HEWLETT
PACKARD

# Software Internal Design Specification Volume II

## For the HP-71

Hewlett-Packard -- Portable Computer Division

Corvallis, Oregon

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                    %
%           HP-71 Software           %
%                                    %
%  Internal Design Specification     %
%                                    %
%                                    %
%           VOLUME   II              %
%                                    %
%   Entry Point and Poll Interfaces  %
%                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
   %%%%%%%%          %%%%%%%%%%%                %%%%
   %%%%%%%%          %%%%%%%%%%%%             %%%%%%%%
      %%               %%      %%%          %%%      %%%
      %%               %%       %%          %%
      %%               %%       %%          %%%
      %%               %%       %%            %%%%%%
      %%               %%       %%              %%%%%%
      %%               %%       %%                   %%%
      %%               %%       %%                    %%
      %%               %%      %%%          %%%      %%%
   %%%%%%%%          %%%%%%%%%%%%             %%%%%%%%
   %%%%%%%%          %%%%%%%%%%%                %%%%

%%     %%    %%%%      %%                   %%%%%%   %%%%%%
 %%   %%    %%   %%    %%                     %%       %%
  %% %%     %    %     %%                     %%       %%
   %%%      %%   %%    %%        %%%          %%       %%
    %        %%%%      %%%%%%    %%%        %%%%%%   %%%%%%
```

December 1983

HP Part No. 00071-90069

****** NOTICE ******

# Table of Contents


1 INTRODUCTION
- 1.1 Entry Point Interface Explanation . . . . . . . . . . . 1-2
- 1.2 Poll Interface Explanation . . . . . . . . . . . . . . . 1-4
- 1.3 Supported Entry Points . . . . . . . . . . . . . . . . . 1-7
- 1.4 Supported Non-Entry Point Symbols . . . . . . . . . . 1-22

2 ADDCAL - Address Calculation Utilities
- 2.1 XMTADR - Get XWORD Main Table Address . . . . . . . . 2-1
- 2.2 MTADDR - Calc Main Table Address for Token . . . . . 2-2
- 2.3 EXCADR - Compute Exec Addr of Token . . . . . . . . . 2-3

3 BUFUTL - System Buffer Utilities
- 3.1 IOFSCR - I/O Find for Available Scratch Buffer . . 3-1
- 3.2 I/OFND - I/O Buffer Find . . . . . . . . . . . . . . 3-2
- 3.3 I/ORES - I/O Buffer Restore . . . . . . . . . . . . . 3-3
- 3.4 I/OCON - I/O Buffer Contract From Buffer End . . . 3-3
- 3.5 I/OCOL - I/O Buffer Collapse . . . . . . . . . . . . 3-5
- 3.6 I/OALL - I/O Buffer Allocate . . . . . . . . . . . . 3-6
- 3.7 I/OEXP - I/O Buffer Expand . . . . . . . . . . . . . 3-7
- 3.8 I/ODAL - I/O Buffer Deallocate . . . . . . . . . . . 3-8
- 3.9 LXFND - Set D1 to LEX Table I/O Buffer . . . . . . . 3-9

4 CONFIG - System Configuration Utiltiies
- 4.1 ISRAM? - Pointing At RAM? . . . . . . . . . . . . . . 4-1
- 4.2 CONF - Configure Everything . . . . . . . . . . . . . 4-2
- 4.3 CNFFND - Configuration Buffer Find . . . . . . . . 4-13
- 4.4 LEXBUF - Set Up LEX Files Buffer . . . . . . . . . 4-14
- 4.5 KYDN? - Is a Key Down in Current Row? . . . . . . 4-16

5 CONVRT - Conversion Utilities
- 5.1 FLTDH - Convert 12-digit Flt To Hex Integer . . . 5-1
- 5.2 DECHEX - Convert DEC Integer To HEX Integer . . . . 5-2
- 5.3 HDFLT - Convert HEX Integer To DEC Flt-pt . . . . . 5-3
- 5.4 FLOAT - Convert Dec Integer Into 12-Dig Float . . 5-4
- 5.5 HEXASC - Convert Hexadecimal to Ascii . . . . . . . 5-5
- 5.6 CNV2UC - Converts 8 chars to uppercase . . . . . . 5-5
- 5.7 CONVUC - Convert To Upper Case . . . . . . . . . . . 5-7
- 5.8 RJUST - Unfloat A Floating-Point Number . . . . . . 5-8
- 5.9 HXDCW - Hex To Decimal Conversion . . . . . . . . . 5-9
- 5.10 DCHXW - Full Word Decimal To Hex Conversion . . 5-10
- 5.11 VARNBR - Pop and Test Variable Number . . . . . 5-11
- 5.12 STR$SB - Convert Number to String . . . . . . . 5-12

6 DSPUTL - Display Utilities
- 6.1 NOSCRL - Request No-display-scrolling . . . . . . . 6-1
- 6.2 VIEWD1 - View A Buffer While Keys Down . . . . . . 6-2
- 6.3 CURSFR - Move Cursor To Far Right . . . . . . . . . 6-3

6.4 CURSFL - Move Cursor To Far Left . . . . . . . . . . 6-3
6.5 SETFMT - Set Display Format . . . . . . . . . . . . . 6-4
6.6 UPDANN - Update Annunciator . . . . . . . . . . . . . 6-5
6.7 ASCII - ASCII Bit Pattern Tables . . . . . . . . . . 6-6
6.8 CMDPR" - Text for command stack prompt . . . . . . . 6-7
6.9 MAKEBF - Make ASCII Buffer from Display Buffer . . 6-7
6.10 BLDDSP - Build Display Pattern from Buffer . . . . 6-8
6.11 BLDBIT - Build Bit Patterns in Display . . . . . . 6-9
6.12 DSPUPD - Display Update . . . . . . . . . . . . . 6-11
6.13 GETMSK - Get Mask for Character Protection Bitmap 6-12
6.14 AVM2DS - Buffer to Display . . . . . . . . . . . 6-13
6.15 DSPCHA - Display Character . . . . . . . . . . . 6-14
6.16 DSPCL? - Clear display buffer if necessary . . . 6-16
6.17 CRLFOF - Send cursor off/CR/LF to disp w/o delay 6-17
6.18 SCNRT - Point Cursor Past Unprotected Field . . 6-18
6.19 ESCSEQ - Send Escape Sequence to Display . . . . 6-19
6.20 DSPRST - Display reset . . . . . . . . . . . . . 6-20
6.21 LCDINI - Initialize LCD display . . . . . . . . 6-20
6.22 SENDWD - Send Out Width-Sized Chunks to Device . 6-21
6.23 DSP$00 - Create String of Readable Characters . 6-22
6.24 FINLIN - Finish line in display/video . . . . . 6-23
6.25 PRPSND - Prepare to send buffer to display . . . 6-25
6.26 LSTLEN - Calculate #chars to list in display buf 6-26
6.27 DONNA - Re-prompt input line . . . . . . . . . . 6-27
6.28 CURSRT - Count cursor-rights . . . . . . . . . . 6-28
6.29 AVS2DS - AvMemSt to display . . . . . . . . . . 6-29
6.30 DSPCNA - Display by count . . . . . . . . . . . 6-30
6.31 DSPBUF - Send a buffer of chars to display . . . 6-31

7 DCMUTL - Decompile Utilities
7.1 CURSRU - Cursor Up . . . . . . . . . . . . . . . . . 7-1
7.2 EXPRDC - Expression Decompile . . . . . . . . . . . . 7-4
7.3 HXDASC - Hex to decimal ASCII conversion . . . . . . 7-8
7.4 ARITH - Get Text For An Arithmetic Operator . . . . 7-8
7.5 LDCOMP - Line Decompile Driver . . . . . . . . . . . 7-9
7.6 GTEXT - Get Text for Keyword/Function . . . . . 7-11
7.7 LIN#DC - Line number decompile . . . . . . . . . 7-13
7.8 ASCICK - Ascii Stream Decompiler . . . . . . . . 7-15
7.9 ARYDC - Array Decompile . . . . . . . . . . . . 7-16
7.10 GTEXT+ - GTEXT Preprocessor . . . . . . . . . . 7-17
7.11 OUTELA - Output End of Stmt Terminator From A . 7-18
7.12 VARDC - Variable Decompile . . . . . . . . . . 7-20
7.13 LABLDC - Lable Decompile . . . . . . . . . . . . 7-21
7.14 FILDC* - File Decompile . . . . . . . . . . . . 7-22
7.15 SKIPDC - Skip Rest of Statement Decompile . . . 7-23
7.16 LISTDC - Decompiles LIST, RENUMBER, SECURE, MERGE 7-24
7.17 PRT#DC - Port# Decompile . . . . . . . . . . . . 7-25
7.18 FTYPDC - File Type Decompile . . . . . . . . . . 7-26

8 EXCUTL - Execute Utilities
8.1 SVTRC - Save Trace Infomation In Stmt Scratch . . 8-1

8.2 EXPEXC - Evaluate Expression . . . . . . . . . . . . 8-2
8.3 FNRTN1 - Function Return . . . . . . . . . . . . . . 8-3
8.4 OUTRES - Round And Return Result . . . . . . . . . . 8-4
8.5 LIMITS - Compute Dimension Limits In Decl Stmt . . 8-5
8.6 HASH1 - Indexed Jump Through A GOTO Table . . . . 8-6
8.7 TRSFMu - Transform Utility Routine . . . . . . . . . 8-7
8.8 COPYu - COPY Utility . . . . . . . . . . . . . . . 8-12
8.9 CK"ON" - Check ON / ATTN Key . . . . . . . . . . . 8-16
8.10 FINDLB - Find Label in Current Program . . . . . . 8-17
8.11 LBLNAM - Get Label Name into Register A . . . 8-18
8.12 PRSCOP - Compute Program Scope . . . . . . . . . . 8-19
8.13 CkLoop - IMAGE parse loop to check for edit chars 8-20
8.14 BOPNM- - Process uOPNM- token during backup . . 8-22
8.15 IMinit - Initiate IMAGE output field . . . . . . 8-24
8.16 BldIMG - Put tokens from C into BldIMG stream . 8-26
8.17 IMoffs - Store offset from D1 in BldIMG stream . 8-28
8.18 PRSscn - IMAGE parse scan . . . . . . . . . . . 8-29
8.19 IMxq27 - Return to IMAGE token executor . . . 8-30
8.20 USst03 - Output characters from address in C . . 8-31
8.21 USGch+ - Display character during USING execution 8-32
8.22 USGrst - Suspend USING execution, restart parse 8-33
8.23 USGnum - Evaluate and execute numeric IMAGE field 8-34
8.24 ENDIMG - Process end of IMAGE string . . . . . . 8-36
8.25 GetEXP - Expression execute for IMAGE output list 8-37
8.26 TstEnd - Test IMAGE output list for end of list 8-39
8.27 USloop - Loop on IMAGE multiplier . . . . . . . 8-40
8.28 DCRMNT - Decrement multiplier in IMAGE string . 8-42
8.29 NXTEXP - Store pointers, execute next expression 8-43
8.30 COUNTC - Count output characters in IMAGE field 8-44
8.31 MGOSUB - Execute A GOSUB From Movable Code . . . 8-45
8.32 STRHDR - String Header . . . . . . . . . . . . . 8-47
8.33 SENDEL - Send Endline to Device via Handler . . 8-48
8.34 SENDIT - Send Buffer to Device via Handle . . . 8-49
8.35 DPART2 - IO Handler For Built-In Display . . . . 8-50
8.36 DPART3 - Finish up DISP line . . . . . . . . . . 8-51
8.37 PUTRES - Put Numeric Result Into RES . . . . . . 8-52
8.38 CKINFO - Check Handler Information . . . . . . . 8-53
8.39 EXCPAR - Execution Time Expression Parse . . . . 8-55
8.40 REPROM - Reprompt for input . . . . . . . . . . 8-56
8.41 INPOFF - Restart statement after DSLEEP . . . . 8-56
8.42 VAL00 - Parse and Execute a String on Stack . . 8-57
8.43 CHKEOL - Check if at End of Statement . . . . . 8-59
8.44 NXTVAR - Get next Variable from READ list . . . 8-59
8.45 STKVCT - Process Array Dope Vector . . . . . . . 8-60
8.46 NXTADR - Get Address of Next Array Elemen . . . 8-61
8.47 NXTELM - Get Next Array Element . . . . . . . . 8-62
8.48 STRHED - Generate String Head on Stack . . . . . 8-63
8.49 GETCHN - Get Channel Number . . . . . . . . . . 8-64
8.50 D1MSTK - Set D1 at MTHSTK (AVMEME) . . . . . . . 8-64
8.51 D1FSTK - Set D1 to FORSTK . . . . . . . . . . . 8-65
8.52 TRFROM - Trace Line Number . . . . . . . . . . . 8-65

8.53 TRTO - Generate Trace Message . . . . . . . . . 8-66
8.54 LINSKP - Line Skip . . . . . . . . . . . . . . . 8-67
8.55 NXTSTM - Scan to Next Stmt/Jump to BASIC 'oop . 8-68
8.56 TKSCN+ - Token Scan . . . . . . . . . . . . . 8-69
8.57 EOLSCN - tEOL Scan . . . . . . . . . . . . . . . 8-70
8.58 KEYFND - Key Assignment Find . . . . . . . . . . 8-71
8.59 KEYDEL - Key Assignment Delete . . . . . . . . . 8-72
8.60 GTKYCD - Get Keycode . . . . . . . . . . . . . 8-73
8.61 STMBUF - Collapse statement buffer check . . . 8-74
8.62 SCOPCK - Scope check . . . . . . . . . . . . 8-75
8.63 KEYNAM - Return key name string from keyc e . . 8-76
8.64 MFER42 - Position DO to start of BASIC st t. . . 8-76
8.65 TBMSTX - Find and Build Message From Lex able . 8-78
8.66 FLDEVX - Make Device Code Explicit . . . . . . . 8-79

9 FILUTL - File Utilities
9.1 TFMDLR - Find Transform Handler . . . . . . . . . 9-1
9.2 LOCFIL - Locate File With FIB . . . . . . . . . . 9-2
9.3 PURGEF - Purge Internal or External File . . . . . 9-3
9.4 ?PRFIL - Check File Protection . . . . . . . . . . 9-4
9.5 RDBAS - Read Line From Basic File . . . . . . . . . 9-5
9.6 RDTEXT - Read Line From Text File . . . . . . . . . 9-6
9.7 READNB - Read/Write Nibs To/From File . . . . . . . 9-7
9.8 OBEDIT - Edit Output Buffer . . . . . . . . . . . . 9-8
9.9 RPLSBH - Replace Memory File Subheader . . . . . . 9-9
9.10 SWPBYT - Swap Bytes . . . . . . . . . . . . . . 9-10
9.11 CREATF - Create File in MAIN . . . . . . . . . . 9-11
9.12 WFTMDT - Write Flags, Time, Date to File Header 9-13
9.13 PEDIT - Program Edit . . . . . . . . . . . . . . 9-15
9.14 FINDL - Find Line# within a Program File . . . 9-16
9.15 NXTLIN - Scan to Next Line . . . . . . . . . . . 9-18
9.16 RDCHDR - Read Current File header, File le gth . 9-19
9.17 GETSTC - Get Start/EOF Curr File/check Fi. type 9-20
9.18 BASCHK - Verify File Type in R2 s BASIC . . . . 9-22
9.19 FCHLBL - Find Label in Current b SIC File . . 9-23
9.20 COMPL# - Compute Line # with DO ' line ler .h . 9-25
9.21 PFINDL - Find Line# Within Progr m . . . . . . . 9-26
9.22 NULLP - Null Program Check . . . . . . . . . 9-28
9.23 CHAIN+ - Chain Subprograms, Labe s, DEF F . . 9-29
9.24 FILCRD - Copy File To Card . . . . . . . . . . 9-30
9.25 CRDFIL - Copy Card Into RAM . . . . . . . . . 9-35
9.26 WSTRFX - Write a String to a DATA File . . . . 9-38
9.27 WRTSTR - Write a string to an open TEXT file . . 9-39
9.28 WRTNUM - Write a Number to DATA or SDATA file. . 9-40
9.29 RDLNAS - Read String Length from a TEXT F. e. . 9-41
9.30 RDBYTA - Read Byte From an Opened File In A . 9-42
9.31 WRBYTC - Write Byte to an Opened File Fro . . 9-43
9.32 BACK1B - Back up the File Pointe by 1 By . . 9-44
9.33 UPCPOS - Update FIB Current Posi ion . . . . . . 9-45
9.34 GTPTRS - Get File Pointers from IB . . . . . . 9-45
9.35 FTYPF# - Look Up File Type Given Type Number . . 9-46

9.36 FTBSCH - Search a File Type Table by Type Number 9-47
9.37 FASCFD - Look Up File Type Given Type Name . . . 9-48
9.38 REWIND - Rewind Open File . . . . . . . . . . . 9-49
9.39 FIBADR - Find FIB entry address for a channel . 9-49
9.40 CRFSUB - Create a File in Mainframe . . . . . . 9-50
9.41 CRTF - Create File in MAIN, PORT, or HPIL . . 9-51
9.42 OPENF - Open File . . . . . . . . . . . . . . . 9-53
9.43 WRTFIB - Write File Information to FIB . . . . . 9-54
9.44 CLOSEF - Close File . . . . . . . . . . . . . . 9-56
9.45 CLOSEA - Close All Open Files . . . . . . . . . 9-56
9.46 FIBON - Reset Devices, Buffers at Power On/Off 9-57
9.47 PUGFIB - Purge the FIB Entries of Purged Files . 9-57
9.48 RENSUB - Renumber Subroutine . . . . . . . . . . 9-58
9.49 EXPSKP - Skip Over Tokenized Expression . . . . 9-59
9.50 FNDFCN - Find User-Defined Function . . . . . . 9-60
9.51 KEYMRG - Key Merge . . . . . . . . . . . . . . . 9-60
9.52 FILXQ^ - Filename Execute . . . . . . . . . . . 9-61
9.53 PDEV - Evaluate Num Expression as Port Device 9-64
9.54 FSPECx - File Specification Execute . . . . . . 9-65
9.55 FINDF - Find a file . . . . . . . . . . . . . . 9-67
9.56 PRGFMF - Purge File in Memory . . . . . . . . . 9-70
9.57 EDIT - Moves EDIT Pointers to Specified File . 9-71
9.58 RAMROM - Classify Memory Device . . . . . . . . 9-73
9.59 LOCADR - Locate, Classify Address's Memory Device 9-74
9.60 GETPRO - Get File Protection of Current File . . 9-75
9.61 FILSKP - File Skip . . . . . . . . . . . . . . . 9-76
9.62 FILFIL - Fill in Missing File Name . . . . . . . 9-77
9.63 FLADDR - Find First/Last Address of Mem Device . 9-79
9.64 RPLLIN - Replace Line in Memory File . . . . . . 9-80

10 FNEXEC - Function Execute
10.1 TRMNTR - Process Terminator In Expr Execute . . 10-1
10.2 GDISP$ - GDISP$ function execution . . . . . . . 10-2
10.3 KEY$ - KEY$ function . . . . . . . . . . . . . 10-3
10.4 CAT$ - CATalog Function . . . . . . . . . . . 10-3

11 GENUTL - General Purpose Utilities
11.1 STKCMD - Pushes Statement On Command STACK . . . 11-1
11.2 D=WORD - Read 8 Bytes And Convert To Uppercase . 11-2
11.3 RANGE - Verify A Byte Is In Certain Range . . . 11-2
11.4 MEMBER - Check If Byte Is A Member Of A Set . . 11-3
11.5 STUFF - Fill Memory With Stuff Or 0's . . . . . 11-4
11.6 MOVEDM - Blk Move To Higher Addr . . . . . . . . 11-5
11.7 MOVEUM - Blk Move To Lower Addr . . . . . . . . 11-6
11.8 STRTST - Test Strings For Equality . . . . . . . 11-8
11.9 CSRC1 - Perform 1 CSRC . . . . . . . . . . . . 11-9
11.10 OUT1TK - Output 1 byte from A(B) . . . . . . . 11-10
11.11 D1C=R3 - Restore C(A),D1 from R3 . . . . . . . 11-12
11.12 R3=D10 - Save D0 and D1 in R3 . . . . . . . . . 11-12
11.13 CSL9RO - Copy D1 to R0(9-5) . . . . . . . . . . 11-14
11.14 IMDO+2 - Add 2 to R1(A), copy value to D0 . . . 11-15

11.15 D12R0A - Copy D1 to R0(A) . . . . . . . . . . . . 11-16
11.16 NwOFFS - Recover old offset, store new one in R 11-17
11.17 RCVOFS - Recover offset from RAM storage . . . 11-18
11.18 BP - Machine-level Beep . . . . . . . . . . 11-19
11.19 CHIRP - Do An Annoying Little Beep . . . . . . 11-20
11.20 ROMCHK - Find ROM / File Chain Start . . . . . 11-21
11.21 ASRW3 - Shift A Right 3 Nibbles . . . . . . . 11-23
11.22 SFLAGS - Sets system flag . . . . . . . . . . . 11-24
11.23 SFLAGC - Clears system flag . . . . . . . . . . 11-25
11.24 SFLAGT - Toggles system flag . . . . . . . . . 11-26
11.25 SFLAG? - Tests system flag . . . . . . . . . . 11-27
11.26 GTFLAG - Gets RAM nib and flag mask . . . . . . 11-28
11.27 FINDA - Look For A(B) In A Table And Jump . . 11-29
11.28 TBLJMP - Indexed table jump . . . . . . . . . . 11-30
11.29 INTRPT - Interrupt Handler . . . . . . . . . . 11-31
11.30 ATNCLR - Clear Attention Flags . . . . . . . . 11-33
11.31 DSLEEP - Deep sleep . . . . . . . . . . . . . . 11-33
11.32 SLEEP - Scan KB, do LSLEEP if key buffer empty 11-36
11.33 CKSREQ - Handle service requests . . . . . . . 11-37
11.34 QUOTCK - Quote and Apostrophe Check . . . . . . 11-38
11.35 MFLG=0 - Clear MLFFLG nibble . . . . . . . . . 11-39
11.36 PSHSTK - Push Stack . . . . . . . . . . . . . . 11-40
11.37 PSHGSB - Push address on GOSUB Stk . . . . . . 11-41
11.38 POPSTK - Pop Stack . . . . . . . . . . . . . . 11-43
11.39 RELJMP - Relative Jump From (D1) . . . . . . . 11-44
11.40 EOLXCK - End of Stmt check . . . . . . . . . . 11-45
11.41 OUTNBS - Output nibbles . . . . . . . . . . . . 11-46
11.42 MFWRN - Warning/message driver . . . . . . . . 11-47
11.43 MFERR* - Error message driver . . . . . . . . . 11-51
11.44 MFERsp - Error Message With Text Insertion . . 11-54
11.45 DOASCI - Send ASCII bytes to DATO . . . . . . . 11-56
11.46 MOVE*M - Move Memory Up or Down Without Ref Adj 11-59
11.47 MVMEM - Move File Memory W/Ref Adjust . . . . 11-60

12 KEYUTL - Keyboard Utilities
12.1 CHEDIT - Character Editor . . . . . . . . . . . 12-1
12.2 KEYRD - Read A Key . . . . . . . . . . . . . . 12-2
12.3 -LINE - Delete Through End Of Line . . . . . . 12-5
12.4 RPTKY - Check For Repeating Keys . . . . . . . 12-6
12.5 CMD1ST - Set command stack pointer to 1st cmd . 12-7
12.6 CMDS00 - Display Cmd Stack Entry . . . . . . . . 12-7
12.7 CMDFND - Find Nth Command Stack Entry . . . . . 12-8
12.8 CMDINI - Recalls CMDPTR and MAXCMD . . . . . . . 12-9
12.9 SCRLLR - Scroll Left and Right . . . . . . . . . 12-10
