAGE
7E 5E D0 0386 1144      MOVL   SP,-(SP)      ; USER IDENTIFIER
20 DD 0389 1145      PUSHL  #32           ; DESCRIPTOR
58 5E D0 038B 1146      MOVL   SP,R8         ; SAVE DESCRIPTOR ADDRESS FOR LATER
                                038E 1147
                                038E 1148      ASSUME  UIC$K_UIC FORMAT EQ 0
                                038E 1149      ASSUME  UIC$K_ID FORMAT EQ 2
                                038E 1150      ASSUME  UIC$V_FORMAT EQ 30
                                038E 1151
52 6B D0 038E 1152      MOVL   (R11),R2      ; GET THE IDENTIFIER NUMBER
27 19 0391 1153      BLSS   75$           ; XFER IF NOT A UIC
52 01 AE 0393 1154      MNEGW  #1,R2         ; SET UP FOR GROUP IDENTIFIER CHECK
                                0396 1155      $IDTOASC_S          ID=R2,-
                                0396 1156                        NAMLEN=(R7),-
                                0396 1157                        NAMBUF=(R7)
OE 50 E9 03A9 1158      BLBC   R0,75$        ; XFER IF ERRORS IN TRANSLATING
                                03AC 1159
                                03AC 1160      ASSUME  UIC$K_WILD_MEMBER EQ <^XFFFF>
                                03AC 1161
52 6B B1 03AC 1162      CMPW   (R11),R2      ; WILD MEMBER (R2 SET ABOVE)
OB 12 03AF 1163      BNEQ   80$           ; XFER IF NOT
68 01 D0 03B1 1164      MOVL   #1,(R8)       ; ELSE SET SIZE
04 B8 2A 90 03B4 1165      MOVB   #^A*\,\,@4(R8) ; SET WILDCARD CHARACTER
1A 11 03B8 1166      BRB    90$           ; GO BUILD UIC
67 B4 03BA 1167 75$:   CLRW   (R7)          ; ELSE SET FOR ZERO SIZE
                                80$:   $IDTOASC_S          ID=(R11),-
                                03BC 1168                        NAMLEN=(R8),-
                                03BC 1169                        NAMBUF=(R8)
                                03BC 1170
02 50 E8 03CF 1171      BLBS   R0,90$        ; XFER IF NO ERRORS
68 B4 03D2 1172      CLRW   (R8)          ; ELSE SET ZERO SIZE
53 67 3C 03D4 1173 90$:  MOVZWL (R7),R3       ; GET GROUP NAME SIZE
02 13 03D7 1174      BEQL   100$         ; XFER IF GROUP DIDN'T TRANSLATE
53 D6 03D9 1175      INCL   R3           ; ELSE ACCOUNT FOR COMMA SEPARATOR
50 68 3C 03DB 1176 100$: MOVZWL (R8),R0      ; GET USER NAME SIZE
66 13 03DE 1177      BEQL   150$         ; XFER IF DIDN'T TRANSLATE
53 50 C0 03E0 1178      ADDL2  R0,R3        ; ELSE TOTAL UP THE SIZE
61 13 03E3 1179      BEQL   150$         ; XFER IF UIC DIDN'T TRANSLATE
                                03E5 1180
                                03E5 1181      ASSUME  UIC$K_ID FORMAT EQ 2
                                03E5 1182      ASSUME  UIC$V_FORMAT EQ 30
                                03E5 1183
03 6B 1F E0 03E5 1184      BBS    #31,(R11),105$ ; XFER IF NOT UIC
53 02 C0 03E9 1185      ADDL2  #2,R3        ; ELSE ACCOUNT FOR SQUARE BRACKETS
56 D5 03EC 1186 105$:  TSTL   R6           ; WIDTH SUPPLIED?
03 18 03EE 1187      BGEQ   110$         ; XFER IF SO
56 53 D0 03F0 1188      MOVL   R3,R6        ; ELSE SET IT
56 53 D1 03F3 1189 110$: CMPL   R3,R6        ; FIELD WIDTH EXCEEDED?
34 14 03F6 1190      BGTR   130$         ; XFER IF SO...NOTE IT
59 53 D1 03F8 1191      CMPL   R3,R9        ; BUFFER EXCEEDED?