12.10 FGTBL - State table for F & G shifted keys . . 12-11
12.11 KEYCOD - Keycode Map . . . . . . . . . . . . . 12-12
12.12 DEBNCE - Debounce and scan keyboard . . . . . . 12-13
12.13 POPBUF - Pop Key Buffer . . . . . . . . . . . . 12-14

13 MATH - System Math Functions
13.1 ADDONE - Add One . . . . . . . . . . . . . . . . 13-1

13.2 1/X15 - 1/X . . . . . . . . . . . . . . . . . . . . . . 13-2
13.3 AD2-15 - Add two 15 digit forms . . . . . . . . . . 13-2
13.4 MP2-15 - Multiply . . . . . . . . . . . . . . . . . . . 13-3
13.5 DV2-15 - Divide . . . . . . . . . . . . . . . . . . . . 13-4
13.6 SQR15 - Square Root . . . . . . . . . . . . . . . . . 13-5
13.7 INVNaN - Create IVL NaN . . . . . . . . . . . . . . 13-6
13.8 LN1+15 - LN(1+X) . . . . . . . . . . . . . . . . . . . 13-7
13.9 LN15 - Natural Logarithm . . . . . . . . . . . . . . 13-8
13.10 EXP15 - EXP(x) (exponential fcn) . . . . . . 13-9
13.11 LGT15 - Log base 10 . . . . . . . . . . . . . . . 13-10
13.12 YX2-15 - Y to the X power . . . . . . . . . . . . 13-10
13.13 FAC15S - Internal Factorial . . . . . . . . . . . 13-11
13.14 uTEST - Perform comparisons . . . . . . . . . . 13-12
13.15 EX12 - Return exponent of 12-dig arg . . . . 13-13
13.16 SQRSAV - SQR for Chain calculations. . . . . . 13-14
13.17 SAVGSB - Put SB into sINX . . . . . . . . . . . 13-15
13.18 ARG12 - Return Arg of X+iY (12-dig args) . . . 13-16
13.19 SIN12 - Trig: Sine of 12-dig arg . . . . . . . 13-17
13.20 TRC90 - Table of numeric constants . . . . . . 13-18
13.21 ASIN12 - ArcSin Inv Trig (12-dig argument) . . 13-19

14 MTHSTK - Math Stack Utilities
14.1 POP2N - Pop 2 Numbers From Stack. . . . . . . . 14-1
14.2 POP1N - Pop 1 Number Off Of Stack . . . . . . . 14-2
14.3 REVPOP - REV$ On String And Then POP1S . . . . . 14-3
14.4 POP1S - Pop 1 String Arg Off Stack . . . . . . 14-3
14.5 MPOP2N - Pop 2 Args W/signan Check . . . . . . . 14-4
14.6 MPOP1N - Pop 1 Arg & Check For Sig NaN . . . . . 14-5
14.7 REV$ - Reverse Characters In A String On Stack 14-6
14.8 POPMTH - Skip Past An Item On Mthstk . . . . . . 14-7
14.9 ARGPR+ - Reads modes, pops and norm. real nbr . 14-8
14.10 ARGPRP - Pops and normalizes real number . . . 14-9
14.11 POP1R - Pops real number from math stack . . . 14-10
14.12 ARGSTA - Pops and tests real number . . . . . . 14-11
14.13 XXHEAD - Remove String Header (Undo ADHEAD) . . 14-12
14.14 ADHEAD - Add String Header . . . . . . . . . . . 14-13
14.15 BF2STK - Buffer To Stack . . . . . . . . . . . . 14-14
14.16 COLLAP - Collapse Math Stack . . . . . . . . . . 14-15
14.17 ERRM$f - Transfer ASCII from AvMem to stack . . 14-15

15 MTHUTL - System Level Math Utilities
15.1 REDUCE - Parse And Execute Partial ExpresSIONS . 15-1
15.2 NRMCON - Convert BLDCON Constant into Usable Form 15-2
15.3 BLDCON - Build A Constant For CalC MODE . . . . 15-3
15.4 READIN - Read Something In . . . . . . . . . . . . 15-3
15.5 RSTST - Restore Status Bits . . . . . . . . . . . 15-4
15.6 SMALL - Create Special Consts . . . . . . . . . . 15-5
15.7 SIGCHK - Report Signaling NaN . . . . . . . . . . 15-6
15.8 RND-12 - Round A 12-digit Fp Number . . . . . . 15-7
15.9 A-MULT - Multiply Two 20-bit Hex Integers . . . 15-8
15.10 SHF10 - Shift to normalize . . . . . . . . . . . 15-8

15.11 SQR70 - Set SB according to Reg C . . . . . . . 15-9
15.12 INF*0 - Inf*0 exception . . . . . . . . . . . . 15-10
15.13 XYEX - EXCHANGE X & Y . . . . . . . . . . . . . 15-11
15.14 SPLITA - SPLIT A . . . . . . . . . . . . . . . . 15-11
15.15 CLRFRC - Clear fractional part . . . . . . . . . 15-12
15.16 IF12A - Integer/Fraction Split . . . . . . . . . 15-13
15.17 SPLTAC - Split & normalize A & C . . . . . . . . 15-14
15.18 SPLITC - SPLIT C . . . . . . . . . . . . . . . . 15-15
15.19 uRES12 - User Result . . . . . . . . . . . . . . 15-15
15.20 uRND>P - user ROUND . . . . . . . . . . . . . . 15-17
15.21 RNDNRM - Round a Normal Number . . . . . . . . . 15-18
15.22 HTRAP - HANDLE TRAPS . . . . . . . . . . . . . . 15-19
15.23 HNDLFL - HANDLE FLAG SETTING . . . . . . . . . . 15-20
15.24 MESSG - MESSAGE . . . . . . . . . . . . . . . . 15-20
15.25 FINITA - Is (A,B) non-finite ? . . . . . . . . . 15-21
15.26 FNPWDS - Weed out NaNs and Infs . . . . . . . . . 15-22
15.27 STAB1 - Store AB into scratch 1 . . . . . . . . 15-23
15.28 IDIVA - A-field Integer Divide . . . . . . . . . 15-24
15.29 IDIV - Full Word Integer Divide. . . . . . . . 15-25
15.30 MPY - HEX * HEX Or HEX * DEC Multiply. . . . 15-26
15.31 RNDAHX - Pops, tests,rounds,converts dec to hex 15-27
15.32 SB15S - 15-digit subtract/add routine . . . . 15-28
15.33 uRESD1 - Variation of uRES12 . . . . . . . . . . 15-29
15.34 GETSA - Tests current statistical array . . . 15-30
15.35 SPLTAX - Split, normalize A; handle signal NaN 15-31
15.36 STSCR - Push 15-Form Onto Math Scratch Stack . 15-32
15.37 RCSCR - Pop 15-Form From Math Scratch Stack . 15-33
15.38 RCLW1 - Recall 1st (Top) Math Scrtch Stack Ent 15-34
15.39 STKCHR - Add a Character to a Stack Item . . . 15-35
15.40 TST12A - Compare numbers: 12-Digit arg's A,C . 15-36
15.41 BIASA+ - Add Exp bias to A . . . . . . . . . . . 15-37
15.42 MSN12 - Find most significant NaN, 12-Dig arg' 15-38
15.43 CLASSA - Classification of numeric arg . . . . 15-39
15.44 GETCON - Get constants from table . . . . . . . 15-40
15.45 MAKE1 - Make 12-dig 1 in C and compare with B. 15-41
15.46 DBLSUB - Double Precision Subtract . . . . . . 15-41
15.47 DBLPI4 - Generate 31-digit PI/4 or 45 . . . . . 15-42
15.48 TWO* - Double Precision Doubler . . . . . . . 15-43
15.49 SHFLAC - Double Precision Shift Left . . . . . 15-43
15.50 SHFRBD - Double Precision Right Shift . . . . . 15-44
15.51 PI/2 - Generate PI/2 . . . . . . . . . . . . 15-44
15.52 FLIP8 - Toggle status bits . . . . . . . . . . 15-45

16 PARUTL - Parse Utilities
16.1 NTOKNL - Lex Analysis . . . . . . . . . . . . . 16-1
16.2 SCAN - Scan LEXfile Text Table For Lexeme . . 16-4
16.3 NUMSCN - Scan Number In Lexical Analysis . . . . 16-5
16.4 LINEP - Parse Main Driver after ENDLINE . . . . 16-6
16.5 LBLINP - Parse Line Number or Label . . . . . . 16-12
16.6 EOLCK - Check for EOL,@,!,ELSE . . . . . . . . 16-14
16.7 WRDSCN - Keyword Scan from Table . . . . . . . . 16-15

16.8 SYNTXe - "Syntax" Parse Error Exit . . . . . . . . 16-17
16.9 RESTAR - Restart Lex Analyzer . . . . . . . . . . 16-19
16.10 GNXTCR - Get Next Non-blank Character . . . . . 16-22
16.11 RESPTR - Restore Input Pointer . . . . . . . . . 16-23
16.12 COMCKO - Check Comma & Output Comma Token . . . 16-24
16.13 #CK - Check for # . . . . . . . . . . . . . 16-25
16.14 NXTP - NEXT statement parse . . . . . . . . . 16-26
16.15 VARP - Variable Parse . . . . . . . . . . . . 16-27
16.16 ARRYCK - Parses Doubly Dimensioned Array . . . 16-29
16.17 NUMCK - Valid Numeric Expression Check . . . . 16-30
16.18 STRGCK - Valid String Expression Check . . . . 16-31
16.19 COMCK - Comma Check . . . . . . . . . . . . . 16-32
16.20 OUTLIT - Output Delimited Literal . . . . . . . 16-33
16.21 OUTVAR - Output Parsed Variable . . . . . . . . 16-35
16.22 FSPECp - File Specification Parse . . . . . . . 16-36
16.23 FLTYPp - Parse File Type . . . . . . . . . . . 16-39
16.24 FILEP - File Name Parse . . . . . . . . . . . 16-40
16.25 CATCHR - Categorize Character . . . . . . . . . 16-42
16.26 EXPPAR - Expression Parse . . . . . . . . . . . 16-43
16.27 P1-10 - Numeric Operand Found . . . . . . . . 16-46
16.28 SE1-10 - String Operand Found . . . . . . . . . 16-46
16.29 ACCEPT - Funny function parse error reentry poi 16-47
16.30 CONCOM - Compile a Numeric Constant . . . . . . 16-48

17 POLL - Poll Interface Descriptions
17.1 pCMPLX - Complex Number Operation Poll . . . . . 17-1
17.2 pTRFMx - Poll for TRANSFORM Execution . . . . . 17-3
17.3 pTIMR# - Poll Timer# > 3 for ON/OFF TIMER . . . 17-8
17.4 pCOPYx - Poll for COPY to external device . . . 17-10
17.5 pCURSR - Cursor Key with non BASIC file Poll . . 17-13
17.6 POLL - Poll LEX Files with Process Number . . 17-15
17.7 FPOLL - Fast Poll all LEX files with Process # 17-21
17.8 pPARSE - Parse Take Over Poll . . . . . . . . . 17-23
17.9 pFSPCp - POLL for File Specifer Parse . . . . . 17-24
17.10 pDEVCp - Poll for Device Specifier Parse . . . 17-27
17.11 pRUNft - Poll on RUN with unknown filetype . . 17-30
17.12 pRUNnB - Poll before non BASIC file exec (BIN) 17-32
17.13 pBSCen - Poll entering BASIC interpreter . . . 17-35
17.14 pBSCex - Poll to Exit BASIC Interpreter . . . . 17-37
17.15 pExcpt - Poll on Exception after Stmt Execute . 17-39
17.16 pZERPG - Poll to zero program information . . . 17-42
17.17 pIMCHR - Poll for unrecognized IMAGE char . . . 17-43
17.18 pIMbck - Backward search, IMAGE parse . . . . . 17-48
17.19 pIMcpi - Initializing IMAGE field in complex . 17-51
17.20 pIMXQT - Begin IMAGE execution . . . . . . . . 17-55
17.21 pIMXCH - Unrecognized symbol in IMAGE execution 17-58
17.22 pIMcpw - Working on complex image field . . . . 17-62
17.23 pWCRD8 - Poll To Write Copycode 8 File To Card 17-64
17.24 pWCRD - Card Write Poll . . . . . . . . . . . 17-65
17.25 pRCRD - Poll After Reading Card. . . . . . . . 17-67
17.26 pCRDAB - ABORT Card Read Poll . . . . . . . . . 17-68

| | | | | |
|---|---|---|---|---|
| 17.27 | pCONFG | - | Configuration Poll . . . . . . . . . . . | 17-70 |
| 17.28 | pWTKY | - | Poll When Waiting For Key . . . . . . | 17-72 |
| 17.29 | pKYDF | - | Poll To Define Key . . . . . . . . . . | 17-74 |
| 17.30 | pCLDST | - | Coldstart poll . . . . . . . . . . . . | 17-76 |
| 17.31 | pMNLP | - | Poll on entry to main loop . . . . . . | 17-77 |
| 17.32 | pPWROF | - | Poll when powering off . . . . . . . . | 17-78 |
| 17.33 | pDSWNK | - | Poll to awake machine w/o key . . . . | 17-80 |
| 17.34 | pDSWKY | - | Poll if machine wants to wake up . . . | 17-81 |
| 17.35 | pSREQ | - | Service Request poll . . . . . . . . . | 17-83 |
| 17.36 | pVER$ | - | VER$ Statement Extension Poll . . . . | 17-86 |
| 17.37 | pPRTIS | - | PRINTER IS handler poll . . . . . . . | 17-87 |
| 17.38 | pPRTCL | - | PRINT class statement handler poll . . | 17-89 |
| 17.39 | pRDNBF | - | Write Current Sector, Read Next Sector | 17-91 |
| 17.40 | pREAD# | - | READ# on File of Copycode = 8 . . . . | 17-92 |
| 17.41 | pEOFIL | - | Poll at End-of-File . . . . . . . . . | 17-94 |
| 17.42 | pPRIN# | - | PRINT# on File of Copycode = 8 . . . . | 17-96 |
| 17.43 | pFTYPE | - | Search for file type table entry . . . | 17-97 |
| 17.44 | pFASCH | - | Search for File Type by Name . . . . . | 17-99 |
| 17.45 | pSREC# | - | Position to Rec# of File w/Copycode 8 | 17-100 |
| 17.46 | pRDCBF | - | Read Current Sector From Mass Memory | 17-102 |
| 17.47 | pWRCBF | - | Write I/O Buffer to Mass Memory Devic | 17-104 |
| 17.48 | pCREAT | - | Create File in External Device . . . | 17-105 |
| 17.49 | pCRT=8 | - | Create File w/Create Code = 8 . . . | 17-107 |
| 17.50 | pFINDF | - | Find External File . . . . . . . . . | 17-109 |
| 17.51 | pDIDST | - | Poll for Device ID Storage . . . . . | 17-111 |
| 17.52 | pDATLN | - | Compute File Len w/Create Code = 8 . | 17-112 |
| 17.53 | pREN | - | Renumber an XWORD line# reference . | 17-114 |
| 17.54 | pCALSV | - | Poll to save local environment on CAL | 17-115 |
| 17.55 | pCALRS | - | Poll to restore local environment . | 17-117 |
| 17.56 | pFNIN | - | Poll at start of multiline U.D.F. . | 17-119 |
| 17.57 | pFNOUT | - | Poll at end of multiline U.D.F. . . | 17-120 |
| 17.58 | pRTNTp | - | Poll on Special Return type . . . . | 17-121 |
| 17.59 | pFILXQ | - | Poll for device to return device ID | 17-124 |
| 17.60 | pFSPCx | - | File Spec Execution poll . . . . . . | 17-126 |
| 17.61 | pPURGE | - | Poll to PURGE file on external device | 17-128 |
| 17.62 | pPRGPR | - | Poll to PURGE file on non-RAM device | 17-130 |
| 17.63 | pRNAME | - | Poll to RENAME file on unknown device | 17-132 |
| 17.64 | pFPROT | - | [UN]SECURE or PRIVATE in non-RAM devi | 17-135 |
| 17.65 | pEDIT | - | Poll to position at non-BASIC file . | 17-137 |
| 17.66 | pFILDC | - | Polls for File Decompile . . . . . . | 17-138 |
| 17.67 | pCAT | - | Poll for CAT on external device . . | 17-140 |
| 17.68 | pCAT$ | - | Poll for CAT$ on external device . . | 17-142 |
| 17.69 | pLIST | - | Poll for LIST on an external device | 17-144 |
| 17.70 | pLIST2 | - | POLL to LIST non-BASIC/non-KEY file t | 17-146 |
| 17.71 | pMERGE | - | Polls to MERGE non-mainframe file . | 17-148 |
| 17.72 | pMRGE2 | - | Polls to MERGE non-BASIC,non-KEY file | 17-149 |
| 17.73 | pWARN | - | Warning poll . . . . . . . . . . . . | 17-151 |
| 17.74 | pERROR | - | Error poll . . . . . . . . . . . . . | 17-154 |
| 17.75 | pMEM | - | Memory error poll . . . . . . . . . | 17-157 |
| 17.76 | pENTER | - | Poll to ENTER Data From HPIL Device | 17-161 |
| 17.77 | pTEST | - | Test Poll for Timing Polls . . . . . | 17-161 |

17.78 pTRANS - Poll to Translate a Message . . . . 17-163

18 PTRUTL - Pointer Utilities
18.1 D=AVMS - Set D(A) to AVMEMS or AVMEME . . . . . . 18-1
18.2 GETAVM - Get Available memory limits . . . . . . . 18-2
18.3 D1=AVE - Set D1 to (AVMEME) . . . . . . . . . . . 18-2
18.4 AVE=D1 - Update AVMEME From D1 or C . . . . . . 18-3
18.5 D0=FIB - Set D0,C(A) to value at STMTD1 . . . . 18-4
18.6 RFAD-I - Adjust Refs when mem moves to lower addr 18-5
18.7 RFUPD+ - Updates a ptr when mem moves . . . . . 18-6
18.8 FORUPD - FOR Stack Update . . . . . . . . . . . 18-7
18.9 RFADJ+ - Adjusts Refs When Mem Moves=>Higher Addr 18-8
18.10 LDCSET - Set D=AVMEME; D0=OUTBS . . . . . . . . . 18-9
18.11 D0=AVS - Set D0=address in AVMEMS . . . . . . . . 18-10
18.12 MEMCKL - Check Avail Memory With, Without Leeway 18-11
18.13 CLCOLL - Collapse Buffer Pointers to CLCSTK . . 18-13

19 SAVSTK - Save Stack Utilities
19.1 SVINFO - Save/Read File Information . . . . . . 19-1
19.2 SALLOC - Allocate Arbitrary Save Stack Block . . 19-3

20 SAVUTL - Save Utilities
20.1 STATRS - Restore Status . . . . . . . . . . . . . 20-1
20.2 STATSV - Save Status S13, S11 - S0 . . . . . . . . 20-2
20.3 RSTK<R - Restore RSTK Level(s) From RSTKBF Buffer 20-3
20.4 R<RSTK - Save RSTK Level(s) Into RSTKBF Buffer . 20-4
20.5 SNAPRS - Restore CPU Snapshot From SNAPSV Buffer 20-5
20.6 SNAPSV - Save Snapshot of CPU in SNAPSV Buffer . 20-6
20.7 SRLEAS - Release Arbitrary Block From Save Stack 20-8

21 STDCMP - Statement Decompile
21.1 DSTRDC - Decompiles Variable Declarations . . . 21-1
21.2 PRNTDC - Expression List Decompile . . . . . . . 21-2
21.3 ONDC - ON..GOTO,..GOSUB,..RESTORE Decompile . 21-3
21.4 RENMDC - PURGE, COPY Decompile . . . . . . . . . 21-4

22 STEXEC - Statement Execute
22.1 ASNMNT - Perform Variable Assignment . . . . . . 22-1
22.2 STORE - Store From Stack To Variable . . . . . 22-2
22.3 ONERR - Execute branch of ON TIMER/ERROR . . . 22-3
22.4 END - END, END ALL, END SUB, END DEF Statement 22-5
22.5 GOTO - Statement Execution . . . . . . . . . . 22-8
22.6 USING - Interpret IMAGE String . . . . . . . . . 22-13
22.7 BEEP - BEEP Keyboard Execute . . . . . . . . . . 22-16
22.8 PRINT* - PRINT class statement execution . . . . 22-17
22.9 PART3 - Finishes up a PRINT class statement . . 22-18
22.10 ZERBUF - Looks Like a Zero Length Buffer . . . 22-19
22.11 CREATE - Statement to Create Data File . . . . 22-19
22.12 CALL - Sub-program call execution . . . . . . 22-20
22.13 CALBIN - Binary program call BASIC subprogram . 22-23
22.14 ENDSUB - ENDSUB execution . . . . . . . . . . . . 22-23

22.15 CAT - Executes CAT Command . . . . . . . . . . 22-24

23 STPARS - Statement Parse
23.1 GOTOp - GOTO Statement Parse . . . . . . . . . . 23-1
23.2 RESTRP - RESTORE Statement Parse . . . . . . . . . 23-2
23.3 BEEPP - BEEP Statement Parse . . . . . . . . . . 23-3
23.4 ONP - ON Statement Parse . . . . . . . . . . . 23-5
23.5 READP - READ, READ# Statement Parse . . . . . . 23-7
23.6 DECP - Parse of Variable Declaration Statements 23-9
23.7 PRTP - PRINT Statement Parse . . . . . . . . . . 23-10
23.8 POKEP - POKE Statement Parse . . . . . . . . . . 23-12
23.9 CALLP - CALL Statement Parse . . . . . . . . . . 23-13
23.10 ADDP - ADD Statement Parse . . . . . . . . . . 23-14

24 SYSTEM - System Level Major Entry Points
24.1 CNFLCT - Report "Data Type" Error. . . . . . . . 24-1
24.2 ARGERR - Report "Invalid Arg" Error. . . . . . . 24-1
24.3 NORDIM - Report "Var Context" Error . . . . . . 24-2
24.4 BSCEXC - BASIC Stmt/Pgm Execution: Keyboard Exec 24-3
24.5 IMerr - Report "Invalid IMAGE" error . . . . . 24-9
24.6 IVAERR - Report "Invalid Arg" error. . . . . . . 24-9
24.7 COLDST - Cold starts machine . . . . . . . . . . 24-10
24.8 MAINLP - Main Loop . . . . . . . . . . . . . . . 24-11
24.9 PWROFF - Power Off . . . . . . . . . . . . . . . 24-13
24.10 RDATTY - Report "Data Type" error . . . . . . . 24-14
24.11 CORUPT - Report "System Error" error . . . . . 24-15
24.12 MFERR - Mainframe BASIC system error . . . . . 24-16
24.13 BSERR - BASIC system error . . . . . . . . . . 24-16
24.14 MFERRS - Stop BASIC execution for error . . . . 24-17
24.15 MEMERR - Insufficient Memory error . . . . . . 24-18
24.16 MEMER* - Low-level memory error . . . . . . . . 24-19

25 TIME - Time and Date Utilities
25.1 CMPT - Return Current Time . . . . . . . . . . . 25-1
25.2 SETIME - Set And Normal Adjust Routine . . . . . 25-2
25.3 ADJA - Absolute Time Adjust Routine . . . . . . 25-3
25.4 EXACT - Compute New Accuracy Factor. . . . . . . 25-5
25.5 SETALM - Set Absolute Alarm Time . . . . . . . . 25-6
25.6 SETALR - Set Alarm Relative To Current Time . . 25-7
25.7 YMDHMS - Return Time And Date . . . . . . . . . 25-8
25.8 SETTMO - Set System Timeout . . . . . . . . . . 25-9
25.9 TODT - Time To Time-of-day And Day# . . . . . . 25-10
25.10 SECHMS - Convert Secs To Hours, Mins, Secs . . 25-11
25.11 HMSSEC - Hours, Mins, Secs To Seconds. . . . . 25-12
25.12 YMDDAY - Convert Year,month,day To Day# . . . . 25-13
25.13 DAYYMD - Day# To Year, Month, Day . . . . . . . 25-14
25.14 DAY2JD - Day# To Julian Date . . . . . . . . . 25-16

26 VARMGT - Variable Management
26.1 STRASN - String Assignment . . . . . . . . . . . 26-1
26.2 DEST - Save Variable Destination Info . . . . 26-2

26.3 BASE - Determine Option Base . . . . . . . . . . 26-3
26.4 SHRT - Store Into Short Variable . . . . . . . . 26-4