3E 14 03FB 1192      BGTR   140$         ; XFER IF SO...NOTE IT
56 53 D0 03FD 1193      MOVL   R3,R6        ; ELSE PUTE LENGTH IN NONVOLATILE REG
53 5A D0 0400 1194      MOVL   R10,R3       ; GET OUTPUT BUFFER ADDRESS
                                0403 1195
                                0403 1196      ASSUME  UIC$K_ID FORMAT EQ 2
                                0403 1197      ASSUME  UIC$V_FORMAT EQ 30
                                0403 1198

```

```

10 6B 1F E0 0403 1199 BBS #31,(R11),120$ : XFER IF NOT UIC
83 5B 8F 90 0407 1200 MCVB #^A/[/, (R3)+ : NOTE START OF UIC
67 B5 040B 1201 TSTW (R7) : DID GROUP NAME TRANSLATE?
08 13 040D 1202 BEQL 120$ : XFER IF NOT
63 04 B7 67 28 040F 1203 MOV C3 (R7),@4(R7),(R3) : ELSE COPY GROUP NAME
83 2C 90 0414 1204 MOV B #^A/./ (R3)+ : SAVE SEPARATOR
63 04 B8 68 28 0417 1205 120$: MOV C3 (R8),@4(R8),(R3) : COPY USER NAME
04 6B 1F E0 041C 1206 BBS #31,(R11),125$ : XFER IF NOT UIC
83 5D 8F 90 0420 1207 MOV B #^A/] / (R3)+ : TIE OFF THE UIC
59 56 C2 0424 1208 125$: SUBL2 R6,R9 : CALC REMAINING ROOM IN THE BUFFER
5A 56 C0 0427 1209 ADDL2 R6,R10 : CALC NEXT AVAILABLE IN BUFFER
5D 11 042A 1210 BRB 190$ : GO FINISH UP
6A 56 2A 6E 00 2C 042C 1211 130$: MOV C5 #0,(SP),#^A/*/,R6,(R10) : NOTE FIELD OVERFLOW
59 56 C2 0432 1212 SUBL2 R6,R9 : ADJUST COUNT
5A 56 C0 0435 1213 ADDL2 R6,R10 : AND POINTER
FCB6 31 0438 1214 BRW OVERFLOW : NOTE OVERFLOW
6A 59 2A 6E 00 2C 043B 1215 140$: MOV C5 #0,(SP),#^A/*/,R9,(R10) : NOTE BUFFER OVERFLOW
59 D4 0441 1216 CLRL R9 : NO ROOM LEFT
FCAB 31 0443 1217 BRW OVERFLOW : NOTE OVERFLOW
0446 1218
0446 1219 : At this point, it has been determined that no translation exists for the
0446 1220 : specified identifier. If it is a UIC, format it using the %U. If it is
0446 1221 : a random identifier, try to convert it to a hex number.
0446 1222
FFFFFFF 8F 6B D1 0446 1223 150$: CML (R11),#-1 : IS THIS THE MATCH-ALL IDENTIFIER?
0B 12 044D 1224 BNEQ 170$ : XFER IF NOT
03 59 F4 044F 1225 SOBGEQ R9,160$ : XFER IF ROOM
FC9C 31 0452 1226 BRW OVERFLOW : ELSE NOTE OVERFLOW
8A 2A 90 0455 1227 160$: MOV B #^A*\ (R10)+ : NOTE THE MATCH-ALL IDENTIFIER
2F 11 0458 1228 BRB 190$ : GO FINISH WITH THIS DIRECTIVE
00 6B 02 1E ED 045A 1229 170$: CMPZV #UIC$V_FORMAT,#UIC$S_FORMAT,(R11),#UIC$K_UIC_FORMAT : UIC?