26.5 INTGR - Store Into An Integer Variable . . . . 26-5
26.6 DYNAMC - Variable Recall . . . . . . . . . . . . . 26-6
26.7 RECADR - Some Recall Utility . . . . . . . . . . . 26-8
26.8 ADRSUB - Get Variable Name From Token Stream . . 26-8
26.9 ADDRSS - Find Address Of A Variable . . . . . . 26-9
26.10 CHNHED - Point To Variable Chain Head . . . . . 26-11
26.11 DPVCTR - Creates Vars, Computes # Of Elements . 26-12
26.12 GETDIM - Get A Dimlimit From Stack . . . . . . 26-13
26.13 SPACE - Compute Space Needs For An Array . . . 26-14
26.14 PREP - Prepare To Create A Variable/array . . 26-15
26.15 DMNSN - Create And Allocate Memory For Variabl 26-16
26.16 DATLEN - Compute Data Length Given Type . . . . 26-17
26.17 ARYSIZ - Compute Array Size, # Elements . . . . 26-18
26.18 GETNAM - Get variable name . . . . . . . . . . . 26-19

| INTRODUCTION | CHAPTER 1 |
| --- | --- |

This document describes the interfaces to the supported entry points of the HP-71 Operating System and to the polls it issues. Each supported interface is described in a documentation header that is extracted directly from the source file that contains that entry point or poll. These headers are listed here according to functional category (poll interfaces are listed under the category "POLL") and all entry points and poll process symbols are indexed for ease of reference. In addition, an alphabetized list of the supported entry points and poll process numbers is given at the end of this chapter.

It is the intent of HP to preserve the supported interfaces described in this document, as well as the absolute address position of each supported entry point, through any future updates of the HP-71 operating system. In general this allows external software which uses these interfaces to work predictably without regard to the version of the HP-71 on which it is run. However, HP reserves the right to adjust the suported interfaces in any manner it chooses. Supported interfaces are identified by the "Name:(S)" line of the documentation header, as described below.

An unsupported entry point may be added to the supported list if HP deems the request to be justified. To request support for an entry point, please contact Systems Engineering Support in the HP Portable Computer Division Product Support Group at (503) 757-2000. Corrections or requested enhancements to the interface documentation are welcome and should also be reported in this manner.

**WARNING !!**

Only supported entry points are available for use by external software. HP expresses no intent to indefinitely preserve the interfaces to any unsupported entry points described in this volume or in Volume III, since it is inevitable that code in any 64K byte operating system will have to change or move occasionally to fix bugs. The interface to unsupported entry points, and their absolute address position, may therefore change at any time and without notice to outside parties.

## 1.1 Entry Point Interface Explanation

The interface to each supported entry point is described in a documentation header which is extracted directly from the source file of the system module which contains that entry point. The fields in the header have the following meanings:

Category:
This line gives the functional category of the entry point as well as the name of the operating system source file (listed in Volume III) which contains that entry point.

Name:(S) (or Name: )
Gives the entry point name followed by its one-line title. Supported entry points are preceded by "Name:(S)" and unsupported (non-stationary) entry points are preceded by "Name:". Please see the warning at the start of this chapter regarding unsupported entry points.

Purpose:
This section describes briefly the intended use of the entry point(s) documented in the header.

Entry:
This section describes the state of the machine which is expected by the entry point. The placement of required values in CPU registers or RAM locations, status settings, and so forth, are given. The mode of the machine (HEX or DEC) is also given where relevant. HEX mode should be assumed if not explicitly stated.

Exit:
This section describes the state of the machine after execution of the entry point routine. Return is to the caller of the entry point, unless explicitly stated otherwise. Note that only documented exit conditions of a routine may be depended on, even if the code currently leaves an enticing value in an undocumented location. If code must be changed to fix a bug, only the documented exit conditions will be preserved. To have an undocumented exit condition documented for use by outside software, please contact HP in the manner described at the start of this chapter.

Calls:
This section lists all routines called or jumped to by this module.

Uses:

This section lists the machine resources (registers, RAM locations, status settings) that are altered by the execution of this routine.

Usage summaries preceded by the word "Inclusive:" indicate the sum total of all the resources used by the routine, including any routines called or jumped to by this routine.

Usage summaries preceded by the word "Exclusive:" indicate only those resources altered by the routine proper, excluding consideraton of any other routines it may call or jump to. Exclusive summaries were produced early in code development in order to facilitate the compilation of inclusive summaries for higher level modules. Some headers still retain the exclusive summaries along with the inclusive summaries.

Note that in some cases the usage summary may claim that a resource (such as an entire register) is used, whereas close inspection of Volume III may disclose that in fact only part of that resource is currently used by the routine's code. Callers of such a routine must not assume that the currently unused portion of that resource will remain unused, since future HP code changes to correct a bug may require that the full resource be used.

In short, the user of a routine should never count on a resource remaining unaltered through execution of the routine if that resource appears in the usage list for the routine.

Stk lvls:

Gives the number of hardware stack levels which are used by the execution of this routine, which unless indicated otherwise is the maximum depth used by this routine considering all routines it calls or jumps to. Note that a GOSUB to a routine requires one hardware stack level in addition to the documented stack level usage of that routine.

NOTE:

Important things to know about the operation of this routine. This section is often omitted.

Detail:

Specific, detailed information about data structures or other constructs used by this routine. This section is often omitted.

Algorithm:

A high-level description of the module flow. This section is often omitted.

History:

A record of the development of this interface. This section is sometimes omitted.

## 1.2 Poll Interface Explanation

The interface to each poll issued by the mainframe is described in a documentation header which is extracted directly from the source file of the system module which issues that poll. The fields in the poll documentation header have the following meanings:

Name:
The symbolic name of the poll process number, followed by its title.

Category:
Identifies this documentation header as being a poll interface description.

Type:
Identifies whether this is a Slow Poll (POLL) or a Fast Poll (FPOLL). A Slow Poll stores and restores certain registers and stack levels as it queries each LEX file in turn. This gives potential handlers more room in which to work, but takes more time. In addition, Slow Polls allow an error exit condition which can be passed back to the code which issued the poll. A Fast Poll saves away no registers or stack levels, so it is faster but gives the potential handlers less room to work and no opportunity to report an error condition. For more information on polling, please refer to the chapter on "Language Extension and Binary Files" in Volume I of this document.

Purpose:
The reason for issuing the poll (e.g., seeking handler for copy to unknown device).

Should poll be "Handled"?:
The poll handler "handles" a poll (declares the poll to have been "handled") by returning to the operating system with the hardware XM bit set to 0. This satisfies (terminates) the polling process: the operating system stops querying LEX files in search of a handler, and returns control to the calling code (the code which issued the poll). Similarly, the operating system returns control to the calling code if no handler declares the poll "handled." The calling code is informed whether the poll was "handled."

In some cases, usually a Fast Poll or a poll which requires no specific action, no handler is allowed to declare the poll "handled." This permits all LEX files present to detect the issuing of such a poll (an example is the service request poll, which is issued when one or more hardware service requests are pending). However, most polls that require a specific action, such as copying to an unknown device, will require that the handler should declare the poll "handled."

This section of the poll documentation header indicates whether a handler should declare the poll "handled." A "Yes" answer here in the header means that the poll is to be "handled" if the needed action has been taken and the exit conditions have been met.

Meaning of "Handling" Poll:
Briefly describes what a LEX file is doing by handling this poll, and what the calling code will do if the poll is handled or not handled.

Entry conditions for handler:
Which inputs are in what registers and what RAM locations.

Normal exit conditions if handled:
Which outputs are expected in what registers and what RAM locations IF the poll is handled.

Normal exit conditions if not handled:
Even if a handler does not declare a poll handled, it may perform actions which affect registers or RAM. (One such poll is pVER$, which expects LEX files to build on the VER$ string and manipulate values in scratch registers, but NOT to handle the poll.) This field describes the required contents of registers and RAM on exit from a handler which does not handle the poll.

Error exit conditions:
POLL (but not FPOLL) allows a handler to indicate an error condition by returning with carry set. The code issuing the poll can discern if this happened. This field indicates what outputs are expected in this case (typically an error number is returned in the C register).

Available subroutine levels:
Indicates how many subroutine levels are available to the handler. In a POLL, the handler is executing one level shallower in the hardware return stack than the caller (because levels are saved in RAM). For example, a routine that uses 4 hardware return stack levels can issue a slow poll whose handler is allowed to use up to 5 levels. In a FPOLL, the handler is executing two levels DEEPER than the caller,

because no levels are saved in RAM. For example, a routine that uses 4 hardware return stack levels can issue a fast poll whose handler is allowed to use up to 2 levels.

NOTE:
Important things to know about the h@Bdling of the poll.

What registers/RAM may be used if handled?:
A list of resources (registers, RAM storage, status settings, etc) which may be altered if the poll is handled. Since handling a poll terminates the polling process, in some cases the handler may use registers that contained input to the handler. ONLY the resources mentioned in this field are available for use by the poll handler in this situation. In some cases it may be possible to add more resources to this list after careful research and testing to demonstrate that no conflict is possible. Any request to add a resource to this list should be communicated to HP as described at the start of this chapter.

What registers/RAM may be used if not handled?:
A list of resources (registers, RAM storage, status settings, etc) which may be altered if the poll is not handled. ONLY the resources mentioned in this field are available for use by the poll handler in this situation. In some cases it may be possible to add more resources to this list after careful research and testing to demonstrate that no conflict is possible. Any request to add a resource to this list should be communicated to HP as described at the start of this chapter.

What registers/RAM may be used if error exit?:
A list of resources (registers, RAM storage, status settings, etc) which may be altered if the poll handler performs an error return (applies only for POLL since FPOLL does not provide for an error return from the handler). ONLY the resources mentioned in this field are available for use by the poll handler in this situation. In some cases it may be possible to add more resources to this list after careful research and testing to demonstrate that no conflict is possible. Any request to add a resource to this list should be communicated to HP as described at the start of this chapter.

Special memory/pointer considerations:
Are pointers or memory in an unusual state (as in CALC mode)?

Envisioned application(s):
Possible machine extensions envisioned when the poll was designed.

History:
A record of the development of this poll interfacr.

## 1.3 Supported Entry Points

The following lists the HP-71 Operating System supported entry points together with their absolute addresses and titles.

| Name | Address | Title |
|---|---|---|
| #CK | (03356) | Check for # |
| -LINE | (15275) | Delete Through End Of Line |
| 1/X15 | (0C33E) | 1/X |
| ?PRFI+ | (17380) | Check File Protection |
| ?PRFIL | (1737E) | Check File Protection |
| A-MULT | (1B349) | Multiply Two 20-bit Hex Integers |
| ACCEPT | (0450F) | Funny function parse error reentry point |
| ACOS12 | (0DBD3) | ArcCos Inv Trig (12-dig argument) |
| ACOS15 | (0DBD7) | ArcCos Inv Trig (15-dig argument) |
| AD15M | (0C366) | Add according to modes |
| AD15S | (0E19D) | 15-digit subtract/add routine |
| AD15s | (0C369) | Add with XM sticky |
| AD2-12 | (0C35F) | Add two 12 digit forms |
| AD2-15 | (0C363) | Add two 15 digit forms |
| ADDF | (0C372) | Add for finite args only |
| ADDONE | (0C330) | Add One |
| ADDP | (03A03) | ADD Statement Parse |
| ADDRSS | (0F527) | Find Address Of A Variable |
| ADHEAD | (181B7) | Add String Header |
| ADJA | (1289A) | Absolute Time Adjust Routine |
| ADJN | (12825) | Set And Normal Adjust Routine |
| ADRS40 | (0F52B) | Find Address Of A Variable |
| ADRS50 | (0F551) | Find Address Of Var Not Of Parm Chain |
| ADRS80 | (0F567) | Find Address Of Var Not Of Parm Chain |
| ADRSUB | (0F4CF) | Get Variable Name From Token Stream |
| ALLDUN | (04BEF) | Lex Analysis |
| ARG12 | (0D67B) | Return Arg of X+iY (12-dig args) |
| ARG15 | (0D67F) | Return Arg of X+iY (15-dig args) |
| ARGERR | (0BF19) | Report "Invalid Arg" Error. |
| ARGF | (0D6A4) | Return Arg of X+iY (15-dig finite args) |
| ARGPR+ | (0E8EB) | Reads modes, pops and norm. real nbr |
| ARGPRP | (0E8EF) | Pops and normalizes real number |
| ARGST- | (0E910) | Pops and tests real number |
| ARGSTA | (0E90C) | Pops and tests real number |
| ARITH | (061E0) | Get Text For An Arithmetic Operator |
| ARRYCK | (0366A) | Parses Doubly Dimensioned Array |
| ARYDC | (05178) | Array Decompile |

| | | |
|---|---|---|
| ARYELM | (0B5A7) | Compute Array Size, # Elements |
| ARYSIZ | (0B61B) | Compute Array Size, # Elements |
| ASCICK | (0514E) | Ascii Stream Decompiler |
| ASCII | (0079B) | ASCII Bit Pattern Tables |
| ASIN12 | (0DBC8) | ArcSin Inv Trig (12-dig argument) |
| ASIN15 | (0DBCC) | ArcSin Inv Trig (15-dig argument) |
| ASLW3 | (0ED21) | Shift A Left 3 Nibbles |
| ASLW4 | (0ED1E) | Shift A Left 4 Nibbles |
| ASLW5 | (0ED1B) | Shift A Left 5 Nibbles |
| ASNMNT | (0F5E0) | Perform Variable Assignment |
| ASRW3 | (0ED10) | Shift A Right 3 Nibbles |
| ASRW4 | (0ED0D) | Shift A Right 4 Nibbles |
| ASRW5 | (0ED0A) | Shift A Right 5 Nibbles |
| ATAN15 | (0DBBE) | ArcTan Inv Trig (15-dig argument) |
| ATNCLR | (00510) | Clear Attention Flags |
| AVE=C | (18BBB) | Update AVMEME From D1 or C |
| AVE=D1 | (18BB8) | Update AVMEME From D1 or C |
| AVS2DS | (09708) | AvMemSt to display |
| BACK1B | (13B0C) | Back up the File Pointer by 1 Byte |
| BACK2B | (13B0A) | Back up the File Pointer by 2 Bytes |
| BACK3B | (13B08) | Back up the File Pointer by 3 Bytes |
| BASCHA | (07741) | Verify File Type in R2 is BASIC |
| BASCHK | (0773E) | Verify File Type in R2 is BASIC |
| BASE | (0F953) | Determine Option Base |
| BEEP | (0EA6E) | BEEP Keyboard Execute |
| BF2DSP | (01C0E) | Buffer to Display |
| BF2STK | (18663) | Buffer To Stack |
| BIASA+ | (0D52D) | Add Exp bias to A |
| BIASC+ | (0D540) | Add Exp bias to C |
| BIG | (0B747) | Create Special Consts |
| BLDBIT | (019BC) | Build Bit Patterns in Display |
| BLDCON | (16279) | Build A Constant For CalC MODE |
| BLDDSP | (01898) | Build Display Pattern from Buffer |
| BLDLCD | (0189C) | BLDDSP Except Display Status Active |
| BLNKCK | (051C1) | Blank Check |
| BOPNM- | (1B864) | Process uOPNM- token during backup |
| BP+C | (0EB40) | Machine-level Beep |
| BRT30 | (0DBE3) | Inv Trig, defined by status |
| BRTF | (0DC15) | Inv Trig, finite arg, defined by status |
| BSCEX2 | (0743A) | BASIC Stmt/Pgm Execution: Program Exec |
| BSCEXC | (07437) | BASIC Stmt/Pgm Execution: Keyboard Exec |
| BSCEXT | (075CF) | BASIC Stmt/Pgm Exec: Reentry into BASIC loop |
| BSERR | (0939A) | BASIC system error |
| BldIM+ | (1BA6A) | Put tokens from C into BldIMG stream |
| BldIMA | (1BA66) | Put 1 or 2 tokens from A into BldIMG |
| BldIMG | (1BA68) | Put tokens from C into BldIMG stream |
| C+A2D1 | (1C053) | Recover offset from RAM storage |
| CALBIN | (18D8C) | Binary program call BASIC subprogram |
| CALL | (18DAE) | Sub-program call execution |
| CALLP | (0389C) | CALL Statement Parse |
| CAT$20 | (06746) | Build CATalog Information Buffer |

| | | |
|---|---|---|
| CATC++ | (03F66) | Convert to Uppercase, Categorize Character |
| CATCH+ | (03F69) | Convert to Uppercase, Categorize Character |
| CATCHR | (03F70) | Categorize Character |
| CATEDT | (06435) | Display CATalog Info on the Current File |
| CHAIN+ | (07C12) | Chain Subprograms, Labels, DEF FNs |
| CHAIN- | (07C1C) | Chain Subprograms, Labels, DEF FNs |
| CHEDIT | (14C99) | Character Editor |
| CHIRP | (0EC5A) | Do An Annoying Little Beep |
| CHKEOL | (13D6D) | Check if at End of Statement |
| CHKmem | (012C7) | Check Available Memory Without Leeway |
| CHNHED | (0F579) | Point To Variable Chain Head |
| CK"ON" | (076AD) | Check ON / ATTN Key |
| CKINF- | (18534) | Specify DISP Stmt & Set Handler Info |
| CKINFO | (18542) | Check Handler Information |
| CKSREQ | (00721) | Handle service requests |
| CLASSA | (0D590) | Classification of numeric arg |
| CLOSEA | (120E4) | Close All Open Files |
| CLOSEF | (12087) | Close File |
| CLRFRC | (0C6F4) | Clear fractional part |
| CMD1ST | (01654) | Set command stack pointer to 1st cmd |
| CMDFND | (01693) | Find Nth Command Stack Entry |
| CMDINI | (016D1) | Recalls CMDPTR and MAXCMD |
| CMDPR" | (01627) | Text for command stack prompt |
| CMDS20 | (01672) | Display Cmd Stack Entry |
| CMPT | (125B2) | Return Current Time |
| CNFFND | (109AC) | Configuration Buffer Find |
| CNFLCT | (0BD15) | Report "Data Type" Error. |
| CNVUCR | (152A7) | Convert To Upper Case |
| CNVWUC | (03FB8) | Converts 8 chars to uppercase |
| COLDST | (00000) | Cold starts machine |
| COLLAP | (091FB) | Collapse Math Stack |
| COMCK | (036CD) | Comma Check |
| COMCK+ | (032AE) | Check Comma & Output Comma Token |
| CONCOM | (0467E) | Compile a Numeric Constant |
| CONF | (10212) | Configure Everything |
| CONVUC | (152AA) | Convert To Upper Case |
| COPYu | (08269) | COPY Utility |
| CORUPT | (09083) | Report "System Error" error |
| COS12 | (0D721) | Trig: Cosine of 12-dig arg |
| COS15 | (0D725) | Trig: Cosine of 15-dig arg |
| COUNTC | (1C346) | Count output characters in IMAGE field |
| CPLN10 | (07887) | Compute Line # with DO anywhere in stmt |
| CRDFIL | (1D21D) | Copy Card Into RAM |
| CREATE | (115A7) | Statement to Create Data File |
| CRETF+ | (084C4) | Create file in MAIN or in IRAM |
| CRFSB- | (11664) | Create a File in Mainframe |
| CRLFND | (0229E) | Send CR/LF to display with no delay |
| CRLFOF | (02296) | Send cursor off/CR/LF to disp w/o delay |
| CRLFSD | (022A2) | Send CR/LF to display with delay |
| CRTF | (116C1) | Create File in MAIN, PORT, or HPIL |
| CSL9RO | (1BA0D) | Copy D1 to R0(9-5) |

| | | |
|---|---|---|
| CSLC1 | (1B441) | Perform 1 CSLC |
| CSLC10 | (1B418) | Perform 10 CSLCs |
| CSLC11 | (1B41B) | Perform 11 CSLCs |
| CSLC12 | (1B41E) | Perform 12 CSLCs |
| CSLC13 | (1B421) | Perform 13 CSLCs |
| CSLC14 | (1B424) | Perform 14 CSLCs |
| CSLC15 | (1B427) | Perform 15 CSLCs |
| CSLC2 | (1B43E) | Perform 2 CSLCs |
| CSLC3 | (1B43B) | Perform 3 CSLCs |
| CSLC4 | (1B438) | Perform 4 CSLCs |
| CSLC5 | (1B435) | Perform 5 CSLCs |
| CSLC6 | (1B432) | Perform 6 CSLCs |
| CSLC7 | (1B42F) | Perform 7 CSLCs |
| CSLC8 | (1B42C) | Perform 8 CSLCs |
| CSLC9 | (1B415) | Perform 9 CSLCs |
| CSLW3 | (0ED43) | Shift C Left 3 Nibbles |
| CSLW4 | (0ED40) | Shift C Left 4 Nibbles |
| CSLW5 | (0ED3D) | Shift C Left 5 Nibbles |
| CSRC1 | (1B427) | Perform 1 CSRC |
| CSRC10 | (1B432) | Perform 10 CSRCs |
| CSRC11 | (1B435) | Perform 11 CSRCs |
| CSRC12 | (1B438) | Perform 12 CSRCs |
| CSRC13 | (1B43B) | Perform 13 CSRCs |
| CSRC14 | (1B43E) | Perform 14 CSRCs |
| CSRC15 | (1B441) | Perform 15 CSRCs |
| CSRC2 | (1B424) | Perform 2 CSRCs |
| CSRC3 | (1B421) | Perform 3 CSRCs |
| CSRC4 | (1B41E) | Perform 4 CSRCs |
| CSRC5 | (1B41B) | Perform 5 CSRCs |
| CSRC6 | (1B418) | Perform 6 CSRCs |
| CSRC7 | (1B415) | Perform 7 CSRCs |
| CSRC8 | (1B42C) | Perform 8 CSRCs |
| CSRC9 | (1B42F) | Perform 9 CSRCs |
| CSRW3 | (0ED32) | Shift C Right 3 Nibbles |
| CSRW4 | (0ED2F) | Shift C Right 4 Nibbles |
| CSRW5 | (0ED2C) | Shift C Right 5 Nibbles |
| CURBOT | (10059) | Cursor Bottom |
| CURDVC | (0A60B) | Classify Current File's Device |
| CURSFL | (151DF) | Move Cursor To Far Left |
| CURSFR | (151D7) | Move Cursor To Far Right |
| CURSRD | (100A4) | Cursor Down |
| CURSRT | (096C1) | Count cursor-rights |
| CURSRU | (1009A) | Cursor Up |
| CURTOP | (10063) | Cursor Top |
| CVUCW | (03FBC) | Converts 8 chars to uppercase |
| CkLoop | (1B669) | IMAGE parse loop to check for edit chars |
| CkLpNC | (1B66D) | IMAGE parse loop, no symbol count |
| D0+2RD | (13A32) | Move file pointer&check buffer overflow |
| D0=AVS | (09B2C) | Set D0=address in AVMEMS |
| D0=FIB | (13AC5) | Set D0,C(A) to value at STMTD1 |
| D0=PCA | (09B37) | Set D0=address in PCADDR |

| | | |
|---|---|---|
| DOASC+ | (0982C) | Send ASCII bytes to DATO |
| DOASCI | (09833) | Send ASCII bytes to DATO |
| D12ROA | (1BA3C) | Copy D1 to R0(A) |
| D1=AVE | (18651) | Set D1 to (AVMEME) |
| D1@AVS | (01299) | Set D1 to Available Memory Start |
| D1C=R3 | (03047) | Restore C(A),D1 from R3 |
| D1FSTK | (1955D) | Set D1 to FORSTK |
| D1MSTK | (1954E) | Set D1 at MTHSTK (AVMEME) |
| D=AVME | (1A476) | Set D(A) to AVMEMS or AVMEME |
| D=AVMS | (1A460) | Set D(A) to AVMEMS or AVMEME |
| D=WORD | (04C0E) | Read 8 Bytes And Convert To Uppercase |
| DATLEN | (0B584) | Compute Data Length Given Type |
| DAY2JD | (13407) | Day# To Julian Date |
| DAYYMD | (13335) | Day# To Year, Month, Day |
| DBLPI4 | (0DAFC) | Generate 31-digit PI/4 or 45 |
| DBLSUB | (0DADD) | Double Precision Subtrace |
| DCHX=C | (1B2D0) | Convert DEC Integer To HEX Integer |
| DCHXF | (1B223) | Convert 12-digit Flt To Hex Integer |
| DCHXW | (0ECDC) | Full Word Decimal To Hex Conversion |
| DCPLIN | (10108) | Decompile line and display it |
| DCRMNT | (1C177) | Decrement multiplier in IMAGE string |
| DEBNCE | (00CF7) | Debounce and scan keyboard |
| DECHEX | (1B2D2) | Convert DEC Integer To HEX Integer |
| DECP | (0328F) | Parse of Variable Declaration Statements |
| DELAYp | (02AC6) | DELAY and WINDOW Statement Parse |
| DEST | (0F7B0) | Save Variable Destination Info |
| DISPDC | (05450) | Expression List Decompile |
| DISPP | (035A4) | DISP Statement Parse |
| DIVF | (0C4B8) | Divide for finite args only |
| DMNSN | (0AE39) | Create And Allocate Memory For Variable |
| DONNA | (09656) | Re-prompt input line |
| DPART2 | (17EA3) | IO Handler For Built-In Display |
| DPART3 | (17EF8) | Finish up DISP line |
| DPVCTR | (0AC50) | Creates Vars, Computes # Of Elements |
| DRANGE | (1B076) | Verify A Byte Is In Range "0"-"9" |
| DROPDC | (05470) | Expression List Decompile |
| DSLEEP | (0056D) | Deep sleep |
| DSP$00 | (185DB) | Create String of Readable Characters |
| DSPBUF | (09723) | Send a buffer of chars to display |
| DSPCHA | (01C3E) | Display Character |
| DSPCHC | (01C3C) | Display Character |
| DSPCL? | (020B6) | Clear display buffer if necessary |
| DSPCNA | (09721) | Display by count |
| DSPCNB | (0971F) | Display by count |
| DSPCNO | (09716) | Display by count |
| DSPLI+ | (1010F) | Display line with cursor on;calc cursor pos. |
| DSPLIN | (10127) | Display line with cursor on;pass cursor pos. |
| DSPRST | (02443) | Display reset |
| DSPUPD | (01ADA) | Display Update |
| DSTRDC | (05280) | Decompiles Variable Declarations |
| DV15M | (0C4AC) | Divide (same as DV2-15) |

| | | |
|---|---|---|
| DV15S | (0C4B2) | Divide without clearing SB |
| DV2-12 | (0C4A8) | Divide for two 12-forms |
| DV2-15 | (0C4AC) | Divide |
| DXP100 | (0CF7F) | EXP for double precision arg |
| EDIT80 | (0A5A5) | Designates Specified File as Current |
| EDITWF | (0A533) | Designates workfile as Current File |
| ENDALL | (0769A) | External Stmt entry to perform END ALL |
| ENDBIN | (0764B) | End Binary Program or Subprogram |
| ENDIMG | (1C040) | Process end of IMAGE string |
| ENDSUB | (195A8) | ENDSUB execution |
| EOLCK | (02A7E) | Check for EOL,@,!,ELSE |
| EOLCKR | (02A7A) | Check for EOL,@,!,ELSE |
| EOLDC | (05402) | End of Stmt check |
| EOLSCN | (08AA7) | tEOL Scan |
| EOLXC* | (052EC) | Check for End of Stmt Decompile |
| EOLXCK | (05405) | End of Stmt check |
| ERRM$f | (09806) | Transfer ASCII from AvMem to stack |
| ERRRTN | (074ED) | Error Exit reentry to BASIC loop |
| ESCSEQ | (023C1) | Send Escape Sequence to Display |
| EX-115 | (0CF48) | EXP(x)-1 (EXPM1(x)) |
| EX12 | (0D5C6) | Return exponent of 12-dig arg |
| EX15M | (0D5CA) | Return exponent of 15-dig arg (XM=SB=0) |
| EX15S | (0D5CE) | Return exponent of 15-dig arg |
| EXAB1 | (0D3E7) | Exchange AB with scratch 1 |
| EXAB2 | (0D40E) | Exchange AB with scratch 2 |
| EXACT | (128B0) | Compute New Accuracy Factor. |
| EXCAD+ | (08631) | Compute Exec Addr of Token |
| EXCHRe | (02E81) | "Excess Characters" Parse Error Exit |
| EXCPAR | (187E8) | Execution Time Expression Parse |
| EXDCLP | (0592E) | Funny function decompile reentry point |
| EXF | (0D5DF) | Return exponent of finite 15-dig arg |
| EXP15 | (0CF5A) | EXP(x) (exponential fcn) |
| EXPEX+ | (0F182) | Evaluate Expression |
| EXPEX- | (0F178) | Evaluate Expression |
| EXPEXC | (0F186) | Evaluate Expression |
| EXPP10 | (03FE3) | Expr Parse (specify start of parse stk) |
| EXPPAR | (03FD9) | Expression Parse |
| EXPPLS | (03FDC) | Expression Parse for Left of Equal Sign |
| EXPR | (0F23C) | Function Return |
| EXPRDC | (05922) | Expression Decompile |
| EXPSKP | (1A9AC) | Skip Over Tokenized Expression |
| FASCFD | (110C3) | Look Up File Type Given Type Name |
| FCHLBL | (0782C) | Find Label in Current BASIC File |
| FCSTRT | (0E757) | Internal Factorial |
| FGTBL | (00C9B) | State table for F & G shifted keys |
| FIBAD- | (11478) | Find FIB entry address for a channel |
| FIBADR | (11457) | Find FIB entry address for a channel |
| FIBOFF | (12132) | Reset Devices, Buffers at Power On/Off |
| FILCRD | (1C879) | Copy File To Card |
| FILDC* | (05759) | File Decompile |
| FILEF | (09FB0) | Find a file |

| | | |
|---|---|---|
| FILEP | (03E9C) | File Name Parse |
| FILEP! | (03F0F) | Literal File Name Parse |
| FILEP+ | (03F07) | Label Declaration Parse |
| FILEP- | (03F00) | Subprogram Name Parse |
| FILEP1 | (03EFC) | Literal File Name Parse |
| FILFIL | (011CE) | Fill in Missing File Name |
| FILSK+ | (06F1D) | File Skip |
| FILXQ$ | (09B95) | Filename Execute For a String Expression |
| FILXQ^ | (09B76) | Filename Execute |
| FIND | (0F563) | Find Address Of Var Not Of Parm Chain |
| FINDA | (023E3) | Look For A(B) In A Table And Jump |
| FINDD0 | (023E0) | Look For (D0) In A Table And Jump |
| FINDF | (09F77) | Find a file |
| FINDF+ | (09F63) | Find a file |
| FINDL | (0FFE4) | Find Line# within a Program File |
| FINDLB | (07786) | Find Label in Current Program |
| FINITA | (0CD03) | Is (A,B) non-finite ? |
| FINITC | (0CD0F) | Is (C,D) non-finite ? |
| FINLIN | (18A3A) | Finish line in display/video |
| FIXDC | (05493) | Expression List Decompile |
| FIXP | (02A6E) | FIX and WAIT Statement Parse |
| FLADDR | (0126B) | Find First/Last Address of Mem Device |
| FLDEVX | (01154) | Make Device Code Explicit |
| FLIP10 | (0DB9C) | Toggle status bits |
| FLIP11 | (0DBAB) | Toggle status bits |
| FLIP8 | (0DB8D) | Toggle status bits |
| FLOAT | (1B322) | Convert Dec Integer Into 12-Dig Float |
| FLTDH | (1B223) | Convert 12-digit Flt To Hex Integer |
| FLTYPp | (03E71) | Parse File Type |
| FNDFCN | (1A0A1) | Find User-Defined Function |
| FNPWDS | (0D3C0) | Weed out NaNs and Infs |
| FNRTN1 | (0F216) | Function Return |
| FNRTN2 | (0F219) | Function Return |
| FNRTN3 | (0F235) | Function Return |
| FNRTN4 | (0F238) | Function Return |
| FORUPD | (0A6AE) | FOR Stack Update |
| FPOLL | (1250A) | Fast Poll all LEX files with Process # |
| FSPECe | (02F02) | "Invalid Filespec" Parse Error Exit |
| FSPECp | (03CC5) | File Specification Parse |
| FSPECx | (09F2D) | File Specification Execute |
| FTBSCH | (11093) | Search a File Type Table by Type Number |
| FTYPDC | (06902) | File Type Decompile |
| FTYPF# | (11059) | Look Up File Type Given Type Number |
| GDISP$ | (1C3C7) | GDISP$ function execution |
| GETAVM | (1864D) | Get Available memory limits |
| GETCH# | (11427) | Get Channel Number |
| GETCON | (0DAA3) | Get constants from table |
| GETDIM | (0AD6B) | Get A Dimlimit From Stack |
| GETMSK | (01BBA) | Get Mask for Character Protection Bitmap |
| GETNAM | (1A085) | Get variable name |
| GETPR1 | (06BFB) | Get File Protection of Specified File |

| | | |
|---|---|---|
| GETPRO | (06BEE) | Get File Protection of Current File |
| GETSA | (0E551) | Tests current statistical array |
| GETST* | (07716) | Get Start/EOF any file/check Filetype |
| GETST- | (07728) | Get Start/EOF Curr File/don't check Filetype |
| GETSTC | (07726) | Get Start/EOF Curr File/check Filetype |
| GETVAL | (0DAB2) | Get constants from table |
| GNXTCR | (03064) | Get Next Non-blank Character |
| GOSUB | (079E9) | Statement Execution |
| GOSUBp | (029F6) | GOSUB Statement Parse |
| GOTO | (079FA) | Statement Execution |
| GOTODC | (0552E) | GOTO Decompile |
| GOTOp | (029F6) | GOTO Statement Parse |
| GTEXT | (05079) | Get Text for Keyword/Function |
| GTEXT+ | (05199) | GTEXT Preprocessor |
| GTEXT1 | (051A5) | GTEXT Preprocessor |
| GTFLAG | (1365E) | Gets RAM nib and flag mask |
| GTKYC+ | (08D9B) | Get Keycode |
| GTKYCD | (08D92) | Get Keycode |
| GTPTRS | (14636) | Get File Pointers from FIB |
| GTPTRX | (14670) | Get File Pointers from FIB |
| GTXT++ | (05192) | GTEXT Preprocessor |
| GetEXP | (1C086) | Expression execute for IMAGE output list |
| HASH1 | (1B0A1) | Indexed Jump Through A GOTO Table |
| HASH2 | (1B0A3) | Indexed Jump Through A GOTO Table |
| HDFLT | (1B31B) | Convert HEX Integer To DEC Flt-pt |
| HEXASC | (17148) | Convert Hexadecimal to Ascii |
| HEXDEC | (0ECAF) | Hex To Decimal Conversion |
| HMSSEC | (13274) | Hours, Mins, Secs To Seconds. |
| HNDLFL | (0CBC9) | HANDLE FLAG SETTING |
| HTRAP | (0CB2F) | HANDLE TRAPS |
| HUGE | (0B75D) | Create Special Consts |
| HXDASC | (05FF4) | Hex to decimal ASCII conversion |
| HXDCW | (0ECB4) | Hex To Decimal Conversion |
| I/OAL+ | (1197B) | I/O Buffer Allocate |
| I/OALL | (1197D) | I/O Buffer Allocate |
| I/OCOL | (11979) | I/O Buffer Collapse |
| I/OCON | (11920) | I/O Buffer Contract From Buffer End |
| I/ODAL | (11A41) | I/O Buffer Deallocate |
| I/OEX2 | (11A0F) | I/O Buffer Expand |
| I/OEXP | (11A11) | I/O Buffer Expand |
| I/OFND | (118BA) | I/O Buffer Find |
| I/ORES | (118FF) | I/O Buffer Restore |
| IDIV | (0EC7B) | Full Word Integer Divide. |
| IDIVA | (0EC6E) | A-field Integer Divide |
| IF12A | (0C739) | Integer/Fraction Split |
| ILCNTe | (02E70) | "Illegal Context" Parse Error Exit |
| IMDO+2 | (1BA2D) | Add 2 to R1(A), copy value to D0 |
| IMDO-2 | (1BA21) | Subtract 2 from R1(A) |
| IMerr | (1B989) | Report "Invalid IMAGE" error |
| IMinit | (1B88F) | Initiate IMAGE output field |
| IMoffs | (1BA58) | Store offset from D1 in BldIMG stream |

| | | |
|---|---|---|
| IMxq27 | (1BB9C) | Return to IMAGE token executor |
| INF*0 | (0C607) | Inf*0 exception |
| INFR15 | (0C73D) | Integer/Fraction Split |
| INPOFF | (18B49) | Restart statement after DSLEEP |
| INTGR | (0F99B) | Store Into An Integer Variable |
| INTR50 | (000DB) | Reentry point for ext. interrupt handler |
| INVNaN | (0C65F) | Create IVL NaN |
| IOFNDO | (118C1) | I/O Buffer Find |
| IOFSCR | (1188E) | I/O Find for Available Scratch Buffer |
| ISRAM? | (10192) | Pointing At RAM? |
| IVAERR | (0E920) | Report "Invalid Arg" error. |
| IVEXPe | (02E35) | "Invalid Expression" Parse Error Exit |
| IVPARe | (02E3F) | "Invalid Parameter" Parse Error Exit |
| IVVARe | (02E66) | "Invalid Variable" Parse Error Exit |
| KEY$ | (1ACA8) | KEY$ function |
| KEYCOD | (1FD22) | Keycode Map |
| KEYDEL | (08D2C) | Key Assignment Delete |
| KEYFND | (08CB8) | Key Assignment Find |
| KEYMRG | (08B8F) | Key Merge |
| KEYNAM | (1AC04) | Return key name string from keycode |
| KEYRD | (14E11) | Read A Key |
| KEYSCN | (00D4D) | Scan keyboard |
| KYDN? | (00774) | Is a Key Down in Current Row? |
| LABELP | (03E9F) | Label Reference Parse |
| LABLDC | (05702) | Lable Decompile |
| LBLINP | (02A04) | Parse Line Number or Label |
| LBLNAM | (077E7) | Get Label Name into Register A |
| LBLNIF | (02A0D) | Parse Line Number or Label after THEN/ELSE |
| LCDINI | (00665) | Initialize LCD display |
| LDCEXT | (04F5E) | Line Decompile Driver |
| LDCM10 | (04F6F) | Line Decompile Driver |
| LDCOMP | (04F69) | Line Decompile Driver |
| LDCSET | (05060) | Set D=AVMEME; D0=OUTBS |
| LDSST1 | (04F72) | Line Decompile Driver |
| LDSST2 | (04F9E) | Line Decompile Driver |
| LEAVE | (04C01) | Lexical Analysis |
| LEXBF+ | (10D0F) | Set Up LEX Files Buffer |
| LGT15 | (0D1AE) | Log base 10 |
| LIMITS | (0AC3E) | Compute Dimension Limits In Decl Stmt |
| LINNAU | (05122) | Line number decompile |
| LINND+ | (05112) | Line number decompile |
| LINNDC | (05115) | Line number decompile |
| LINEP | (02620) | Parse Main Driver after ENDLINE |
| LINEP+ | (02626) | Parse Main Driver from anywhere |
| LINP | (02A07) | Parse Line Number only |
| LISTDC | (05839) | Decompiles LIST, RENUMBER, SECURE, MERGE |
| LN1+15 | (0CD44) | LN(1+X) |
| LN1+XF | (0CD51) | LN(1+X) for finite args only |
| LN12 | (0CD7D) | LOG for 12-form args. |
| LN15 | (0CD81) | Natural Logarithm |
| LN30 | (0CD9C) | LOG entry for finite args only. |

| | | |
|---|---|---|
| LNEP66 | (027EA) | Parse Main Driver return entry |
| LNPEXT | (02617) | Parse Main Driver external entry |
| LNSKP- | (089FF) | Line Skip |
| LOCADR | (0A611) | Locate, Classify Address's Memory Device |
| LOCFIL | (1721D) | Locate File With FIB |
| LSLEEP | (006CD) | Light Sleep |
| LSTLEN | (06C27) | Calculate #chars to list in display buf |
| LXFND | (0979D) | Set D1 to LEX Table I/O Buffer |
| MAIN05 | (00338) | Main Loop |
| MAIN30 | (0037E) | Main Loop |
| MAINLP | (002FD) | Main Loop |
| MAKE1 | (0DACE) | Make 12-dig 1 in C and compare with B. |
| MAKEBF | (01751) | Make ASCII Buffer from Display Buffer |
| MEMBER | (1B098) | Check If Byte Is A Member Of A Set |
| MEMCKL | (012A5) | Check Avail Memory With, Without Leeway |
| MEMER* | (0945B) | Low-level memory error |
| MEMERR | (0944D) | Insufficient Memory error |
| MEMERX | (0944F) | Insufficient Memory error |