36 13 045F 1230 BEQL 200$ : XFER IF SO
5E 0000050 8F C0 0461 1231 ADDL2 #8+32+8+32,SP : CLEAN UP THE STACK
01FC 8F BA 0468 1232 POPR #ID_REG_MASK : RESTORE WORK REGS
03 59 F4 046C 1233 SOBGEQ R9,180$ : INSURE ROOM FOR %X
FC7F 31 046F 1234 BRW OVERFLOW
8A 25 90 0472 1235 180$: MOV B #^A*\ (R10)+
03 59 F4 0475 1236 SOBGEQ R9,185$
FC76 31 0478 1237 BRW OVERFLOW
8A 58 8F 90 047B 1238 185$: MOV B #^A*\ (R10)+
53 01 D0 047F 1239 MOVI #1,R3 : SET UP FOR HEX CONVERSION
54 4C 8F 9A 0482 1240 MOVZBL #^A/L/,R4
FD43 31 0486 1241 BRW CVTNUM : GO TRY TO CONVERT
5E 0000050 8F C0 0489 1242 190$: ADDL2 #8+32+8+32,SP : CLEAN UP THE STACK
01FC 8F BA 0490 1243 POPR #ID_REG_MASK : RESTORE WORK REGS
8B D5 0494 1244 TSTL (R1)+ : SET TO NEXT PARAMETER
05 0496 1245 RSB : RETURN FOR MORE
5E 0000050 8F C0 0497 1246 200$: ADDL2 #8+32+8+32,SP : CLEAN UP THE STACK
01FC 8F BA 049E 1247 POPR #ID_REG_MASK : RESTORE WORK REGISTERS
04A2 1248
04A2 1249 :
04A2 1250 : Convert the longword value to a UIC in a standard format. This format is
04A2 1251 : [group,member]. Where the group and member portions are a word (16-bits)
04A2 1252 : each. If a width is supplied, the UIC is centered (by the comma) in the
04A2 1253 : field.
04A2 1254 :
04A2 1255 :

```

		01FC	8F	BB	04A2	1256	210\$:	PUSHR	#ID_REG_MASK	:	SAVE WORK REGISTERS
		5E	10	C2	04A6	1257		SUBL2	#16,SP	:	MAKE ROOM FOR GROUP & MEMBER
		57	5E	D0	04A9	1258		MOVL	SP,R7	:	SET ADDRESS FOR GROUP
	58	08	AE	9E	04AC	1259		MOVAB	8(SP),R8	:	SET ADDRESS FOR MEMBER
		50	58	D0	04B0	1260		MOVL	R8,R0	:	SET ADDRESS OF MEMBER STRING
			80	B4	04B3	1261		CLRW	(R0)+	:	RESET CHARACTER COUNT
		52	0F	D0	04B5	1262		MOVL	#15,R2	:	SET STARTING BIT
51	68	10	00	EF	04B8	1263		EXTZV	#UIC\$V MEMBER,#UIC\$S MEMBER,(R11),R1	:	GET MEMBER NUMBER
	FFFF	8F	51	B1	04BD	1264		CMPW	R1,#UIC\$K_WILD_MEMBER	:	IS IT A WILDCARD MEMBER?
			09	12	04C2	1265		BNEQ	220\$:	XFER IF NOT
		68	01	B0	04C4	1266		MOVW	#1,(R8)	:	ELSE SET SIZE
		80	2A	90	04C7	1267		MOVB	#^A/^/, (R0)+	:	SET WILDCARD STRING
			001F	31	04CA	1268		BRW	250\$:	GO GET THE GROUP
53	51	03	52	EF	04CD	1269	220\$:	EXTZV	R2,#3,R1,R3	:	GET AN OCTAL DIGIT
			04	12	04D2	1270		BNEQ	230\$:	XFER IF NON-ZERO
			68	B5	04D4	1271		TSTW	(R8)	:	ELSE CHECK FOR ZERO SUPPRESSION
			06	13	04D6	1272		BEQL	240\$:	XFER IF SUPPRESSING
	80	53	30	81	04D8	1273	230\$:	ADDB3	#^A/O/,R3,(R0)+	:	CONVERT TO ASCII AND SAVE IT
			68	B6	04DC	1274		INCW	(R8)	:	ONE MORE CHARACTER
			52	03	C2	04DE	240\$:	SUBL2	#3,R2	:	SET FOR THE NEXT DIGIT
				EA	18	04E1	1276	BGEQ	220\$:	CONTINUE TILL ALL DONE
				68	B5	04E3	1277	TSTW	(R8)	:	ANYTHING THERE?
			05	12	04E5	1278		BNEQ	250\$:	XFER IF SO
		80	30	90	04E7	1279		MOVB	#^A/O/, (R0)+	:	ELSE SAVE AT LEAST ONE ZERO
			68	B6	04EA	1280		INCW	(R8)	:	COUNT IT
			50	D0	04EC	1281	250\$:	MOVL	R7,R0	:	SET ADDRESS OF GROUP STRING
				80	B4	04EF	1282	CLRW	(R0)+	:	RESET CHARACTER COUNT
			52	0F	D0	04F1	1283	MOVL	#15,R2	:	SET STARTING BIT
51	68	0E	10	EF	04F4	1284		EXTZV	#UIC\$V GROUP,#UIC\$S GROUP,(R11),R1	:	GET GROUP NUMBER
	3FFF	8F	51	B1	04F9	1285		CMPW	R1,#UIC\$K_WILD_GROUP	:	IS IT A WILDCARD GROUP?