| MESSG | (0CC17) | MESSAGE |
| MFER42 | (0962C) | Position D0 to start of BASIC stmt. |
| MFERR | (09393) | Mainframe BASIC system error |
| MFERR* | (093F1) | Error message driver |
| MFERRS | (0939E) | Stop BASIC execution for error |
| MFERsp | (0940D) | Error Message With Text Insertion |
| MFLG=0 | (13DA1) | Clear MLFFLG nibble |
| MFWRN | (093BC) | Warning/message driver |
| MFWRNQ | (093C5) | Warning/message driver |
| MFWRQ8 | (093C3) | Warning/message driver |
| MGOSUB | (1AF01) | Execute A GOSUB From Movable Code |
| MOVE*M | (01308) | Move Memory Up or Down Without Ref Adj |
| MOVED0 | (1B0F4) | Blk Move To Higher Addr |
| MOVED1 | (1B101) | Blk Move To Higher Addr |
| MOVED2 | (1B104) | Blk Move To Higher Addr |
| MOVED3 | (1B109) | Blk Move To Higher Addr |
| MOVEDA | (1B0FA) | Blk Move To Higher Addr |
| MOVEDD | (1B106) | Blk Move To Higher Addr |
| MOVEDM | (1B0EE) | Blk Move To Higher Addr |
| MOVEU0 | (1B162) | Blk Move To Lower Addr |
| MOVEU1 | (1B16F) | Blk Move To Lower Addr |
| MOVEU2 | (1B172) | Blk Move To Lower Addr |
| MOVEU3 | (1B177) | Blk Move To Lower Addr |
| MOVEU4 | (1B174) | Blk Move To Lower Addr |
| MOVEUA | (1B168) | Blk Move To Lower Addr |
| MOVEUM | (1B15C) | Blk Move To Lower Addr |
| MP1-12 | (0C436) | Multiply for one 12-form |
| MP15S | (0C440) | Multiply without clearing SB |
| MP2-12 | (0C432) | Multiply for two 12-forms |
| MP2-15 | (0C43A) | Multiply |
| MPOP1N | (0BD8D) | Pop 1 Arg & Check For Sig NaN |
| MPOP2N | (0BD54) | Pop 2 Args W/signan Check |
| MPY | (0ECB8) | HEX * HEX Or HEX * DEC Multiply. |

| | | |
|---|---|---|
| MSN12 | (0D553) | Find most significant NaN, 12-Dig arg's |
| MSN15 | (0D557) | Find most significant NaN, 15-Dig arg's |
| MSPARe | (02E5C) | "Missing Parameter" Parse Error Exit |
| MTADDR | (08195) | Calc Main Table Address for Token |
| MTADR+ | (081A1) | Calc Main Table Address for Token |
| MULTF | (0C446) | Multiply for finite args only |
| MVMEM+ | (0133C) | Move File Memory W/Ref Adjust |
| NORDIM | (0AE2D) | Report "Var Context" Error |
| NOSCRL | (14C8A) | Request No-display-scrolling |
| NRMCON | (161AF) | Convert BLDCON Constant into Usable Form |
| NTOKEN | (0493B) | Lex Analysis |
| NTOKNL | (048E6) | Lex Analysis |
| NULLP | (07999) | Null Program Check |
| NUMC++ | (03690) | Move D1 1-Byte, Do Valid Numeric Expr Check |
| NUMCK | (0369D) | Valid Numeric Expression Check |
| NUMSCN | (04D18) | Scan Number In Lexical Analysis |
| NXTADR | (147E8) | Get Address of Next Array Element |
| NXTELM | (148AC) | Get Next Array Element |
| NXTEXP | (1C2F7) | Store pointers, execute next expression |
| NXTLIN | (10031) | Scan to Next Line |
| NXTP | (03455) | NEXT statement parse |
| NXTSTM | (08A48) | Scan to Next Stmt/Jump to BASIC Loop |
| NXTVA- | (13E58) | Get next Variable from READ list |
| NwOFFS | (1C02D) | Recover old offset, store new one in RAM |
| OAGNXT | (03060) | Output byte, Get Next Non-blank Character |
| OBCOLL | (01435) | Collapse Output Buffer |
| OBEDIT | (17687) | Edit Output Buffer |
| ONDC20 | (05501) | Keyword and Opt Line#/Label Decompile |
| ONP40 | (02B7B) | GOTO,GOSUB,RESTORE in middle of stmt Parse |
| ONTIMR | (08008) | Execute branch of ON TIMER/ERROR |
| OPENF | (11B06) | Open File |
| ORGSB | (0D65B) | Set SB if sINX=1 |
| ORSB | (0D63C) | Set SB if sIX=1 |
| ORXM | (0D633) | Set XM if sXM=1 and Set SB if sIX=1 |
| OUT1T+ | (02CDF) | Increment D1, Output 1 byte from A(B) |
| OUT1TK | (02CEB) | Output 1 byte from A(B) |
| OUT2TC | (02CFD) | Output 2 bytes from C(3-0) |
| OUT2TK | (02CFF) | Output 2 bytes from A(3-0) |
| OUT3TC | (02D12) | Output 3 bytes from C(5-0) |
| OUT3TK | (02D15) | Output 3 bytes from A(5-0) |
| OUTBY+ | (02CE5) | Increment D1, Output 1 byte from C(B) |
| OUTBYT | (02CE8) | Output 1 byte from C(B) |
| OUTC15 | (05421) | Output nibbles |
| OUTEL1 | (05300) | Exit for End of Stmt Decompile |
| OUTELA | (05303) | Output End of Stmt Terminator From A |
| OUTLI1 | (03709) | Output Delimited Literal |
| OUTLIT | (036F3) | Output Delimited Literal |
| OUTNBC | (05423) | Output nibbles |
| OUTNBS | (05426) | Output nibbles |
| OUTNIB | (02D28) | Output 1 nibble from C(0) |
| OUTRES | (0BC84) | Round And Return Result |

| | | |
|---|---|---|
| OUTVAR | (0373E) | Output Parsed Variable |
| OVFL | (0CA73) | Create overflow value |
| P1-10 | (041C1) | Numeric Operand Found |
| PARERR | (02F08) | Generic Parse Error Exit |
| PART3 | (18097) | Finishes up a PRINT class statement |
| PDEV | (09E9E) | Evaluate Num Expression as Port Device |
| PEDIT | (0FF5F) | Program Edit |
| PEDITD | (0FF62) | Program Edit to delete line |
| PFINDL | (078DF) | Find Line# Within Program |
| PFNDZL | (078E2) | Find Line# Within Program |
| PI/2 | (0DB77) | Generate PI/2 |
| PI/2D | (0DB7A) | Generate signed PI/2 |
| PI/4 | (0DAA1) | Fetch Pi/4 from table |
| POLL | (12337) | Poll LEX Files with Process Number |
| POLLD+ | (1232D) | Poll LEX Files adjusting AVMEME in D(A) |
| POP1N | (0BD1C) | Pop 1 Number Off Of Stack |
| POP1N+ | (0BD91) | Pop 1 Arg & Check For Sig NaN |
| POP1R | (0E8FD) | Pops real number from math stack |
| POP1S | (0BD38) | Pop 1 String Arg Off Stack |
| POP2N | (0BC8C) | Pop 2 Numbers From Stack. |
| POP2N+ | (0BD58) | Pop 2 Args W/signan Check |
| POPBUF | (010EE) | Pop Key Buffer |
| POPMTH | (1B3DB) | Skip Past An Item On Mthstk |
| POPSTK | (08F55) | Pop Stack |
| POPSTR | (1B405) | Skip Past An Item On Mthstk |
| POPUPD | (08F3E) | Pop Stack |
| PREP | (0ADAF) | Prepare To Create A Variable/array |
| PRESCN | (04A49) | Lex Analysis |
| PRGFMF | (0A146) | Purge File in Memory |
| PRINT* | (17F37) | PRINT class statement execution |
| PRNEXe | (02E95) | ") Expected" Parse Error Exit |
| PRNTDC | (05450) | Expression List Decompile |
| PRPSND | (06B17) | Prepare to send buffer to display |
| PRSCOO | (07B93) | Compute Program Scope; GETSTC exit cond |
| PRSsc+ | (1BA84) | IMAGE parse scan, increment D0 first |
| PRSscn | (1BA88) | IMAGE parse scan |
| PRT#DC | (06841) | Port# Decompile |
| PSHGSB | (08F13) | Push address on GOSUB Stk |
| PSHMCR | (08F0B) | Push address on GOSUB Stk |
| PSHSTK | (08C7F) | Push Stack |
| PSHSTL | (08C85) | Push Stack |
| PSHUPD | (08F0D) | Push address on GOSUB Stk |
| PUGFIB | (12198) | Purge the FIB Entries of Purged Files |
| PURGEF | (17359) | Purge Internal or External File |
| PURGDC | (05745) | PURGE, COPY Decompile |
| PUTRES | (18115) | Put Numeric Result Into RES |
| PWROFF | (00526) | Power Off |
| QUOEXe | (02E8B) | "Quote Expected" Parse Error Exit |
| QUOTCK | (0623D) | Quote and Apostrophe Check |
| R3=D10 | (03526) | Save D0 and D1 in R3 |
| R<RSTK | (014DD) | Save RSTK Level(s) Into RSTKBF Buffer |

| | | |
|---|---|---|
| RAMROM | (0A5F7) | Classify Memory Device |
| RANGE | (1B07C) | Verify A Byte Is In Certain Range |
| RCCD1 | (0D3F5) | Recall CD into scratch 1 |
| RCCD2 | (0D41C) | Recall CD into scratch 2 |
| RCL* | (0E983) | Recall Selected Math Scratch Stack Entry |
| RCLW1 | (0E981) | Recall 1st (Top) Math Scrtch Stack Entry |
| RCLW2 | (0E98E) | Recall 2nd Math Scratch Stack Entry |
| RCLW3 | (0E9C4) | Recall 3rd Math Scratch Stack Entry |
| RCSCR | (0E954) | Pop 15-Form From Math Scratch Stack |
| RCVOFS | (1C050) | Recover offset from RAM storage |
| RDATTY | (17CC6) | Report "Data Type" error |
| RDBAS | (173FF) | Read Line From Basic File |
| RDBYTA | (13A2F) | Read Byte From an Opened File Into A |
| RDCHD+ | (076EE) | Read Current File header,File length and type |
| RDCHDR | (076F0) | Read Current File header, File length |
| RDHDR1 | (076FD) | Read File header, File length |
| RDINFO | (0846B) | Read Source/Dest File Information |
| RDLNAS | (13A1F) | Read String Length from a TEXT File. |
| RDTEXT | (17489) | Read Line From Text File |
| READIN | (0F484) | Read Something In |
| READNB | (17518) | Read/Write Nibs To/From File |
| READP5 | (0323B) | Destination Variable List Parse |
| RECADR | (0F4B7) | Some Recall Utility |
| RECALL | (0F281) | Variable Recall |
| REDUCE | (15977) | Parse And Execute Partial ExpresSIONS |
| RELJMP | (05047) | Relative Jump From (D1) |
| RENSUB | (1A753) | Renumber Subroutine |
| REPROM | (18A1E) | Reprompt for input |
| RESCAN | (04A4C) | Lex Analysis |
| RESPTR | (03172) | Restore Input Pointer |
| REST* | (03035) | Restart Lex Analyzer |
| REV$ | (1B38E) | Reverse Characters In A String On Stack |
| REVPOP | (0BD31) | REV$ On String And Then POP1S |
| REWIND | (11365) | Rewind Open File |
| RFAD++ | (0A6FB) | Adjusts Refs When Mem Moves=>Higher Addr |
| RFAD+I | (0A702) | Adjusts Refs When Mem Moves=>Higher Addr |
| RFAD-- | (0A652) | Adjust Refs when mem moves to lower addr |
| RFAD-I | (0A659) | Adjust Refs when mem moves to lower addr |
| RFUPD+ | (0A66E) | Updates a ptr when mem moves |
| RJUST | (12AE2) | Unfloat A Floating-Point Number |
| RND-12 | (1B01F) | Round A 12-digit Fp Number |
| RND12+ | (0C9D5) | Round 15-form |
| RNDAHX | (136CB) | Pops, tests,rounds,converts dec to hex |
| RNDNRM | (0CAB1) | Round a Normal Number |
| ROMFND | (1102F) | Find ROM / File Chain Start |
| RPLLIN | (013F7) | Replace Line in Memory File |
| RPLSBH | (1799B) | Replace Memory File Subheader |
| RPTKY | (152BA) | Check For Repeating Keys |
| RSTK<R | (014A8) | Restore RSTK Level(s) From RSTKBF Buffer |
| RSTST | (0F5C5) | Restore Status Bits |
| RUNRT1 | (074E7) | Stmt reentry to BASIC loop;sERROR,sENDx clred |

| | | |
|---|---|---|
| RUNRTN | (074EA) | Stmt reentry to BASIC loop; sERROR cleared |
| SALLOC | (0153B) | Allocate Arbitrary Save Stack Block |
| SAVESB | (0D66E) | Put SB into sIX |
| SAVEXM | (0D663) | Put XM into sXM & SB into sIX |
| SAVGSB | (0D64E) | Put SB into sINX |
| SB15S | (0E19A) | 15-digit subtract/add routine |
| SCAN | (04C40) | Scan LEXfile Text Table For Lexeme |
| SCNRT | (022B9) | Point Cursor Past Unprotected Field |
| SCOPCK | (0915B) | Scope check |
| SCRLLR | (0212E) | Scroll Left and Right |
| SE1-10 | (04468) | String Operand Found |
| SECHMS | (13252) | Convert Secs To Hours, Mins, Secs |
| SEND20 | (17DFA) | Send Buffer to Device via Handler |
| SENDEL | (17DC1) | Send EndLine to Device via Handler |
| SENDIT | (17DE3) | Send Buffer to Device via Handler |
| SENDWD | (17E15) | Send Out Width-Sized Chunks to Device |
| SETALM | (1290D) | Set Absolute Alarm Time |
| SETALR | (12917) | Set Alarm Relative To Current Time |
| SETFMT | (0F01F) | Set Display Format |
| SETSB | (0D641) | Set SB |
| SETTMO | (13158) | Set System Timeout |
| SFLAG? | (1364C) | Tests system flag |
| SFLAGC | (13601) | Clears system flag |
| SFLAGS | (135FA) | Sets system flag |
| SFLAGT | (13608) | Toggles system flag |
| SHF10 | (0C486) | Shift to normalize |
| SHFLAC | (0DB46) | Double Precision Shift Left |
| SHFRAC | (0DB51) | Double Precision Shift Right |
| SHFRBD | (0DB5F) | Double Precision Right Shift |
| SHRT | (0F96C) | Store Into Short Variable |
| SIGCHK | (0BD98) | Report Signaling NaN |
| SIGTST | (0E636) | Handle signal NaN |
| SIN12 | (0D716) | Trig: Sine of 12-dig arg |
| SIN15 | (0D71A) | Trig: Sine of 15-dig arg |
| SKIPDC | (057F6) | Skip Rest of Statement Decompile |
| SLEEP | (006C2) | Scan KB, do LSLEEP if key buffer empty |
| SNAPR* | (01578) | Restore CPU Snapshot From Any Buffer |
| SNAPRS | (01571) | Restore CPU Snapshot From SNAPSV Buffer |
| SNAPSV | (015A7) | Save Snapshot of CPU in SNAPSV Buffer |
| SNDWD+ | (17E1F) | Send Out Width-Sized Chunks to Device |
| SPACE | (0AD9D) | Compute Space Needs For An Array |
| SPLITA | (0C6BF) | SPLIT A |
| SPLITC | (0C940) | SPLIT C |
| SPLTAC | (0C934) | Split & normalize A & C |
| SPLTAX | (0E62B) | Split, normalize A; handle signal NaN |
| SQR15 | (0C534) | Square Root |
| SQR17 | (0C553) | SQRT for finite arguments only |
| SQR70 | (0C5C3) | Set SB according to Reg C |
| SQRSAV | (0D629) | SQR for Chain calculations. |
| SRLEAS | (015EC) | Release Arbitrary Block From Save Stack |
| STAB1 | (0D3D9) | Store AB into scratch 1 |

| | | |
|---|---|---|
| STAB2 | (0D400) | Store AB into scratch 2 |
| STATRS | (172F3) | Restore Status |
| STATSV | (1732F) | Save Status S13, S11 - S0 |
| STCD2 | (0D427) | Store CD into scratch 2 |
| STKCHR | (18504) | Add a Character to a Stack Item |
| STKCMD | (155ED) | Pushes Statement On Command STACK |
| STKVCT | (1470C) | Process Array Dope Vector |
| STMBCL | (090E7) | Collapse statement buffer check |
| STMBUF | (090DF) | Collapse statement buffer check |
| STORE | (0F5F8) | Store From Stack To Variable |
| STR$00 | (1815C) | Convert Number to String(Generic) |
| STR$SB | (18149) | Convert Number to String |
| STRASN | (0F6B3) | String Assignment |
| STREQL | (1B1EF) | Test Strings For Equality |
| STRGCK | (036BA) | Valid String Expression Check |
| STRHDR | (0F09A) | String Header |
| STRHED | (14C2E) | Generate String Head on Stack |
| STRNGP | (0379D) | Parse of a Mandatory String Expression |
| STRTST | (1B1C7) | Test Strings For Equality |
| STSCR | (0E92C) | Push 15-Form Onto Math Scratch Stack |
| STUFF | (1B0B2) | Fill Memory With Stuff Or 0's |
| SUBONE | (0C327) | Subtract One |
| SVINF+ | (08457) | Save/Read File Information |
| SVINFO | (0845A) | Save/Read File Information |
| SVTRC | (0FA35) | Save Trace Information In Stmt Scratch |
| SWPBYT | (17A24) | Swap Bytes |
| SYNTXe | (02E2B) | "Syntax" Parse Error Exit |
| TAN12 | (0D72F) | Trig: Tangent of 12-dig arg |
| TAN15 | (0D733) | Trig: Tangent of 15-dig arg |
| TBLJMC | (02426) | Indexed table jump |
| TBLJMP | (0242A) | Indexed table jump |
| TBMSG$ | (099AB) | Find and Build Message From Lex Table |
| TFHDLR | (1702F) | Find Transform Handler |
| TKSCN+ | (08A6B) | Token Scan |
| TKSCN7 | (08A99) | Token Scan |
| TODT | (13229) | Time To Time-of-day And Day# |
| TONE | (0EBEB) | Machine-level Beep |
| TRACDC | (052FC) | TRACE Statement Decompile |
| TRC90 | (0DA11) | Table of numeric constants |
| TRFROM | (0FE59) | Trace Line Number |
| TRMNTR | (0F1DD) | Process Terminator In Expr Execute |
| TRSFMu | (16B84) | Transform Utility Routine |
| TRTO+ | (0FE7B) | Generate Trace Message |
| TST12A | (0D476) | Compare numbers: 12-Digit arg's A,C |
| TST15 | (0D47A) | Compare numbers: 15-Digit arg's A/B, C/D |
| TWO* | (0DB38) | Double Precision Doubler |
| TstEnd | (1C0FF) | Test IMAGE output list for end of list |
| UPCPOS | (13C67) | Update FIB Current Position |
| UPDANN | (13571) | Update Annunciator |
| USGch+ | (1BC15) | Display character during USING execution |
| USGch- | (1BC0B) | Display character during USING execution |

| | | |
|---|---|---|
| USGrst | (1BC63) | Suspend USING execution, restart parse |
| USING | (1B446) | Interpret IMAGE String |
| USINGp | (03628) | USING statement Parse |
| USloop | (1C14B) | Loop on IMAGE multiplier |
| USnm05 | (1BD12) | Execute numeric IMAGE field |
| USst03 | (1BBCE) | Output characters from address in C |
| USst05 | (1BBD4) | Output characters from address in D1 |
| VAL00 | (1AD8F) | Parse and Execute a String on Stack |
| VARDC | (0537C) | Variable Decompile |
| VARNB- | (0E28D) | Pop and Test Variable Number |
| VARNBR | (0E289) | Pop and Test Variable Number |
| VARP | (0350E) | Variable Parse |
| VIEWD1 | (15147) | View A Buffer While Keys Down |
| VRIABL | (04BC4) | Lex Analysis |
| WFTMDT | (085DD) | Write Flags, Time, Date to File Header |
| WIPOUT | (1B0AF) | Fill Memory With Stuff Or 0's |
| WRBYTC | (13A73) | Write Byte to an Opened File From C |
| WRDSC+ | (02C26) | Keyword Scan from Table |
| WRDSCN | (02C2A) | Keyword Scan from Table |
| WRITNB | (1752B) | Read/Write Nibs To/From File |
| WRTFIB | (11CEE) | Write File Information to FIB |
| WRTNUM | (139C4) | Write a Number to DATA or SDATA file. |
| WRTSTR | (1396F) | Write a string to an open TEXT file |
| WSTRFX | (138B5) | Write a String to a DATA File |
| XMTADR | (08133) | Get XWORD Main Table Address |
| XXHEAD | (1A44E) | Remove String Header (Undo ADHEAD) |
| XYEX | (0C697) | EXCHANGE X & Y |
| YMDDAY | (13304) | Convert Year,month,day To Day# |
| YMDHO1 | (130E5) | Convert Time To YYMMDD And HHMMSS |
| YMDHMS | (130DB) | Return Time And Date |
| YX2-12 | (0D274) | Y^X for 12-form arguments |
| YX2-15 | (0D27A) | Y to the X power |
| ZERBUF | (18B20) | Looks Like a Zero Length Buffer |
| uRES12 | (0C994) | User Result |
| uRESD1 | (0E1EE) | Variation of uRES12 |
| uRESNX | (0C9BD) | User Result (non exceptional) |
| uRESXT | (0C9C1) | User Result for exact results |
| uRND>P | (0C9CF) | user ROUND |
| uTEST | (0D435) | Perform comparisons |

## 1.4 Supported Non-Entry Point Symbols

The following table lists other supported symbols which are defined by various modules in the operating system. These symbols are not entry points, but are externally referenced between modules. Examples include the symbolic names for fixed RAM locations, poll process numbers, and so forth.

It is the intent of HP to preserve the values of these supported symbols through any future updates of the operating system. However, HP reserves the right to adjust the values of supported symbols in any manner it chooses. A file containing these symbol values may be obtained by contacting the HP Portable Computer Division Product Support Group at (503) 757-2000.

| Name | Hex Value |
|---|---|
| ACTIVE | 2F5A8 |
| ALRM1 | 2F719 |
| ALRM2 | 2F725 |
| ALRM3 | 2F731 |
| ALRM4 | 2F73D |
| ALRM5 | 2F749 |
| ALRM6 | 2F755 |
| ANN1.5 | 2E101 |
| ANNAD1 | 2E100 |
| ANNAD2 | 2E102 |
| ANNAD3 | 2E34C |
| ANNAD4 | 2E34E |
| ATNDIS | 2F441 |
| ATNFLG | 2F442 |
| AUTINC | 2F6CB |
| AVMEME | 2F599 |
| AVMEMS | 2F594 |
| BACK | 1BA4F |
| BASICs | 000B5 |
| BitsOK | 00001 |
| CALSTK | 2F5AD |
| CHN#SV | 2F96F |
| CHNLST | 2F5BE |
| CKSUM2 | 0AA81 |
| CKSUM3 | 153A9 |
| CKSUM4 | 1DBA6 |
| CLASSA | 0D590 |
| CLCBFR | 2F576 |
| CLCSTK | 2F585 |
| CLRPRM | 04827 |
| CMDPTR | 2F6D4 |
| CMOSTV | 0168F |
| CMOSTW | 2F438 |
| CNTADR | 2F67E |
| CONFST | 2F9E6 |
| CR | 2C000 |
| CSPEED | 2F977 |
| CURREN | 2F56C |
| CURRL | 2F7E8 |
| CURRST | 2F55D |
| CURSOR | 2F47E |

| | |
|---|---|
| Clear | 00005 |
| CurOff | 00006 |
| D1MST+ | 13E21 |
| DATPTR | 2F692 |
| DCONTR | 2E3FE |
| DD1CTL | 2E3FF |
| DD1END | 2E34C |
| DD1ST | 2E300 |
| DD2CTL | 2E2FF |
| DD2END | 2E260 |
| DD2ST | 2E200 |
| DD3CTL | 2E1FF |
| DD3END | 2E160 |
| DD3ST | 2E104 |
| DEFADR | 2F967 |
| DELAYT | 2F948 |
| DISINT | 2F470 |
| DISPt | 00000 |
| DPOS | 2F94D |
| DSPBFE | 2F540 |
| DSPBFS | 2F480 |
| DSPCHX | 2F674 |
| DSPDGT | 2F6DD |
| DSPFMT | 2F6DC |
| DSPMSK | 2F540 |
| DSPSET | 2F7B1 |
| DSPSTA | 2F475 |
| DVZNIB | 2F6FC |
| DWIDTH | 2F94F |
| DZP | 00003 |
| EFIELD | 00000 |
| EOLLEN | 2F95A |
| EOLSTR | 2F95B |
| ERR# | 2F7E4 |
| ERRADR | 2F688 |
| ERRL# | 2F7EC |
| ERRLCH | 2F97C |
| ERRSUB | 2F683 |
| ESCSTA | 2F47B |
| EndNum | 000E6 |
| Except | 0000C |
| F-R0-0 | 2F89B |
| F-R0-1 | 2F8A0 |
| F-R0-2 | 2F8A5 |
| F-R0-3 | 2F8AA |
| F-R1-0 | 2F8AB |
| F-R1-1 | 2F8B0 |
| F-R1-2 | 2F8B5 |
| F-R1-3 | 2F8BA |
| FIRSTC | 2F47C |
| FLGREG | 2F6E9 |

| | |
|---|---|
| FNDCLR | 1DAEF |
| FORSTK | 2F59E |
| FRange | 0B46A |
| FUNCD0 | 2F8B8 |
| FUNCD1 | 2F8C0 |
| FUNCR0 | 2F89B |
| FUNCR1 | 2F8AB |
| GSBSTK | 2F5A3 |
| HPSCRH | 2F97F |
| INADDR | 2F6D4 |
| INBS | 2F6C6 |
| INTA | 2F410 |
| INTB | 2F420 |
| INTM | 2F430 |
| INTR4 | 2F400 |
| INTRPT | 0000F |
| INXNIB | 2F6F9 |
| IOBFEN | 2F576 |
| IOBFST | 2F571 |
| IS-DSP | 2F78D |
| IS-INP | 2F79B |
| IS-PLT | 2F7A2 |
| IS-PRT | 2F794 |
| IS-TBL | 2F78D |
| IVARG | 0D749 |
| IVLNIB | 2F6FD |
| IVP | 00004 |
| InhEOL | 00004 |
| Insert | 00007 |
| KCOL0 | 2F46F |
| KCOL1 | 2F46E |
| KCOL2 | 2F46D |
| KCOL3 | 2F46C |
| KCOL4 | 2F46B |
| KCOL5 | 2F46A |
| KCOL6 | 2F469 |
| KCOL7 | 2F468 |
| KCOL8 | 2F467 |
| KCOL9 | 2F466 |
| KCOLA | 2F465 |
| KCOLB | 2F464 |
| KCOLC | 2F463 |
| KCOLD | 2F462 |
| KEYBUF | 2F444 |
| KEYPTR | 2F443 |
| KEYSAV | 2F462 |
| LASTFN | 000B4 |
| LBLINM | 2F871 |
| LDCSPC | 2F6C1 |
| LEEWAY | 000D4 |
| LEXPTR | 2F6CF |

| | |
|---|---|
| LOCKWD | 2F7B2 |
| LOOPST | 2F7AC |
| LXTXTT | 1EE9F |
| MAINEN | 2F571 |
| MAINST | 2F558 |
| MAXCMD | 2F976 |
| MBOX^ | 2F7A9 |
| MLFFLG | 2F870 |
| MTHSTK | 2F599 |
| NEEDSC | 2F94A |
| NUMC+0 | 03696 |
| NXTIRQ | 2F70D |
| NoCont | 0000E |
| OFFFLG | 2F442 |
| OKP | 00000 |
| ONINTR | 2F68D |
| OUTBS | 2F58F |
| OVFNIB | 2F6FB |
| OVP | 00002 |
| PCADDR | 2F679 |
| PNDALM | 2F761 |
| PPOS | 2F956 |
| PRGMEN | 2F567 |
| PRGMST | 2F562 |
| PRINTt | 00001 |
| PRMCNT | 2F94B |
| PRMPTR | 2F5B7 |
| PWIDTH | 2F958 |
| PgmRun | 0000D |
| R1REV | 00785 |
| R2REV | 0AA83 |
| R3REV | 153AB |
| R4REV | 1DBA8 |
| RAMEND | 2F5B2 |
| RAWBFR | 2F580 |
| RESERV | 2F986 |
| RESREG | 2F7C2 |
| RFNBFR | 2F57B |
| RNSEED | 2F6FE |
| ROMCID | 00BFE |
| ROWDVR | 2E350 |
| RSTKBF | 2F820 |
| RSTKBp | 2F81F |
| ResetC | 00008 |
| S-R0-0 | 2F871 |
| S-R0-1 | 2F876 |
| S-R0-2 | 2F87B |
| S-R0-3 | 2F880 |
| S-R1-0 | 2F881 |
| S-R1-1 | 2F886 |
| S-R1-2 | 2F88B |

| | |
|---|---|
| S-R1-3 | 2F890 |
| SAVSTK | 2F59E |
| SCREX0 | 2F941 |
| SCREX1 | 2F951 |
| SCREX2 | 2F961 |
| SCREX3 | 2F971 |
| SCROLT | 2F946 |
| SCRPTR | 2F966 |
| SCRST0 | 2F901 |
| SCRTCH | 2F901 |
| SNAPBF | 2F7F0 |
| STATAR | 2F7AD |
| STMTD0 | 2F891 |
| STMTD1 | 2F896 |
| STMTR0 | 2F871 |
| STMTR1 | 2F881 |
| STSAVE | 2F6BE |
| SYSEN | 2F58A |
| SYSFLG | 2F6D9 |
| SavLvl | 00005 |
| SetAVM | 1B9FA |
| TASTK | 2F599 |
| TERCHR | 2F97D |
| TFORN | 2F59E |
| TGSBS | 2F5A3 |
| TIMAF | 2F787 |
| TIMER1 | 2E3F8 |
| TIMER2 | 2E2F8 |
| TIMER3 | 2E1F8 |
| TIMLAF | 2F77B |
| TIMLST | 2F76F |
| TIMOFS | 2F763 |
| TMRAD1 | 2F697 |
| TMRAD2 | 2F69C |
| TMRAD3 | 2F6A1 |
| TMRIN1 | 2F6A6 |
| TMRIN2 | 2F6AE |
| TMRIN3 | 2F6B6 |
| TRACEM | 2F7B0 |
| TRFMBF | 2F8C5 |
| TRKDON | 1CFAC |
| TRPREG | 2F6F9 |
| Trace | 0000F |
| UNFNIB | 2F6FA |
| UNP | 00001 |
| UPD1EN | 2F599 |
| UPD1ST | 2F55D |
| UPD2EN | 2F6A6 |
| UPD2ST | 2F674 |
| VALCHK | 1AE61 |
| VECTOR | 2F43C |

| | |
|---|---|
| ValSub | 0000A |
| WINDLN | 2F473 |
| WINDST | 2F471 |
| XDelay | 00009 |
| XROM01 | 00001 |
| a! | 00021 |
| a" | 00022 |
| a$ | 00024 |
| a' | 00027 |
| a. | 0002E |
| a0 | 00030 |
| a1 | 00031 |
| a2 | 00032 |
| a3 | 00033 |
| a4 | 00034 |
| a5 | 00035 |
| a6 | 00036 |
| a7 | 00037 |
| a8 | 00038 |
| a9 | 00039 |
| bALTCH | 00BFB |
| bASSGN | 00804 |
| bCARD | 00807 |
| bCHARS | 00BFB |
| bECOMD | 00809 |
| bFBF#E | 00BCF |
| bFBF#S | 00BA0 |
| bFIB | 00803 |
| bFILE | 00805 |
| bFLI01 | 0080A |
| bFLI02 | 0080B |
| bFLI03 | 0080C |
| bFLI04 | 0080D |
| bFLI05 | 0080E |
| bIEXKY | 00802 |
| bLEX | 00BFC |
| bPILAI | 00810 |
| bPILSV | 0080F |
| bROMTB | 00BFE |
| bSCRTC | 00E00 |
| bSTART | 00808 |
| bSTAT | 00806 |
| bSTMT | 00801 |
| bSTMXQ | 00811 |
| cC->C | 00068 |
| cR->C | 00069 |
| cRCL | 00067 |
| dCARD | 00007 |
| dIRAM | 00001 |
| dMAIN | 00000 |
| dPCRD | 00007 |

| | |
|---|---|
| dPORT | 00001 |
| eNofN | 000F7 |
| e0^0 | 00006 |
| e0^NEG | 00005 |
| e1^INF | 00011 |
| e2MROM | 0001A |
| eAF | 0001B |
| eALGN | 000F0 |
| eCALGN | 00060 |
| eCHNLN | 00029 |
| eDATTY | 0001F |
| eDVCNF | 00040 |
| eEOFIL | 00036 |
| eEXCHR | 0004E |
| eEXPO | 00003 |
| eEXPCT | 000E7 |
| eF2BIG | 0004A |
| eFACCS | 0003C |
| eFEXST | 0003B |
| eFILE | 000EA |
| eFNNtF | 00021 |
| eFOPEN | 0003E |
| eFPROT | 0003D |
| eFSPEC | 0003A |
| eFTYPE | 0003F |
| eFnFND | 00039 |
| eFwoNX | 0002A |
| eIF*ZR | 00010 |
| eIF-IF | 0000F |
| eIF/IF | 0000E |
| eILCNT | 0004F |
| eILEXP | 00050 |
| eILKEY | 00055 |
| eILLEG | 000E6 |
| eILPAR | 00051 |
| eILTFM | 00037 |
| eILVAR | 00053 |
| eIMGOV | 0002F |
| eINF | 000F3 |
| eINF^0 | 00012 |
| eINPUT | 000F4 |
| eINVIM | 0002D |
| eINVLD | 000EC |
| eINVST | 000ED |
| eINVUS | 0002E |
| eINX | 00015 |
| eIVARG | 0000B |
| eIVSAR | 00033 |
| eIVSOP | 00035 |
| eIVSTA | 00034 |
| eIVTAB | 00030 |

| | |
|---|---|
| eL2LNG | 00041 |
| eLNO | 0000C |
| eLOBAT | 00016 |
| eLOG- | 0000D |
| eMEM | 00018 |
| eMMCOR | 00017 |
| eMPI | 00019 |
| eMSPAR | 00052 |
| eNEG^X | 00009 |
| eNFOUN | 000E8 |
| eNODAT | 00020 |
| eNOTIN | 00043 |
| eNSVAR | 00033 |
| eNUMIN | 00026 |
| eNVSTR | 00033 |
| eNXwoF | 0002B |
| eOVFL* | 000F5 |
| eOVFLW | 00002 |
| ePALGN | 0005E |
| ePLLC | 0005A |
| ePLLC# | 00059 |
| ePRCER | 00054 |
| ePRMIS | 00024 |
| ePRNEX | 0004C |
| ePROTD | 00042 |
| ePRTCT | 000F8 |
| ePULL | 000F6 |
| eQUOEX | 0004D |
| eROWRN | 00056 |
| eR1WRN | 00057 |
| eRALGN | 0005D |
| eRECOR | 0001D |
| eRWERR | 00046 |
| eRwoGS | 0002C |
| eSIGOP | 00013 |
| eSPGNF | 00031 |
| eSQR- | 0000A |
| eSTMNF | 0001E |
| eSTROV | 00025 |
| eSUBSC | 0001C |
| eSYNTX | 0004B |
| eSYSER | 00017 |
| eTFFLD | 00038 |
| eTFM | 000F1 |
| eTFWRN | 00058 |
| eTNINF | 00004 |
| eTOO | 000EF |
| eTOOFI | 00028 |
| eTOOMI | 00027 |
| eTRKDN | 00061 |
| eTRKOF | 000E5 |

| | |
|---|---|
| eTUFAS | 00047 |
| eTUSLO | 00048 |
| eUALGN | 0005F |
| eUNFLW | 00001 |
| eUNKCD | 00045 |
| eUNORC | 00014 |
| eVALGN | 0005C |
| eVARTY | 00032 |
| eVFYER | 00044 |
| eWALGN | 0005B |
| eWRGNM | 00049 |
| eXFNNF | 00022 |
| eXWORD | 00023 |
| eZRDIV | 00008 |
| eZRO/0 | 00007 |
| enull | 00000 |
| ew/o | 000EB |
| fAOS | 000DF |
| fASCII | 00001 |
| fBASIC | 0E214 |
| fBIN | 0E204 |
| fDATA | 0E0F0 |
| fEOF | 000FF |
| fEOR | 000EF |
| fEOS | 0006F |
| fKEY | 0E20C |
| fLEX | 0E208 |
| fLIF1 | 00001 |
| fMOS | 0007F |
| fSDATA | 0E0D0 |
| fSOS | 000CF |
| fTEXT | 00001 |
| f1AC | FFFC7 |
| f1ALRM | FFFC4 |
| f1BASE | FFFF0 |
| f1BAT | FFFC3 |
| f1BEEP | FFFFE |
| f1BPLD | FFFE7 |
| f1CALC | FFFC0 |
| f1CLOC | FFFD3 |
| f1CMDS | FFFD1 |
| f1CTON | FFFFD |
| f1CTRL | FFFD0 |
| f1DG0 | FFFEF |
| f1DG1 | FFFEE |
| f1DG2 | FFFED |
| f1DG3 | FFFEC |
| f1DORM | FFFD5 |
| f1DVZ | FFFF9 |
| f1EOT | FFFE9 |
| f1EXAC | FFFD2 |

| | |
|---|---|
| f1EXTD | FFFEA |
| f1FXEN | FFFF3 |
| f1INFR | FFFF5 |
| f1INX | FFFFC |
| f1IVL | FFFF8 |
| f1LC | FFFF1 |
| f1MKOF | FFFCE |
| f1NEGR | FFFF4 |
| f1NOFN | FFFD6 |
| f1NOPR | FFFE6 |
| f1NZ4 | FFFE8 |
| f1NZ5 | FFFCB |
| f1NZ6 | FFFCA |
| f1NZ7 | FFFC9 |
| f1NZ8 | FFFC8 |
| f1OVF | FFFFA |
| f1PDWN | FFFEB |
| f1PRGM | FFFC2 |
| f1PWDN | FFFCF |
| f1QIET | FFFFF |
| f1RAD | FFFF6 |
| f1RPTD | FFFC5 |
| f1RTN | FFFD4 |
| f1SCEN | FFFF2 |
| f1SUSP | FFFC1 |
| f1TNOF | FFFCD |
| f1UNF | FFFFB |
| f1USER | FFFF7 |
| f1USRX | FFFC6 |
| f1VIEW | FFFCC |
| k#-CHR | 00068 |
| k#-LIN | 0006B |
| k#1 | 00027 |
| k#2 | 00028 |
| k#3 | 00029 |
| k#ATTN | 0002B |
| k#BKSP | 00067 |
| k#BOT | 000A3 |
| k#CALC | 0006F |
| k#CONT | 00070 |
| k#CTRL | 0009E |
| k#DOWN | 00033 |
| k#EOL | 00026 |
| k#FLFT | 0009F |
| k#FRT | 000A0 |
| k#GON | 0009B |
| k#I/R | 00069 |
| k#LAST | 000A4 |
| k#LC | 0006A |
| k#LERR | 000A1 |
| k#LFT | 0002F |

| | |
|---|---|
| k#OFF | 00063 |
| k#RT | 00030 |
| k#RUN | 0002E |
| k#SST | 00066 |
| k#TOP | 000A2 |
| k#UP | 00032 |
| k#USER | 0006D |
| k#USEX | 000A5 |
| k#VIEW | 0006E |
| kc-CHR | 00000 |
| kc-LIN | 00004 |
| kcATTN | 0000E |
| kcBKSP | 00007 |
| kcBOT | 00015 |
| kcCALC | 00017 |
| kcCONT | 00010 |
| kcCTRL | 0000A |
| kcDOWN | 00013 |
| kcEOL | 0000D |
| kcFLFT | 00005 |
| kcFRT | 00006 |
| kcGON | 00016 |
| kcI/R | 00002 |
| kcLAST | 00019 |
| kcLC | 00001 |
| kcLERR | 0001A |
| kcLFT | 00008 |
| kcOFF | 00018 |
| kcRT | 00009 |
| kcRUN | 0000F |
| kcSST | 00011 |
| kcTOP | 00014 |
| kcUP | 00012 |
| kcUSER | 00003 |
| kcUSEX | 0000C |
| kcVIEW | 0000B |
| lACCSb | 00001 |
| lAp | 00010 |
| lBPOSp | 00005 |
| lCOPYb | 00001 |
| lCPOSb | 00006 |
| lD0p | 00005 |
| lD1p | 00005 |
| lDATEh | 00006 |
| lDBEGb | 0000B |
| lDEVC | 00005 |
| lDEVCb | 00001 |
| lDLENb | 00006 |
| lDp | 00010 |
| lEOL | 00002 |
| lFBEGb | 00006 |

| | |
|---|---|
| lFBF#b | 00003 |
| lFIB | 0003F |
| lFIL#b | 00002 |
| lFILBF | 00100 |
| lFILSV | 00032 |
| lFLAGh | 00002 |
| lFLENh | 00005 |
| lFNAM+ | 00004 |
| lFNAM8 | 00010 |
| lFNAMh | 00010 |
| lFSIZb | 00006 |
| lFTYPb | 00004 |
| lFTYPh | 00004 |
| lLXADR | 00005 |
| lLXENT | 0000B |
| lLXFAD | 00005 |
| lLXID | 00002 |
| lLXTKR | 00004 |
| lMSGp | 00004 |
| lPOL#p | 00005 |
| lPOLLp | 00005 |
| lPOLSV | 0003E |
| lPOLra | 00006 |
| lPROTb | 00001 |
| lREC#b | 00004 |
| lRECLb | 00004 |
| lRLENb | 00005 |
| lRTN1p | 00005 |
| lRTN2p | 00005 |
| lRTN3p | 00005 |
| lSHLNb | 00002 |
| lSPDTB | 0004E |
| lSPDn | 00001 |
| lSPDn2 | 00001 |
| lTEXTp | 00004 |
| lTIMEh | 00004 |
| o41sod | 00005 |
| oACCSb | 0000B |
| oAp | 0003E |
| oBNsod | 00011 |
| oBPOSp | 00005 |
| oBSsod | 00011 |
| oCOPYb | 0000A |
| oCPOSb | 00028 |
| oD0p | 00019 |
| oD1p | 0001E |
| oDATEh | 0001A |
| oDAsod | 0000D |
| oDBEGb | 00015 |
| oDEVCb | 0000C |
| oDLENb | 0002E |

| | |
|---|---|
| oDp | 0002E |
| oFBEGb | 0000D |
| oFBF#b | 00002 |
| oFIL#b | 00000 |
| oFLAGh | 00014 |
| oFLENh | 00020 |
| oFLSTr | 00031 |
| oFNAMh | 00000 |
| oFSIZb | 00039 |
| oFT-FL | 00010 |
| oFTYPb | 00005 |
| oFTYPh | 00010 |
| oIMPLh | 00025 |
| oKYsod | 00005 |
| oLXsod | 00005 |
| oMAINT | 0005D |
| oMSGPT | 00009 |
| oPOL#p | 0000A |
| oPROTb | 00009 |
| oREC#b | 00020 |
| oRECLb | 00024 |
| oRLENb | 00034 |
| oRTN1p | 0000A |
| oRTN2p | 0000F |
| oRTN3p | 00014 |
| oSHLNb | 00013 |
| oSPDTB | 00111 |
| oSPDn2 | 0000E |
| oSUBLn | 00025 |
| oTIMEh | 00016 |
| oTXsod | 00005 |
| pBSCen | 000F5 |
| pBSCex | 000F6 |
| pCALRS | 00036 |
| pCALSV | 00037 |
| pCAT | 00006 |
| pCAT$ | 00007 |
| pCLDST | 000FF |
| pCMPLX | 00038 |
| pCONFG | 000FB |
| pCOPYx | 00008 |
| pCRDAB | 00033 |
| pCREAT | 00009 |
| pCRT=8 | 00023 |
| pCURSR | 00029 |
| pDATLN | 0002A |
| pDEVCp | 00001 |
| pDIDST | 0000A |
| pDSWKY | 000FD |
| pDSWNK | 000FE |
| pEDIT | 0002B |

| | |
|---|---|
| pENTER | 00012 |
| pEOFIL | 00025 |
| pERROR | 000F2 |
| pExcpt | 000F8 |
| pFASCH | 0002C |
| pFILDC | 00002 |
| pFILXQ | 00003 |
| pFINDF | 00017 |
| pFNIN | 0003D |
| pFNOUT | 0003E |
| pFPROT | 0000B |
| pFSPCp | 00004 |
| pFSPCx | 00005 |
| pFTYPE | 0002D |
| pIMCHR | 0001E |
| pIMXCH | 0001F |
| pIMXQT | 0001D |
| pIMbck | 00020 |
| pIMcpi | 00021 |
| pIMcpw | 00022 |
| pKYDF | 0001B |
| pLIST | 0000C |
| pLIST2 | 0002E |
| pMEM | 000F1 |
| pMERGE | 0000D |
| pMNLP | 000FA |
| pMRGE2 | 0002F |
| pPARSE | 000F4 |
| pPRGPR | 00032 |
| pPRIN# | 00026 |
| pPRTCL | 0000E |
| pPRTIS | 0000F |
| pPURGE | 00010 |
| pPWROF | 000FC |
| pRCRD | 00034 |
| pRDCBF | 00018 |
| pRDNBF | 00019 |
| pREAD# | 00027 |
| pREN | 00039 |
| pRNAME | 00011 |
| pRTNTp | 0003A |
| pRUNft | 00030 |
| pRUNnB | 00031 |
| pSREC# | 00028 |
| pSREQ | 000F9 |
| pTEST | 000F0 |
| pTIMR# | 0003B |
| pTRANS | 000EF |
| pTRFMx | 0003C |
| pVER$ | 00000 |
| pWARN | 000F3 |

| | |
|---|---|
| pWCRD | 00035 |
| pWCRD8 | 00024 |
| pWRCBF | 0001A |
| pWTKY | 0001C |
| pZERPG | 000F7 |
| sARITM | 00007 |
| sBYEx | 00000 |
| sC/P | 00001 |
| sCARD | 00002 |
| sCARDC | 00008 |
| sCHAIN | 0000B |
| sCONT | 0000A |
| sCONTK | 00009 |
| sCURBT | 00003 |
| sCURUD | 00004 |
| sCURUP | 00002 |
| sCntg | 00002 |
| sCplxP | 00007 |
| sDEST | 00003 |
| sENDx | 00001 |
| sEOF | 00007 |
| sERROR | 00000 |
| sEXTDV | 00000 |
| sEXTGS | 00005 |
| sFOUND | 0000A |
| sGOSUB | 00003 |
| sI/OBF | 0000A |
| sINFRD | 0000A |
| sINX | 00005 |
| sIRAM | 00002 |
| sIX | 00007 |
| sInit | 00003 |
| sKEYS | 00005 |
| sMAINc | 00005 |
| sMULT | 00008 |
| sNEGRD | 0000B |
| sNoChn | 00002 |
| sONERR | 00004 |
| sONTMR | 00006 |
| sPCRD | 00008 |
| sPRGCF | 0000B |
| sRAD | 00009 |
| sRDX | 0000B |
| sREADI | 00004 |
| sRENAM | 00006 |
| sRENUM | 00008 |
| sRESTR | 0000A |
| sRETRN | 00000 |
| sRFILE | 00008 |
| sRUNBn | 00004 |
| sRUNDC | 00007 |

| | |
|---|---|
| sSIGN | 00009 |
| sSST | 00002 |
| sSSTdc | 00001 |
| sSTAT | 00006 |
| sSTOP | 00005 |
| sSpecl | 00006 |
| sUNDEF | 00001 |
| sXCPT | 00004 |
| sXQT | 00000 |
| sXWORD | 00009 |
| t! | 000FC |
| t% | 00085 |
| t& | 00089 |
| t* | 00083 |
| t+ | 00087 |
| t- | 00082 |
| t/ | 00084 |
| t@ | 000F4 |
| tABS | 000A2 |
| tACOS | 0009A |
| tADD | 000D5 |
| tADIG0 | 00060 |
| tADIG1 | 00061 |
| tADIG2 | 00062 |
| tADIG3 | 00063 |
| tADIG4 | 00064 |
| tADIG5 | 00065 |
| tADIG6 | 00066 |
| tADIG7 | 00067 |
| tADIG8 | 00068 |
| tADIG9 | 00069 |
| tALL | 000F8 |
| tAND | 0008B |
| tANGLE | 601B3 |
| tARRAY | 0007D |
| tASIN | 00099 |
| tATAN | 0009B |
| tAUTO | 000EE |
| tBASE | 000E9 |
| tBEEP | 000E8 |
| tBIG | 00010 |
| tCALL | 000F9 |
| tCARD | 000D0 |
| tCAT | 000EC |
| tCEIL | 00072 |
| tCFLAG | 000FA |
| tCHR$ | 000A4 |
| tCLOCK | 501EF |
| tCMPLX | 0007A |
| tCOLON | 000E2 |
| tCOMMA | 000F1 |

| | |
|---|---|
| tCOPY | 000B5 |
| tCOS | 00097 |
| tCVAL | 000E1 |
| tDATA | 000C6 |
| tDATE | 00077 |
| tDATE$ | 00078 |
| tDEF | 000B9 |
| tDEG | 0006F |
| tDEGRE | 000D3 |
| tDELAY | 000D6 |
| tDELET | 000B7 |
| tDIM | 000CC |
| tDISP | 000C5 |
| tDIV | 00086 |
| tDMYAR | 0007E |
| tDSTRY | 000BE |
| tDVZ | 000B1 |
| tEDIT | 000B8 |
| tELSE | 000F5 |
| tEND | 000DA |
| tENDDF | 000BA |
| tENDSB | 000C2 |
| tENTER | 4FFEF |
| tEOL | 000F0 |
| tEPS | 00071 |
| tERRL | 00075 |
| tERRN | 00076 |
| tERROR | 000E3 |
| tEXOR | 0008C |
| tEXP | 00094 |
| tEXTIF | 000F4 |
| tEXTND | 601EF |
| tFACT | 000A8 |
| tFETCH | 000C8 |
| tFFN | 000B4 |
| tFLOW | 901EF |
| tFLT1 | 0001D |
| tFLT10 | 00014 |
| tFLT11 | 00013 |
| tFLT12 | 00012 |
| tFLT2 | 0001C |
| tFLT3 | 0001B |
| tFLT4 | 0001A |
| tFLT5 | 00019 |
| tFLT6 | 00018 |
| tFLT7 | 00017 |
| tFLT8 | 00016 |
| tFLT9 | 00015 |
| tFN | 0007C |
| tFOR | 000C3 |
| tFP | 0006B |

| | |
|---|---|
| tGOSUB | 000DC |
| tGOTO | 000DD |
| tIF | 000DF |
| tIMAGE | 000FF |
| tIN | 000F2 |
| tINF | 00070 |
| tINPUT | 000C9 |
| tINT | 0009C |
| tINT10 | 00004 |
| tINT11 | 00003 |
| tINT12 | 00002 |
| tINT2 | 0000C |
| tINT3 | 0000B |
| tINT4 | 0000A |
| tINT5 | 00009 |
| tINT6 | 00008 |
| tINT7 | 00007 |
| tINT8 | 00006 |
| tINT9 | 00005 |
| tINTEG | 000CA |
| tINTO | E01EF |
| tINTR | 015FF |
| tINX | 000B2 |
| tIP | 0006A |
| tIS | 000E7 |
| tISUB$ | 000A7 |
| tIVL | 000AE |
| tKEY | 000E5 |
| tKEY$ | 00073 |
| tKEYS | 000CF |
| tLBLRF | 0000E |
| tLBLST | 000F6 |
| tLEN | 000A9 |
| tLET | 000C0 |
| tLINE# | 0000F |
| tLINPT | 000BF |
| tLIST | 000BB |
| tLITRL | 000C4 |
| tLN | 00091 |
| tLOG | 00090 |
| tLOG10 | 00093 |
| tLPRP | 000AA |
| tLR | 000B6 |
| tMAIN | 000D2 |
| tMATH | 601EF |
| tMAX | 000AD |
| tMAXRL | 0006C |
| tMEAN | 0009D |
| tMIN | 000AC |
| tMOD | 00074 |
| tNAME | 000BD |

| | |
|---|---|
| tNEAR | C01EF |
| tNEG | D01EF |
| tNEXT | 000C4 |
| tNOT | 00081 |
| tNUM | 000A3 |
| tOFF | 000E1 |
| tON | 000E0 |
| tOPT'N | 000ED |
| tOR | 0008D |
| tOVF | 000AF |
| tPAUSE | 000D7 |
| tPCRD | E01EF |
| tPI | 00079 |
| tPORT | 000D1 |
| tPOS | 201B3 |
| tPREDV | 0009F |
| tPRINT | 000CD |
| tPRMEN | 000F8 |
| tPRMST | 000F3 |
| tPURGE | 000EB |
| tRAD | 0006E |
| tRDIAN | 000D4 |
| tREAD | 000C7 |
| tREAL | 000BC |
| tRELOP | 0008A |
| tREM | 000E6 |
| tRES | 0007F |
| tRESTR | 000DE |
| tRETRN | 000DB |
| tRFILE | 000DE |
| tRMD | 0006D |
| tRND | 000A0 |
| tROUND | C01EF |
| tRUN | 000FE |
| tSDEV | 0009E |
| tSEMIC | 000F2 |
| tSFLAG | 000FB |
| tSGN | 000A1 |
| tSHORT | 000CB |
| tSIN | 00096 |
| tSMALL | 00011 |
| tSQR | 00092 |
| tSTAT | 000CE |
| tSTEP | 000F6 |
| tSTOP | 000D9 |
| tSTR$ | 000A6 |
| tSUB | 000C1 |
| tSVAR | 0002D |
| tTAB | 000F7 |
| tTAN | 00098 |
| tTHEN | 000F4 |

| | |
|---|---|
| tTIME | 0007B |
| tTIME$ | 00095 |
| tTIMER | 000E4 |
| tTO | 000F3 |
| tTRACE | 000EA |
| tUNF | 000B0 |
| tUPRC$ | 000AB |
| tUSER | 000E2 |
| tUSING | 000FD |
| tVAL | 000A5 |
| tVARS | B01EF |
| tWAIT | 000D8 |
| tXFN | 000B3 |
| tXWORD | 000EF |
| tZ | 0005A |
| tZERO | C01EF |
| t^ | 00080 |
| uALit | 000F7 |
| uCPLXC | 000EE |
| uDELIM | 000F4 |
| uHKB^ | 000F6 |
| uIMXCH | 000D4 |
| uIMbck | 000DC |
| uIMend | 000F0 |
| uIMsta | 000DE |
| uJMPdl | 000DB |
| uJMPst | 000DA |
| uJMP{} | 000D9 |
| uLOOPB | 000D2 |
| uLOOPP | 000EF |
| uLOOPS | 000D3 |
| uMODES | 0BDB1 |
| uMULT | 000D1 |
| uNUMEn | 000FC |
| uNUMEs | 000FD |
| uNUMFn | 000FA |
| uNUMFs | 000FB |
| uNUMNn | 000F8 |
| uNUMNs | 000F9 |
| uOPNM- | 000DF |
| uOPNNM | 000D8 |
| uOPNUM | 000E0 |
| uRESTP | 000F1 |
| uSTRPT | 000D0 |
| xANGLE | 00006 |
| xCLOCK | 00015 |
| xEXTND | 00026 |
| xFLOW | 00029 |
| xINTO | 0002E |
| xMATH | 00036 |
| xNEAR | 0003C |

| | |
|---|---|
| xNEG | 0003D |
| xPCRD | 0003E |
| xPOS | 00042 |
| xROUND | 0004C |
| xVARS | 0005B |
| xZERO | 0001C |