			09	12	04FE	1286		BNEQ	260\$:	XFER IF NOT
			67	01	B0	0500		MOVW	#1,(R7)	:	ELSE SET SIZE
			80	2A	90	0503		MOVB	#^A/^/, (R0)+	:	SET WILDCARD STRING
				001F	31	0506		BRW	290\$:	GO GET THE GROUP
53	51	03	52	EF	0509	1290	260\$:	EXTZV	R2,#3,R1,R3	:	GET AN OCTAL DIGIT
			04	12	050E	1291		BNEQ	270\$:	XFER IF NON-ZERO
			67	B5	0510	1292		TSTW	(R7)	:	ELSE CHECK FOR ZERO SUPPRESSION
			06	13	0512	1293		BEQL	280\$:	XFER IF SUPPRESSING
	80	53	30	81	0514	1294	270\$:	ADDB3	#^A/O/,R3,(R0)+	:	CONVERT TO ASCII AND SAVE
			67	B6	0518	1295		INCW	(R7)	:	COUNT THE CHARACTER
			52	03	C2	051A	280\$:	SUBL2	#3,R2	:	SET FOR THE NEXT DIGIT
				EA	18	051D	1297	BGEQ	260\$:	CONTINUE TILL DONE
			67	B5	051F	1298		TSTW	(R7)	:	ANYTHING THERE?
			05	12	0521	1299		BNEQ	290\$:	XFER IF SO
		80	30	90	0523	1300		MOVB	#^A/O/, (R0)+	:	ELSE SAVE AT LEAST ONE ZERO
			67	B6	0526	1301		INCW	(R7)	:	COUNT IT
			88	D5	0528	1302	290\$:	TSTL	(R11)+	:	STEP OVER UIC
			50	68	B0	052A	1303	MOVW	(R8),R0	:	GET SIZE OF MEMBER FIELD
			50	67	A0	052D	1304	ADDW2	(R7),R0	:	AND GROUP FIELD
			50	03	A0	0530	1305	ADDW2	#3,R0	:	PLUS DELIMITERS
			50	50	3C	0533	1306	MOVZWL	R0,R0	:	FULL LONGWORD
				56	D5	0536	1307	TSTL	R6	:	ANY FIELD WIDTH GIVEN?
				16	18	0538	1308	BGEQ	300\$:	XFER IF SO
			53	5A	D0	053A	1309	MOVL	R10,R3	:	COPY ADDRESS OF OUTPUT FIELD
			56	50	D0	053D	1310	MOVL	R0,R6	:	SET FIELD WIDTH
			59	50	D1	0540	1311	CMPL	R0,R9	:	ELSE SEE IF THERE IS ROOM FOR THE UIC
				36	15	0543	1312	BLEQ	320\$:	XFER IF THERE IS ROOM

6A	59	2A	6E	00	2C	0545	1313		MOVCS	#0,(SP),#A/*/,R9,(R10)	:	ELSE FILL REMAINING ROOM
				59	D4	054B	1314		CLRL	R9	:	NO REMAINING ROOM
			FBA1	31	054D	1315			BRW	OVERFLOW	:	ELSE INDICATE THE OVERFLOW
			59	56	D1	0550	1316	300\$:	CMPL	R6,R9	:	IS THERE ROOM IN THE OUTPUT BUFFER?
				03	15	0553	1317		BLEQ	310\$:	XFER IF SO
			0046	31	0555	1318			BRW	330\$:	ELSE INDICATE OVERFLOW
			56	50	D1	0558	1319	310\$:	CMPL	R0,R6	:	IS THE ROOM IN THE FIELD FOR THE UIC?