```
+---------------------------------------------+-----------------+
|                                             |                 |
|  ADDCAL  -  Address Calculation Utilities   |  CHAPTER  2     |
|                                             |                 |
+---------------------------------------------+-----------------+
```

## 2.1 XMTADR - Get XWORD Main Table Address

Category: ADDCAL     File: JP&EXC::MS

```
Name:(S) XMTADR  -  Get XWORD Main Table Address

Purpose:
     Find & Read XWORD MAINT Address

Entry:
     A(B)   = LEX ID
     A(2,3) = Entry #

Exit:
     Carry clear
        C    = MAINT address for XWORD
        B(A)= Relative Entry # for LEX ID with B(2-4) = 0
        A(B)= Actual Entry #

     Carry set
       LEX ID not found

     D1 preserved

Calls:       LXFND, RANGE

Uses.....
 Exclusive:  A(A),B(A),C(A),R1
             R1 = Preserved D1, RSTK holds LEX ID, Entry#

 Inclusive:  A(A),B(A),C(A),R1

Stk lvls:    1

Algorithm:

     Find Main Table Address for ROM ID
       Save LEX ID, Entry#  (B)
```

```
        Save D1                 (R1)
        Find LEX Table Buffer     (LXFND)
        If Buffer not found --> goto 1  (return, carry set)
        Save LEX ID, Entry#  (RSTK)
        Repeat until (LEX ID = 0)
           Read LEX ID in table
           If IDs match
              Pop Lex ID, Entry # off stack
              If Entry# within Range for LEX ID     (RANGE)
                 Shift Entry# to B(B), Zero B(XS) field
                 Compute Relative entry #
                 Read Main Table address --> C
                 Restore D1
                 RTNCC
              Restore LEX ID,Entry# to B(A)
           Skip to next entry
    0:  Pop LEX ID, Entry # off stack
    1:  Restore D1
        RTNSC (not found)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |
| 11/01/82 | JP | Interfaced to New Lex File format |
| 03/28/83 | JP | Save LexID, Entry # on Stack |
| 04/28/83 | JP | Restore Entry# to A(B) |

## 2.2 MTADDR - Calc Main Table Address for Token

Category: ADDCAL    File: JP&EXC::MS

Name:(S) MTADDR - Calc Main Table Address for Token
Name:(S) MTADR+ - Calc Main Table Address for Token

Purpose:
Calculates address of Main Table entry for token

Entry:
MTADDR: A(B) = Token to be looked up

Loads C with Mainframe MAINT

MTADR+: B(A) = Token to be looked up
C(A) = Main table address

Exit:
D1 contains main table entry address for token
C(A) contains value of D1 at time of call

Calls: None

Uses.......
Exclusive: B(B),C(A),A(A),D1
Inclusive: B(B),C(A),A(A),D1

Stk lvls: 0

Detail:
Multiplies token number by length of Main Table entry

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |

## 2.3 EXCADR - Compute Exec Addr of Token

Category: ADDCAL File: JP&EXC::MS

Name: EXCADR - Compute Exec Addr of Token
Name:(S) EXCAD+ - Compute Exec Addr of Token

Purpose: Return Execution Address of Command Token, preserving D0,D1

Entry: EXCADR: A(B) = Command token
Assumes MAIN Table in Mainframe
EXCAD+: A(B) = Command token
C(A) = Main Table + 3 of XROM

Position @ Execution Address field

Exit: C(A) = Execution Address for token

Calls: None

Stk lvls: 0

Uses: A(A),C(A)

Detail: Preserves D0
Address = Token * 9 + Main Table Adjustment

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | JP | Modified documentation |

```
+---------------------------------------------+------------------+
|                                             |                  |
|  BUFUTL  -  System Buffer Utilities         |  CHAPTER  3      |
|                                             |                  |
+---------------------------------------------+------------------+
```

## 3.1 IOFSCR - I/O Find for Available Scratch Buffer

Category: BUFUTL     File: SC&FIL::MS

```
Name:(S) IOFSCR  -  I/O Find for Available Scratch Buffer

Purpose:
     Returns available scratch buffer ID

Entry:
     P       =  0

Exit:
     P       =  0
     Carry clear => Available Buffer ID in C(X)
             set => No available scratch buffers
                    C(X)=000

Calls:      I/OFND

Uses.......
     A, C(A), D1

Stk lvls:   1


Detail:
     Scratch buffer ID's range from E00 (bSCRTC) to FFF

History:

  Date      Programmer           Modification
--------    ----------    ------------------------------------
02/08/83    S.W.          Added documentation
```

## 3.2 I/OFND - I/O Buffer Find

Category: BUFUTL File: SC&FIL::MS

```
Name:(S) I/OFND  -  I/O Buffer Find
Name:(S) IOFNDO  -  I/O Buffer Find

Purpose:    Find the specified I/O buffer

            IOFNDO looks for the buffer ID specified in C(X).

            I/OFND sets the high bit of the buffer ID
            specified in C(X), then looks for that buffer.
            (Buffer IDs with the high bit clear are those
            which will be deallocated at the next configura-
            tion).

Entry:    C(X)= Buffer ID#

Exit:   C(X)= Buffer ID#
          Carry set=> Match found
                      D1   points past buffer header
                      A(A) Buffer length field
                      C(S)=#addresses to update in buffer
          Carry clr=> No match

Calls:    none

Uses:     A, C(A), C(S), D1

Stack lvls: 0

Detail:   Buffer length field in header reflects the amount
          of available scratch space in that buffer, but is
          not the entire length of the buffer (eg doesn't
          include 7 nibbles for the header)

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/04/82    S.W.         Added documentation
 02/10/83    S.W.         Added 1 nibble to header front
 03/10/83    S.W.         Save Leeway setting in B(S)
 03/14/83    M.B.         Packed 3 nibs in I/OFN+
```

## 3.3 I/ORES - I/O Buffer Restore

Category: BUFUTL File: SC&FIL::MS

Name:(S) I/ORES - I/O Buffer Restore

Purpose: Sets high bit of buffer ID to preserve buffer

Entry: C(X) IS BUF ID#

Exit: CARRY SET=> BUFFER FOUND AND HIGH BIT OF ID# SET.
D1 POINTS PAST HEADER.
C(X) IS ID# WITH HIGH BIT SET.

CARRY CLR=> BUFFER NOT FOUND.

Calls: I/OFND

Uses: A, C, D1

Stack lvls: 1

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/04/82 | S.W. | Added documentation |

## 3.4 I/OCON - I/O Buffer Contract From Buffer End

Category: BUFUTL File: SC&FIL::MS

Name:(S) I/OCON - I/O Buffer Contract From Buffer End
Name: IOCNDO - I/O Buffer Contract From Buffer Middle

Purpose:
Contract an I/O buffer.

I/OCON contracts the buffer from its end, losing data stored at the end of the buffer.

IOCNDO contracts a specified section of the buffer.

Entry:
```
     C(X)   =  Buffer number
     B(A)   =  Amount to shrink existing buffer
                 A positive number - not to exceed 00FFF
     2 entry points:
            1) I/OCON - No additional requirements
            2) IOCNDO - D0 points to the beginning of the
               block that is to be deleted.
```

Exit:
```
     Carry clear=> Buffer not found
             set=> Buffer contracted specified amount
                   D1 points past buffer header
                   D0 points 1 nibble past front of header
                     (at buffer ID)
                   P=0
```

Calls: I/OFND, IDLNSV, MOVEMU, PTRADJ

Uses.......
Exclusive: A-D, D0, D1
Inclusive: A-D, D0, D1

Stk lvls: 3

Detail:
If amount to contract given in B(A) is greater than the current buffer size, the buffer is collapsed.
See I/OCOL

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | S.W. | Added documentation |
| 09/13/83 | S.W. | Modified doc. to show stk lvls=3 |

## 3.5 I/OCOL - I/O Buffer Collapse

Category: BUFUTL File: SC&FIL::MS

Name:(S) I/OCOL - I/O Buffer Collapse

Purpose:
Collapses specified I/O Buffer -
Leaves header intact, but shrinks length to zero

Entry:
C(X) = Buffer ID#

Exit:
```
Carry clear=> Buffer not found; Created w/zero length
        set=> Buffer collapsed
              D1 past buffer header
              P=0
              D0 1 nibble past buffer header
               (at buffer ID)
```

Calls: I/OFND, MOVEMU, PTRADJ

Uses.......
Inclusive: A-D, D0, D1

Stk lvls: 2

Detail: It is assumed that I/OCOL will only be called on existing buffers; if the buffer doesn't exist, 6 nibbles of user RAM will be utilized for the header w/o the leeway memory check.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/04/82 | S.W. | Added documentation |

## 3.6 I/OALL - I/O Buffer Allocate

Category: BUFUTL    File: SC&FIL::MS

```
Name:(S) I/OALL  -  I/O Buffer Allocate
Name:(S) I/OAL+  -  I/O Buffer Allocate

Purpose:    Allocates space for I/O buffer specified.
            If it already exists, will expand or contract to
            conform to size specified.  If it doesn't exist,
            will create it.

Entry:      C(X)=IDM
            B(A)= Desired buffer size (not to exceed FFF)

            I/OALL: Assumes P=0
                    Guarantess Leeway added in Mem Check
            I/OAL+: Sets P=1, guarantess NO Leeway in Mem Ck

Exit:      CARRY SET   => BUFFER ALLOCATED
                          D1 points past buffer header
                          D0 1 nib past buffer header front
                           (at buffer ID)
                          P=0
                          B(A) = buf size if just created,
                                 else net change in size
                          C(6-0) contains buf header info:
                                 C(0)   #addresses to update
                                 C(1-3) Buf ID
                                 C(4-6) Buf length
                          If Buffer already exists and expands
                             to a larger size:
                          A=D1 (past buffer header)
                          D(A) points to point of expansion
                              Buffer expanded from bottom


              CLR      => NO ROOM
                          C(4) = Error Number (eMEM)
                          P=0

Calls:     I/OFND, MOVEMD, MEMCL+, MOVEMU, IDLNSV

Uses:      A, B, C, D, D1, D0
           C(S) used to save Leeway setting for MEMCL+
```

Stack lvls: 2
3 - existing buffer decreases in size

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 07/04/82 | S.W. | Added documentation |
| 09/12/82 | J.P. | MEMCL+ interface, entries |
| 10/12/82 | S.W. | Eliminated I/OAL1 & I/OAL2 entry points. Changed I/OALL entry point to ASSUME P clear |
| 09/13/83 | S.W. | Modified stack level doc. |

## 3.7 I/OEXP - I/O Buffer Expand

Category: BUFUTL File: SC&FIL::MS

Name:(S) I/OEXP - I/O Buffer Expand
Name:(S) I/OEX2 - I/O Buffer Expand

Purpose:
Expand I/O buffer from high memory by the amount specified.

I/OEXP guarantees that the memory check is done including consideration for leeway.

I/OEX2 does the memory check without regard to leeway.

Entry:
C(X)= Buffer ID#
B(A)= Amount to expand buffer (in nibs)
Not to exceed 00FFF
2 Entry points:
1) I/OEXP - P=0
2) I/OEX2 - No additional requirements

Exit:
Carry clear=> Buffer not found
OR No room

```
                OR Buffer size requested too big
            set=> Buffer expanded
                  P=0
                  D1 points past buffer header
                  D0 points 1 nibble past buf header
                   (at buffer ID)
                  A(A)=D(A)= Point of expansion
                       (Old buffer end for I/OEXP)
```

Calls: I/OFND, IDLNSV, MEMCL+, MOVEMD

Uses: A-D, D1, D0
C(S) saves Leeway setting for MEMCL+

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | S.W. | Added documentation |
| 09/12/82 | J.P. | Added Leeway Check entries |

## 3.8 I/ODAL - I/O Buffer Deallocate

Category: BUFUTL     File: SC&FIL::MS

Name:(S) I/ODAL - I/O Buffer Deallocate

Purpose: Deallocates an I/O Buffer

Entry: C(X)=BUF ID#

```
Exit:    CARRY SET=> BUFFER DEALLOCATED
                     P=0
               CLR=> BUFFER NOT FOUND
```

Calls: I/OFND, MOVEMU, PTRADJ

Uses: A, B, C, D1, D0

Stack lvls: 2

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/04/82 | S.W. | Added documentation. Modified code to return with carry set if buffer deallocated. |

## 3.9 LXFND - Set D1 to LEX Table I/O Buffer

Category: BUFUTL File: TI&ERD::MS

Name:(S) LXFND - Set D1 to LEX Table I/O Buffer

Purpose:
Set D1 to LEX table I/O buffer.

Entry:
no necessary conditions.

Exit:
```
P      = 0
Carry set: buffer found.
  A(A)= buffer length
  D1 points past buffer header.
  C(S)=#addresses to update in buffer (?=0)
Carry clear: buffer not found.
```

Calls: I/OFND

Uses.......
 Exclusive: C(X), P
 Inclusive: A,C(A),C(S),D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 01/05/83 | MB | Documentation |

```
+-----------------------------------------------+------------------+
|                                               |                  |
|   CONFIG  -  System Configuration Utiltiies   |   CHAPTER  4     |
|                                               |                  |
+-----------------------------------------------+------------------+
```

## 4.1 ISRAM? - Pointing At RAM?

Category: CONFIG File: MN&CNF::MS

Name:(S) ISRAM? - Pointing At RAM?

Purpose:
Determine whether an address is in RAM or something else. This was put in to save writing to non-RAM devices, which for ROMs does no harm but for EEPROMs does plenty of harm.

Entry:
Address to check in C[A].

Exit:
P=0.
Carry set if address is in system RAM or IRAM.
Address passed is now in B[A].

Calls: CNFFND, MSIZ++

Uses.......
A,B[A],C,D1.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/09/82 | NM | Wrote. |

## 4.2 CONF - Configure Everything

Category: CONFIG File: MN&CNF::MS

Name:(S) CONF - Configure Everything

Purpose:
To configure all soft-configurable devices on the system bus.

Entry:

```
 CONF: S0=0 if requesting a power-up configuration
            (preserve integrity of system),
          1 if requesting a coldstart configuration (reset
            all pointers to coldstart values).
 CONFS3: S0 as above plus:
         S3 = 1 if we intentionally want configuration to
              behave as though ROM configuration changed.
```

Exit:
Configuration proper falls through to LEXBUF. S0 indicates whether a power-up configuration (S0=0) or a coldstart configuration (S0=1) was done.

Calls: AD1P,C=MAIN,C=RAME,CDIV10,CLKSPD,CLRXDS, CONFP4,CSLC3,CSLW4,CSLW5,CSRC3,CSRW3, D=AVME,DSLW-P,FNDBUB,INITPT,MODSIZ,MOVED2, MOVEUA,MOved3,MRKNEW,MRKOLD,MSIZE,Moveu3, R3<RST,RFADJ+,ROMTPT,RST<R3,SIZE10,SORT, SORTP2,STMBF?,TBLPT+,TBLPTR,UNCFG8,WAITKY, WHLTBL,csrw6.

Uses.......
A,B,C,D,D0,D1,P,R0-R4,Display buffer,S0-S3, RSTKBF.

Stk lvls: 3 (four are saved in RSTKBF)

NOTE:
The configuration code may decide on its own to perform a coldstart configuration when a power-up config was requested. This would be done if certain memory was corrupt, disallowing the manipulations necessary to maintain system integrity. In this case, the code will GOVLNG to COLDST (address #00000), which will wipe out the machine and call this code with S0=1.

If configuration code determines that ROM configuration has changed to a point endangering the validity of the umpteen pointers in the mainframe, it will essentially perform an "EDIT workfile" before falling into the LEXBUF code. It will also close all files in the FIB. This may be forced by entering at CONFS3 with S3=1.

If code detects the presence of too many ROMS to configure in the address space, it will give a warning message. It is not written to cover the contingency of too many RAMS or MMI/O devices, on the assumption that the possibility of said happening is too small to merit the immense code required.

Detail:

This code configures all soft-configurable devices on the system Bus. The code builds three tables in the configuration buffers: System RAM, Other memory (ROM, EEPROM, independent RAM, etc.), Memory-mapped I/O. The buffer IDs for the above configuration tables are, respectively, FF, FE, FD. The exact format of the information in the tables is explained below.

Following is the pre-configuration memory layout:

```
00000-1FFFF: Operating system
2C000-2C01F: Card reader
2E100-2E3FF: Display RAM
2F400-2FFFF: Disp Driver RAM
(FFC00-FFFFF: Reserved for config garbage collection)
```

The configuration code assigns addresses as follows:

Memory-mapped I/O upward from 20000-28000.

System RAM contiguously upward from 30000.
To achieve this contiguous mapping, system RAM is configured in reverse size order. This assures that 64 Knib RAMS are configured on 64 Knib boundaries, 32 Knib RAMS on 32 Knib boundaries, etc.

Uses S0-S3 internally as follows:
- S0: Set for coldstart, clear for power-up configure.
- S1: Used internally in debubbling system RAM, then used to indicate presence of ROMs for which there is no room to configure. Results in message.
  (Debubbling is explained in algorithm (below)

shortly below CON400 label.)
S2: Set to indicate failure of internal file chain verify. Results in message.
S3: Set to indicate that system ROM configuration has changed to an extent which may endanger the validity of some pointers. Results in collapsing of stacks and resetting pointers as though an "edit" command was entered.

To explain configuration, the following terms are used below:

PORT#: Physical port location (1-5) whose daisy chain is addressed by a bit (0-4) in output register. Port #0 is the internal daisy chain. Port #5 goes to the card reader slot.
DEV#: Position of a plug-in (0-15) in a daisy chain. Unless there is a port extender, all plug-ins will be device #0.
SEQUENCE: Consecutive chips in a module to be used as a single entity (e.g., a quad RAM which appears as one plug-in to the user).
DEVICE TYPE: Type of memory (RAM, ROM, etc., or memory-mapped I/O).
DEVICE CLASS: Identifies memory-mapped I/O device.

*** CHIP ID ***

The CHIP ID is a (usually) mask-programmed 20-bit pattern which is read by the CPU on an ID poll (C=ID instruction). A chip responds to the ID poll if two conditions are met:
1) The chip is unconfigured,
2) Daisy-in is high on the chip.
By examining the daisy chains one at a time, configuring each chip as we find it, we can locate and identify all soft-configurable chips on the bus.

The chip-id contains the following information:

NIBBLE 0: 15-Log2(size).

| Memory Size | Nib 0 | MM I/O space |
|---|---|---|
| 1 knib | F | 1 word (16 nibs) |
| 2 | E | 2 |
| 4 | D | 4 |
| 8 | C | 8 |
| 16 | B | 16 |
| 32 | A | 32 |
| 64 (max RAM) | 9 | 64 |
| 128 | 8 | 128 |

```
        256  (max memory)  7         256
                           6         512
                           5         1024
```

```
NIBBLE 1: (Reserved for future use)
          This nibble from the first chip in a
          sequence is stored in the configuration
          table for all sequences.

NIBBLE 2: Device type-- 0: RAM
                        1: ROM
                        2-E: assorted memory types
                        F: Memory-mapped I/O

NIBBLE 3: For memory, (unassigned).
          For memory-mapped I/O, contains device
            class--     0: HPIL mailbox
                        1-15: (unassigned)
          (Note: Card reader is hard configured
                 at 2C000-2C01F.)

NIBBLE 4: bits 0-1: (unassigned)
          bit 2: Last chip in sequence (see note (1)
                 below).
                 Always assumed high for MM I/O
                 devices, meaning all such devices
                 have their own table entry.
          bit 3: Last chip in module.
```

The top two bits (bits 2-3 of nibble 4) are used to determine what chips are in what physical plug-ins. Every sequence of chips (e.g., four identical RAMS in a RAM plug-in, an applications pack containing two ROMS, etc.) results in one entry in the configuration tables.

(1) End of sequence (but not module) is identified in one of two ways: 1) next chip returns ID with different value in nibs 0-3; 2) last chip of sequence has bit 18 set. The second approach is necessary if consecutive, identical chips are to be considered as different sequences, and will probably NEVER be used in the entire lifetime of the machine. But it can be done.