				41	14	055B	1320		BGTR	330\$:	XFER IF NOT...FIELD WIDTH OVERFLOW
6A	56	20	6E	00	2C	055D	1321		MOVCS	#0,(SP),#A/ /,R6,(R10)	:	ELSE FILL FIELD FIRST
		51	56	03	C3	0563	1322		SUBL3	#3,R6,R1	:	CALC SIZE MINUS DELIMITERS
			51	02	C6	0567	1323		DIVL2	#2,R1	:	MAX SIZE FOR CENTERING
			51	67	B1	056A	1324		CMPL	(R7),R1	:	ROOM FOR GROUP SUBFIELD?
				2F	14	056D	1325		BGTR	330\$:	XFER IF NOT...F.W.O.
			51	68	B1	056F	1326		CMPL	(R8),R1	:	ROOM FOR MEMBER SUBFIELD?
				2A	14	0572	1327		BGTR	330\$:	XFER IF NOT...F.W.O.
			51	67	A2	0574	1328		SUBW2	(R7),R1	:	CALC NUMBER OF LEADING SPACES
		53	5A	51	C1	0577	1329		ADDL3	R1,R10,R3	:	ADJUST FOR LEADING SPACES
		83	5B	8F	90	057B	1330	320\$:	MOVB	#A/[/,(R3)+	:	FIRST DELIMITER
63	02	A7	67	28	057F	1331			MOVCS	(R7),2(R7),(R3)	:	COPY GROUP SUBFIELD
		83	2C	90	0584	1332			MOVB	#A/./,(R3)+	:	SUBFIELD DELIMITER
63	02	A8	68	28	0587	1333			MOVCS	(R8),2(R8),(R3)	:	COPY MEMBER SUBFIELD
		83	5D	8F	90	058C	1334		MOVB	#A/]/,(R3)+	:	TIE OFF THE UIC
			5A	56	C0	0590	1335		ADDL2	R6,R10	:	CALC NEXT AVAILABLE POSITION
			59	56	C2	0593	1336		SUBL2	R6,R9	:	CALC REMAINING BUFFER POSITIONS
			5E	10	C0	0596	1337		ADDL2	#16,SP	:	CLEAN UP THE STACK
			01FC	8F	BA	0599	1338		POPR	#ID_REG_MASK	:	RESTORE WORK REGISTERS
					05	059D	1339		RSB		:	AND RETURN
6A	56	2A	6E	00	2C	059E	1340	330\$:	MOVCS	#0,(SP),#A/*/,R6,(R10)	:	INDICATE OVERFLOW
			5A	56	C0	05A4	1341		ADDL2	R6,R10	:	POINT TO NEXT FIELD IN OUTPUT
			59	56	C2	05A7	1342		SUBL2	R6,R9	:	DEDUCT FIELD
			FB44	31	05AA	1343			BRW	OVERFLOW	:	FIELD WIDTH OVERFLOW ERROR

```

05AD 1345 .SBTTL HANDLER - Condition handler
05AD 1346 :++
05AD 1347 :
05AD 1348 : FUNCTIONAL DESCRIPTION:
05AD 1349 :
05AD 1350 : This condition handler is used to catch any errors which
05AD 1351 : occurred while processing the arguments, such as access
05AD 1352 : violation. This is because we don't want exceptions
05AD 1353 : occurring within the system service.
05AD 1354 : Care must be taken in this handler to deal with a second access
05AD 1355 : violation while storing the return value for $FAO.
05AD 1356 :
05AD 1357 : INPUTS:
05AD 1358 :
05AD 1359 : CHF$S_SIGARGLST(AP) = Address of signal vector
05AD 1360 : CHF$S_MCHARGLST(AP) = Address of mechanism vector
05AD 1361 :
05AD 1362 : OUTPUTS:
05AD 1363 :
05AD 1364 : The final R0 is set to the status code and the service
05AD 1365 : is exited via $UNWIND.
05AD 1366 :--
05AD 1367 :
05AD 1368 .WEAK EXE$$SIGTORET
05AD 1369
05AD 1370 HANDLER:
05AD 1371 .WORD 0
05AF 1372
6D 00000000'EF 9E 05AF 1373 MOVAB L^EXE$$SIGTORET,(FP) ;Simple handler for errors here
32 13 05B6 1374 BEQL 90$ ;*****
05B8 1375
05B8 1376 ASSUME CHF$S_MCHARGLST,EQ,CHF$S_SIGARGLST+4
04 A0 50 04 AC 7D 05B8 1377 MOVQ CHF$S_SIGARGLST(AP),R0 ; Get address of signal argument list
00000920 8F D1 05BC 1378 CMPL #SS$_UNWIND,CHF$S_SIG_NAME(R0) ;Unwinding?
24 13 05C4 1379 BEQL 90$ ;Exit if yes
08 A1 D5 05C6 1380 TSTL CHF$S_MCH_DEPTH(R1) ;Exception within FAO?
1A 12 05C9 1381 BNEQ 80$ ;Resignal if no
0C A1 04 A0 D0 05CB 1382 MOVL CHF$S_SIG_NAME(R0),CHF$S_MCH_SAVRO(R1) ;Set final return status
00000000'GF 02 FB 05D0 1383 CLRQ -(SP) ;Clear depth and new PC arguments
50 08 AD D0 05D9 1384 CALLS #2,G^SYSSUNWIND ;Unwind to establisher's caller
50 08 A0 D0 05DD 1385 ;**** The next instruction may ACCVIO
02 13 05E1 1386 MOVL SF$S_SAVE_AP(FP),R0 ;Get address of FAO's argument list
60 B4 05E3 1387 MOVL OUTLEN(R0),R0 ;Output length requested?
05E5 1388 BEQL 10$ ;Branch if not
50 0918 8F 3C 05E5 1389 CLRW (R0) ;Indicate nothing returned in buffer
04 05E5 1390 10$: ;**** End of potential ACCVIO
05EA 1391 80$: MOVZWL #SS$_RESIGNAL,R0 ;Resignal (ignore after UNWIND)
05EB 1392 90$: RET ;
05EB 1393
05EB 1394
05EB 1395 .END

```

SYSFAO
Symbol table

ARGCOUNT	=	00000000			OPS-ADDF3	=	00000041
ASC_NAMES		00000000	R	02	OPS-ADDG2	=	000040FD
CASE_BSB		000000C8	R	02	OPS-ADDG3	=	000041FD
CASE_LOOP		000000CA	R	02	OPS-ADDH2	=	000060FD
CHFSL_MCHARGLST	=	00000008			OPS-ADDH3	=	000061FD
CHFSL_MCH_DEPTH	=	00000008			OPS-ADDP4	=	00000020
CHFSL_MCH_SAVRO	=	0000000C			OPS-ADDP6	=	00000021
CHFSL_SIGARGLST	=	00000004			OPS-ASHP	=	000000F8
CHFSL_SIG_NAME	=	00000004			OPS-CLRD	=	0000007C
CNTRL_LENGTH	=	00000010			OPS-CLRF	=	000000D4
CNTRL_TABLE		00000010	R	02	OPS-CLRG	=	0000007C
CR	=	0000000D			OPS-CLRH	=	00007CFD
CVTASC		0000014B	R	02	OPS-CMPD	=	00000071
CVTNUM		000001CC	R	02	OPS-CMPF	=	00000051
CVT_BIN_TO_ASC		0000025D	R	02	OPS-CMPG	=	000051FD
DATA_TYPES		00000024	R	02	OPS-CMPH	=	000071FD
DECR_ARGPTR		000002AE	R	02	OPS-CMPP3	=	00000035
DONE		000000FA	R	02	OPS-CMPP4	=	00000037
ENDFIELD		000002EF	R	02	OPS-CRC	=	0000000B
EXCL	=	00000021			OPS-CVTBD	=	0000006C
EXESFAO		0000003F	RG	02	OPS-CVTBF	=	0000004C
EXESFAOL		0000004C	RG	02	OPS-CVTBG	=	00004CFD
EXESSIGTORET		*****W	GX	02	OPS-CVTBH	=	00006CFD
FAO		00000057	R	02	OPS-CVTDB	=	00000068
FAO_CASE		000000CF	R	02	OPS-CVTDF	=	00000076
FAO_EXIT		000000FD	R	02	OPS-CVTDH	=	000032FD
FF	=	0000000C			OPS-CVTDL	=	0000006A
FIELDEND	=	FFFFFFFFC			OPS-CVTDW	=	00000069
FIELDS		0000002C	R	02	OPS-CVTFB	=	00000048
FIRSTARG	=	00000010			OPS-CVTFD	=	00000056
GETCHAR		00000109	R	02	OPS-CVTFG	=	000099FD
GETCOUNT		00000116	R	02	OPS-CVTFH	=	000098FD
HANDLER		000005AD	R	02	OPS-CVTFI	=	0000004A
ID_REG_MASK	=	000001FC			OPS-CVTFW	=	00000049
ILLEGAC		000000EC	R	02	OPS-CVTGB	=	000048FD
ILLFIELD		000002E1	R	02	OPS-CVTGF	=	000033FD
INCR_ARGPTR		000002AB	R	02	OPS-CVTGH	=	000056FD
INDSC	=	00000004			OPS-CVTGL	=	00004AFD
INLEN	=	FFFFFFFF0			OPS-CVTGW	=	000049FD
INPTR	=	FFFFFFF4			OPS-CVTHB	=	000068FD
INSERT_CHAR		000002B9	R	02	OPS-CVTHD	=	0000F7FD
INSERT_IT		000002BF	R	02	OPS-CVTHF	=	0000F6FD
INSERT_OVF		000002BC	R	02	OPS-CVTHG	=	000076FD
LASTVAL	=	FFFFFFFF8			OPS-CVTHL	=	00006AFD
LF	=	0000000A			OPS-CVTHW	=	000069FD
MAIN_SCAN		00000064	R	02	OPS-CVTLD	=	0000006E
NEWLINE		000002B1	R	02	OPS-CVTLF	=	0000004E
OCT_HEX_DIGITS		00000033	R	02	OPS-CVTLG	=	00004EFD
ONE_CHAR_INDEX	=	00000008			OPS-CVTLH	=	00006EFD
ONE_CHAR_CNTRLS		00000018	R	02	OPS-CVTLP	=	000000F9
OPS-ACBD	=	0000006F			OPS-CVTPL	=	00000036
OPS-ACBF	=	0000004F			OPS-CVTPS	=	00000008
OPS-ACBG	=	00004FFD			OPS-CVTPT	=	00000024
OPS-ACBH	=	00006FFD			OPS-CVTRDL	=	0000006B
OPS-ADDD2	=	00000060			OPS-CVTRFL	=	0000004B
OPS-ADDD3	=	00000061			OPS-CVTRGL	=	00004BFD
OPS-ADDF2	=	00000040			OPS-CVTRHL	=	00006BFD

OP\$ _CVTSP = 00000009
OP\$ _CVTTP = 00000026
OP\$ _CVTWD = 0000006D
OP\$ _CVTWF = 0000004D
OP\$ _CVTWG = 00004DFD
OP\$ _CVTWH = 00006DFD
OP\$ _DIVD2 = 00000066
OP\$ _DIVD3 = 00000067
OP\$ _DIVF2 = 00000046
OP\$ _DIVF3 = 00000047
OP\$ _DIVG2 = 000046FD
OP\$ _DIVG3 = 000047FD
OP\$ _DIVH2 = 000066FD
OP\$ _DIVH3 = 000067FD
OP\$ _DIVP = 00000027
OP\$ _EDITPC = 00000038
OP\$ _EMODD = 00000074
OP\$ _EMODF = 00000054
OP\$ _EMODG = 000054FD
OP\$ _EMODH = 000074FD
OP\$ _MATCHC = 00000039
OP\$ _MNEGD = 00000072
OP\$ _MNEGF = 00000052
OP\$ _MNEGG = 000052FD
OP\$ _MNEGH = 000072FD
OP\$ _MOVD = 00000070
OP\$ _MOVF = 00000050
OP\$ _MOVG = 000050FD
OP\$ _MOVH = 000070FD
OP\$ _MOVH' = 00000034
OP\$ _MOV1C = 0000002E
OP\$ _MOVTUC = 0000002F
OP\$ _MULD2 = 00000064
OP\$ _MULD3 = 00000065
OP\$ _MULF2 = 00000044
OP\$ _MULF3 = 00000045
OP\$ _MULG2 = 000044FD
OP\$ _MULG3 = 000045FD
OP\$ _MULH2 = 000064FD
OP\$ _MULH3 = 000065FD
OP\$ _MULP = 00000025
OP\$ _POLYD = 00000075
OP\$ _POLYF = 00000055
OP\$ _POLYG = 000055FD
OP\$ _POLYH = 000075FD
OP\$ _SCANC = 0000002A
OP\$ _SKPC = 0000003B
OP\$ _SPANC = 0000002B
OP\$ _SUBD2 = 00000062
OP\$ _SUBD3 = 00000063
OP\$ _SUBF2 = 00000042
OP\$ _SUBF3 = 00000043
OP\$ _SUBG2 = 000042FD
OP\$ _SUBG3 = 000043FD
OP\$ _SUBH2 = 000062FD
OP\$ _SUBH3 = 000063FD
OP\$ _SUBP4 = 00000022

OP\$ _SUBP6 = 00000023
OP\$ _TSTD = 00000073
OP\$ _TSTF = 00000053
OP\$ _TSTG = 000053FD
OP\$ _TSTH = 000073FD
OUTDSC = 0000000C
OUTLEN = 00000008
OVERFLOW = 000000F1 R 02
PARSE DIRECTIVE = 0000008F R R 02
PERCENT = 00000301 R R 02
PERCENT_STR = 00000027 R R 02
RADIX = 0000003A R R 02
REPEATIT = 000002C6 R R 02
REPLACEMENT = 0000002F R R 02
REPLACE CHRS = 0000001C R 02
REPL_OFFSET = 0000000C
SF\$ _SAVE AP = 00000008
SS\$ _BADPARAM = 00000014
SS\$ _BUFFEROVF = 000000601
SS\$ _NORMAL = 00000001
SS\$ _RESIGNAL = 000000918
SS\$ _UNWIND = 000000920
STARTFIELD = 000002DD R 02
STARTOK = 000002E4 R R 02
STRING TYPES = 00000020 R 02
SYSSASCTIM = ***** GX 02
SYSSIDTOASC = ***** GX 02
SYSSUNWIND = ***** X 02
TAB = 00000009
TWO CHAR_CNTRLS = 00000010 R 02
UIC\$K_ID_FORMAT = 00000002
UIC\$K_UIC_FORMAT = 00000000
UIC\$K_WILD_GROUP = 00003FFF
UIC\$K_WILD_MEMBER = 0000FFFF
UIC\$S_FORMAT = 00000002
UIC\$S_GROUP = 0000000E
UIC\$S_MEMBER = 00000010
UIC\$V_FORMAT = 0000001E
UIC\$V_GROUP = 00000010
UIC\$V_MEMBER = 00000000

! Psect synopsis !

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
YF\$SYSFAO	000005EB (1515.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.09	00:00:00.95
Command processing	140	00:00:00.78	00:00:08.10
Pass 1	497	00:00:17.29	00:00:45.32
Symbol table sort	0	00:00:01.48	00:00:06.27
Pass 2	234	00:00:05.88	00:00:16.94
Symbol table output	25	00:00:00.19	00:00:01.16
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	935	00:00:25.74	00:01:18.76

The working set limit was 2100 pages.
81515 bytes (160 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 892 non-local and 90 local symbols.
4147 source lines were read in Pass 1, producing 16 object records in Pass 2.
143 pages of virtual memory were used to define 142 macros.

! Macro library statistics !

Macro library name	Macros defined
-\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	1
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	12
TOTALS (all libraries)	13

946 GETS were required to define 13 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:SYSFAO/OBJ=OBJ\$:SYSFAO MASD\$: [EMULAT.SRC]MISSING/UPDATE=(MASD\$: [EMULAT.ENH]MISSING)+MASD\$: [SYS.SRC]SYSFAO/UPDATE=(MAS

0384 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

The image displays a dense grid of terminal windows, each showing a different system utility or data output. The windows are arranged in 15 columns and 12 rows. Several windows are highlighted with larger text labels:

- SYSGETJPI LIS**: Located in the top right corner.
- SYSERAPAT LIS**: Located in the second row, fourth column.
- SYSFAD LIS**: Located in the third row, seventh column.
- SYSGETDVI LIS**: Located in the sixth row, tenth column.
- SYSEXIT LIS**: Located in the eighth row, fourth column.
- SYSEUTSRU LIS**: Located in the tenth row, fourth column.
- SYSFORCEX LIS**: Located in the tenth row, tenth column.

The other windows in the grid contain various types of data, including lists of files, system status information, and command-line outputs.