A module containing four 8-Kbit RAMS might return the following sequence of IDs:

    0000E  0000E  0000E  8000E

The resulting table entry would identify the chip size, chip count, device type, physical location, and configuration address of the device.

A module containing two 128-Kbit ROMS, a memory-mapped I/O interface using 2 words of address space, and four 16-Kbit RAMS might present the following sequence of IDs:

```
        0010A First ROM            \ one ROMtable entry
        0010A End of ROM sequence  /
        01F0E MM I/O devclass 1    one MM I/Otable entry
        0000D Start of RAMS        \
        0000D                      | one RAMtable entry
        0000D                      |
        8000D End of module        /
```

Restrictions: 16 chips/sequence
16 sequences/device
16 devices/port

Format of table entries:

```
        System RAM               Other Memory
        (cnftable ID FF)         (cnftable ID FE)
        ----------               ------------
NIB 0   Seq position             Seq position
NIB 1   Device #                 Device #
NIB 2   Port #                   Port #
NIB 3   15-Log2(size) **         15-Log2(size)
NIB 4 /                        /
NIB 5 | Address (kbit)         | Address (kbit)
NIB 6 \                        \
NIB 7   0                        Device type
NIB 8   #chips/plugin-1          #chips/plugin-1
NIB 9   Nibble 1 from ID         Nibble 1 from ID

        Memory-mapped I/O
        (cnftable ID FD)
        -----------------
NIB 0   Sequence position in dev
NIB 1   Device #
NIB 2   Port #
NIB 3   15-Log2(size)
NIB 4 /
NIB 5 | Address (words rel to 10000)
NIB 6 \
NIB 7   Device type  (always F)
NIB 8   Device class
NIB 9   Nibble 1 from ID
```

** FREEPORT routine may set this to zero to indicate that the RAM has been removed intentionally. This affects operation of this code in the spot where the old and new tables are compared to determine which

RAMs are new and which are missing.

```
Algorithm:
  CONF:
     S3=0 {to indicate we do not want EDITWF unless
       necessary}.
  CONFS3:
     Save 4 subroutine levels in RSTKBF.
  CONFRS:
     B=0000000000000001 {B contains device counters and
       other good things: B[B]=bit for output register,
       B[XS]=device#, B[3]=port#, B[S]=sequence#, B[6-5]=
       RAM counter, B[8-7]=ROM counter, B[10-9]=MMIO
       counter, B[12-11]=sum of other three counters,
       B[4]=(temporary storage of ID hinib).}
     D1=start of display buffer area where we build table.
     Perform a bus reset.
  IDLOOP:
     Is there room for any more entries?  If not then goto
       CONF10.
     Energize daisy chain for this port (OUT=B[B]/2).
     Get ID of next device on daisy chain (C=ID).
     If response#0 then goto IDLP20.
     Increment port# (B[3]).
     Reset device# (B[XS]).
     Reset sequence# (B[S]).
     Move port bit over one (B=B+B B).
     If port bit<=80H then goto IDLOOP else goto CONF10.
  IDLP20:
     Hold ID in R3.
     Hold ID hinibble in B[4].
     Build device table entry (except address) in C.
     If devicetype=RAM then goto IDLP90.
     If devicetype#Memory-mapped-I/O then goto IDLP60.
     Write memory-mapped-I/O table entry at D1.  Configure
       device to 40000H.
     P=(position of MMIO counter).
  IDLP30:
     If hibit of ID clear then goto IDLP40.
     Increment device#.
     Reset sequence#.
  IDLP40:
     Increment device counter pointed to by P.
     Increment total-#-devices counter.
     Goto IDLOOP.
  IDLP60:  {configuring "ROMs"}
     Set address field of table entry to FFF00.
     Find and configure all chips in this sequence to
       40000H (gosub CONFP4).
     P=(position of ROM counter).
     Goto IDLP30.
```

IDLP90: {configuring RAMs}
  Configure chip to 80000H.
  If first 8 nibbles of chip = IRAM ID then unconfigure chip and goto IDLP60 {if IRAM then treat as ROM}.
  Unconfigure chip.
  Find and configure all chips in this sequence to 400000H (gosub CONFP4).
  P=(position of RAM counter).
  Goto IDLP30.
CONF10: {Having identified everything plugged in...}
  Sort table by device type. {Sorting on WP, where P=7. Besides separating RAMs from ROMs from MMI/O, this will separate RAMs from IRAMs, since IRAMs were given an address (FFF00) while RAMs were not, and address serves as a secondary sort key.} This will arrange table into three pieces: RAM, ROM, MMIO.
  A=300H {starting address/100H of first RAM}.
CONF20: {assign addresses to RAM table entries}
  If there are no more system RAMs in table then goto CONF60.
  Write A[X] to address field of table entry.
  Increment A[X] by module size {module size=chipsize * #chips in module}.
  Goto CONF20.
CONF60:
  Save RAMEND {A[X]*100H} in R1.
  Point at ROM table. Sort it by size.
  Clear B for building allocation map {B will contain a bitmap of what pages --a page is 10000H nibbles-- are available for configuring ROMs}.
  If there is anything non-zero at E0000 {i.e., a hard-configured device} then B[15]=B[14]=F {mark those pages as unavailable}.
  Mark pages as unavailable which are occupied or partially occupied by operating system and system RAM.
CONF70: {loop to assign addresses to big ROMs}
  Any more ROMs in table? If not then goto CON170.
  If size of this entry < 1 page then goto PAKROM.
  Compute legal configuration boundaries and # pages needed for this ROM.
  Examine bitmap (starting at high end) for possible locations to configure this ROM.
  If possible location is found, write address to table entry. {Otherwise, table entry still contains FFC00 from ID loop}. Mark allocation map for space taken by this ROM.
  Goto CONF70.
PAKROM: {loop to assign addresses to small ROMs}
  Compute boundaries of one or (if available) two bubbles (blobs of unconfigured address space).

```
PAKR50:
   Examine ROM table entry;
     If ROM fits in either bubble, write address to
     table entry and reduce bubblesize appropriately.
   If there are more ROMs in list then goto PAKR50.
CON170: {now to configure memory-mapped I/O}
   A=0 X (address of MM I/O relative to 20000H).
CON180:
   If no more table entries then goto CON210.
   Write A[X] to table entry.
   Add device size to A[X].
   Goto CON180.
CON210:
   Sort entire table (RAMs, ROMs and MMI/O) by port-dev#.
   Perform bus reset.
CON220: {loop to configure all at assigned addresses}
   Any more table entries?  If not then goto CON270.
   Read table entry.
   Compute output register value for this port.  OUT=C.
   If not memory-mapped I/O then goto CON240.
   Compute configuration address (20000H + [addr]*10H)
     and issue CONFIG command at that address.
   Goto CON220.
CON240:
   Compute chipsize (from table entry) and configuration
     address ([table entry]*100H).
   Configure all chips in the sequence contiguously.  If
     address=FFF00, then do not increment address for each
     chip {this is rubbish plug-in, to be unconfigured
     soon; all chips goto FFF00}.
   Unconfigure everything at FFF00 {chips for which there
     wasn't room}.
   If R4[A] has been disturbed since we began {an
     interrupt occurred, and the output register may have
     been screwed} then goto CONFRS {start over}.
   Sort entire table by device type {separates system RAMs
     from "ROMs" from MMIO}.
   Sort RAM table by port-device# {for comparison with old
     table in configuration buffers}.
   {Time for the hard work.  If this is a coldstart we
     will initialize all system pointers.  If this is not
     we need to compare the old and new RAMtables and move
     memory to adjust for any modules which may have been
     added since the last configuration.}
   Was coldstart requested on entry (S0=1)?  If not then
     goto CON280.
coldst:  {here if config decides to coldstart}
   Was coldstart requested on entry?  If not then GOVLNG
     to 00000.
   Clear password.
   Initialize all pointers, filechain, command stack,
```

```
         variable chain heads, I/O buffers to coldstart
         values.
      Goto PUTBUF.
   CON280: {ready to incorporate new RAMs}
      Look for old RAMtable.  If not there then goto coldst.
   CON310: {start of loop to compare RAM tables}
      Anything more in newtable?  If not then goto CON380.
      Anything more in oldtable?  If not then goto CON390.
   CON330:
      Read two table entries.  If size, port-dev#, sequence#
         and chipcount the same then goto CON310.
      If newtable pdev# < oldtable pdev# {newtable has new
         device} then goto CON360.
      If newtable pdev# > oldtable pdev# {oldtable has
         missing device} then goto CON350.
      {Pdev#'s the same.  Something went away, something else
         appeared.}
      Mark newtable entry as new.
      Mark oldtable entry as missing.
      Goto CON310.
   CON350:
      Mark oldtable entry as missing.
      Goto CON310.
   CON360:
      Mark newtable entry as new.
      Increment newtable pointer.
      If more newtable entries goto CON330.
   CON370: {remaining oldtable entries missing}
      Mark oldtable entry as missing.
   CON380:
      Any more oldtable entries?  If yes then goto CON370
         else goto CON400.
   CON390: {remaining newtable entries new}
      Mark newtable entry as missing.
      Any more entries?  If yes then goto CON390.
   CON400:
      Read current values of AVMEMS and AVMEME.  Look at
         oldtable.  If any entries are marked as missing and
         were not entirely contained between AVMEMS and AVMEME
         then goto coldst.
      Compute nw AVMEME.  Store in R3.
      {Now comes the really hard part.  We will rearrange
         everything in memory to restore contiguity in light
         of any system RAMs which were added.}
      Sort RAMtable by address.
      Point past last entry in RAM table {we well read
         back from end of table}.
      {This is a debubbling process; that is, removing
         "bubbles" of new memory from existing memory.  This
         is done by creating a zero-length bubble at RAMEND.
         The bubble is then moved down through memory, passing
```

```
        RAMs which are not marked as new and expanding to
        contain RAMs which are marked as new.  This process
        continues until the bubble hits available memory,
        which is itself a bubble.}
      D0=new RAMEND {lowbound of bubble}.
      R3=new RAMEND {highbound of bubble}.
      D=old AVMEME {to know when we are done}.
      S1=0 {indicate that we are not almost done}.
   CON470:
      Any more table entries?  If yes then goto CON480.
      Dope up entry to look like built-in hard-configured
        RAM.
      S1=1 {indicate we are almost done}.
      Goto CON490.
   CON480:
      Read next table entry down.
      Marked as new?  If not then goto CON490.
      D0=D0-modulesize {expand bubble by changing lowbound}.
      Goto CON470.
   CON490:
      If bubblesize#0 then goto CON500.
      D0=D0-modulesize {move lowerbound of bubble}.
      R3=R3-modulesize {move upperbound of bubble}.
      If S1=1 {i.e., if we are almost done} then goto CON550
        else goto CON470.
   CON500:
      Move bubble down (i.e., move data up) size of module.
        If there is nothing to move (i.e., we have hit
        available memory) then goto CON550.
      If S1=0 goto CON470.
   CON550:
      {now that we have debubbled the stacks, it is time to
        debubble program memory.}
      R3=30000H {lowbound of bubble}.
      D0=30000H {highbound of bubble}.
      D=AVMEMS {to determine when we are done}.
   CON560: {start of loop}
      Any more table entries?  If not then goto CON650.
      Read next entry.  If not marked as new then goto
        CON580.
      Increase upperbound of bubble (D0) by size of this
        module.
      Goto CON560.
   CON580:
      Move bubble down past this module (i.e., move that
        amount of data down--to lower memory).
      If we are not done (bubble has not hit available
        memory) then goto CON560.
   CON650:
      Unmark all RAMtable entries which were marked as new.
      Update all pointers past AVMEME {since available memory
```

may have changed size).
Update variable chain heads.
Sort RAMtable in port-dev# order {since this will be the oldtable next time, it needs to be in this order).
Sort ROMtable in port-dev# order.
Look for old ROMtable. If not found then goto coldst.
Compare old and new ROMtables. If any old ROMs are missing or moved then S3=1 {indicate that we wish to force an edit-workfile to occur}.

PUTBUF:

Sort Memory-mapped I/O table by port-dev#.
Delete all table entries in all tables with an assigned address of FFF00 {these were not configured}. If any entries deleted, S1=1 {indicate that configuration error has occurred}.
{Now we will move tables from display buffer, where they were built, to configuration buffer area, where they will live, and will be known as oldtables on the next configuration.}
Compute size needed for configuration tables. Compute size taken by current configuration tables. Compute difference and move memory to make proper amount of room.
If there is insufficient memory to hold new tables, pinch off tables one entry at a time until there is room and indicate configuration error {S1=1}.
Move tables from display buffer to configuration buffer area.
Compute clockspeed and store in RAM (gosbvl CLKSPD).
Restore subroutine levels saved at beginning.
Fall through to LXBF++.
{ Configuration proper is done. The LXBF code will find and build tables of all lexfiles. It will also report configuration error if that was requested and perform an edit-workfile if that was requested. That could not be done at this point in the code because some polls are issued, and that cannot be done until we have a valid list of lexfiles.}

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 09/15/82 | NM | Added name to documentation |

## 4.3 CNFFND - Configuration Buffer Find

Category: CONFIG File: MN&CNF::MS

Name:(S) CNFFND - Configuration Buffer Find

Purpose: FINDS CONFIGURATION BUFFER

Entry: C(B) IS BUF ID#

```
Exit:   C(B)= BUFFER ID# (preserved from input)
          CARRY SET=> MATCH FOUND
                      D1   POINTS PAST BUFFER HEADER
                      A(A) BUFFER LENGTH
                      SB=0
          CARRY CLR=> NO MATCH
```

Calls: none

Stack lvls: 0

Uses: A(A), D1

Detail: Length given in header reflects the amount of scratch area in the buffer, but doesn't include the total buffer area (e.g. the 5 nibbles used by the header)

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 07/04/82 | SW | Added documentation |
| 02/11/83 | NM | Moved to CNF module |

## 4.4 LEXBUF - Set Up LEX Files Buffer

Category: CONFIG    File: MN&CNF::MS

```
Name:    LEXBUF  -  Set Up LEX Files Buffer
Name:(S) LEXBF+  -  Set Up LEX Files Buffer

Purpose:
     Set up Language Extension Files Table Buffer
     Must be called whenever Configuration or # Lex Files
     changes

Entry:
     LEXBUF:  At power on (through CONF)
              If coldstart
                 Statement Buffer created

     LEXBF+:  When Lex file copied into RAM
              Statement Buffer not created

Exit:
     Return after Fast POLL for Configuration

     If not enough memory to add all Lex files to Buffer
        Lex Buffer is collapsed down
        XROM01 and MAINT are added to Lex Buffer

Calls:      I/OAL+,LEXF00,LEXFND,ROMCHK,ROMFND,POLL,I/OCOL
            R<RSTK,RSTK<R

Uses:
 Exclusive: A,B,C,D1
            RSTKBF (3 levels)
              Needed for pCONF can be issued
 Inclusive: A,B,C,D,D0,D1,R1,R2,R3

            R1 = Pointer to next entry in ROM Config Table
               = Length remaining in ROM Config Table

Stk lvls:   4
           +4 levels saved in RSTKBF
            Allows LEXFND to use 4 lvls, also

NOTE:
     The Statement Buffer must be created FIRST in the
     I/O Buffer area.  Since the LEX Buffer size can
```

change between Power ON and the Statement Buffer may be in use, updating PCADDR that points into the Statement Buffer would be near IMPOSSIBLE, since an offset is not easy to calculate.

Algorithm:

```
LEXBUF: If Coldstart    (S0=1)
            Create Statement Buffer          (I/OAL+)
LEXBF+  Allocate Language Extension Buffer (I/OAL+)
            ID=FC, Size=0
        Save 4 stack levels                  (R<RSTK)
        Search for LEX files in RAM          (LEXFND)
        Check if ROM Table is non-empty      (ROMCHK)
        If ROM Config Table NOT empty
            Search ROM for LEX files & Update LEX Table
            Repeat until (End of ROM Table)
              Find next ROM  (ROMFND)
              Search ROM & Update LEX Table (LEXFND)
              If not enough memory to Expand (Carry Clear)
     1:          Collaspe Lex Buffer          (I/OCOL)
                 Set C(S) so I/OEX1 will not use Leeway
                 goto 2;
        Set C(S) so I/OEX1 will use Leeway
     2: Add Built-in XROM, MAINT to LEX Table Buffer
            Set R3 @ "00" byte to indicate end of file
            Set D0 @ start of XROM01
            Add xrom01 and MAINT to LEX Buffer (LEXF00)
            If not enough memory to add --> goto 1;
        CONFIGURATION Poll.
        Restore return levels to stack       (RSTK<R)
        If handled, restart CONFIGURATION from the
          beginning.
        else
          go Auto delete I/O buffers
```

Detail:

xrom01 and MAINT Lex Files are CHAINED together. The next Lex File relative address pointer within xrom01 points to the start of MAINT. One call to LEXF00 will add both xrom01 and MAINT to the Lex Buffer.

4 stack levels are saved to fixed TWO problems:

Within LEXFND (called by LEXBUF)
  Usage is 4 levels  (Stack save, I/OEX2 (uses2)
  One level too deep ---

pCONF issued at end of LEXBUF
  Since FPOLL uses 2 levels to get there

```
        No levels left for HPIL/Lex file to deal with
           its buffers

      COPY
        COPYu
          RSTK <-- R1
            LEXBUF
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/09/82 | JP | Modified documentation |
| 09/09/82 | JP | Add no memory to expand handling |
| 11/01/82 | JP | Added New Lex file format |
| 11/04/82 | JP | Calling LEXFOO to add xrom01/MAINT |
| 01/03/83 | JP | Removed S9 usage |
| 03/09/83 | JP | Changed STMBID to bSTMT |
| 07/05/83 | JP | Save 3 levels in RSTKBF |
| 07/05/83 | JP | Adjusted documentation |
| 07/22/83 | NM | Moved configuration excpt handling |
| 09/13/83 | JP | Updated documentation:<br>4 stack levels used;4 saved |

## 4.5 KYDN? - Is a Key Down in Current Row?

Category: CONFIG File: SB&DVR::MS

Name:(S) KYDN? - Is a Key Down in Current Row?

Purpose:
Determine if a key is down which could cause a problem for configuration.

Entry:

Exit:
Carry clear if a key is down in the currently energized row(s), else carry clear.

Calls: None.

Uses.......
None.

Stk lvls: 1

NOTE:

A brief description of the problem: If 2 or more keys are down in a column, and a row containing one of the keys is energized, the multiple keys short the rows together, resulting in energizing multiple rows. In configuration, this amounts to addressing more than one port daisy chain at once, which can lead to disaster. This routine is called at appropriate times to ensure that no keys are down that can screw up configuration.

Detail:

Preserves all registers at the expense of a subroutine level.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/16/82 | NM | Added documentation |

```
+----------------------------------------------+------------------+
|                                              |                  |
|   CONVRT  -  Conversion Utilities            |   CHAPTER  5     |
|                                              |                  |
+----------------------------------------------+------------------+
```

## 5.1 FLTDH - Convert 12-digit Flt To Hex Integer

**Category:** CONVRT **File:** AB&UTL::MS

Name:(S) FLTDH - Convert 12-digit Flt To Hex Integer
Name:(S) DCHXF - Convert 12-digit Flt To Hex Integer

Purpose:
    Convert a 12-digit floating-point number to a 5-digit
    hex integer.

Entry:
    A=12-digit floating-point number.
    (FLTDH and DCHXF are two names for same entry point.)

Exit:
    P=0.
    A[A] = hex integer.
    Carry set if number is positive and in range.
    Carry clear ->
      If XM=1, number is out of range (returns FFFFF).
        (NaN is considered out-of-range.)
      If XM=0, number is negative (returns result in 2's
        complement).
    Also B[S]#0 iff number is negative.
    HEX mode.

Calls: OVFLOW.

Uses.......
    A,B,C,P,XM.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|

| | | |
|---|---|---|
| | SA | Wrote |
| 12/20/82 | SW | Added info about B[S] to doc hdr |
| 10/18/83 | NM | Attempted to document |

## 5.2 DECHEX - Convert DEC Integer To HEX Integer

Category: CONVRT File: AB&UTL::MS

Name:(S) DECHEX - Convert DEC Integer To HEX Integer
Name:(S) DCHX=C - Convert DEC Integer To HEX Integer

Purpose:
Convert decimal integer to hex integer.

Entry:
DECHEX: A[W] = decimal integer.
DCHX=C: C[W] = decimal integer.

Exit:
P=0.
A[A] = hex integer.
HEX mode.
Carry set -> result is good.
Carry clear -> overflow.
XM = not carry.

Calls: None.

Uses.......
A,B,C,P,XM.

Stk lvls: 1.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 5.3 HDFLT - Convert HEX Integer To DEC Flt-pt

Category: CONVRT File: AB&UTL::MS

**Name:(S) HDFLT - Convert HEX Integer To DEC Flt-pt**

**Purpose:**
Convert hex integer to 12-digit decimal floating-point number.

**Entry:**
A[A] = hex integer.

**Exit:**
P=0.
A=12-digit floating-point number.
Carry set.
DEC mode.

**Calls:** HEXDEC.

**Uses.......**
A,B,C,P.

**Stk lvls:** 1

**History:**

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/15/82 | SA | Changed to NM's conversion |
| 10/18/83 | NM | Attempted to document |

## 5.4 FLOAT - Convert Dec Integer Into 12-Dig Float

Category: CONVRT    File: AB&UTL::MS

```
Name:(S) FLOAT  - Convert Dec Integer Into 12-Dig Float

Purpose:
     Convert right-justified decimal integer into floating
     point number.

Entry:
     Argument in A[W] (unsigned).
     Maximum 999999999999 (1e12-1).

Exit:
     Floating-point number in A[W].
     DEC mode.
     Carry set.

Calls:      None.

Uses.......
            A[W], P.

Stk lvls:   0

Algorithm:
     Return if A=0.
     ASL 3 times, A[X]=011.
     While A[14]=0 do
       begin
         ASL M    {loop to align mantissa}
         A=A-1 X
       end.

History:

   Date     Programmer              Modification
 --------   ----------   -----------------------------------
            SA           Wrote
 06/11/82   NM           Attempted to document
```

## 5.5 HEXASC - Convert Hexadecimal to Ascii

Category: CONVRT File: FH&TFM::MS

Name:(S) HEXASC - Convert Hexadecimal to Ascii

Purpose: Converts specified number of hex digits to ASCII and returns the string (backwards) in A(W), B(W)

Entry:
```
     A(W)   =  Hex digits
     C(S)   =  #nibs-1 to convert; must be 7 or less
     P      =  0
```

Exit:
```
     A(W)   =  Converted string (high digit in low memory)
     B(W)   =  Converted string (high digit in low memory)
     C(S)   =  F
     P      =  0
     Carry  =  Set
```

Calls: none

Stack lvls: 0

Uses: A, B, C(S), C(B)

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | SW | Added documentation |

## 5.6 CNV2UC - Converts 8 chars to uppercase

Category: CONVRT File: JP&PR3::MS

```
Name:     CNV2UC  -  Converts 8 chars to uppercase
Name:(S)  CNVWUC  -  Converts 8 chars to uppercase
Name:(S)  CVUCW   -  Converts 8 chars to uppercase
```

**Purpose:**

Converts 8 lowercase characters to uppercase.

Lowercase characters are converted to uppercase by clearing bit 5 of the ASCII code. All characters with character codes from 60-7F HEX get bit 5 cleared. This results in ensuring that digits, uppercase letters, and most special characters are left unchanged. However, any character within the range of 60-7F that is not a lowercase letter WILL have its character code altered.

**Entry:**

```
3 entry points:
 1) CNV2UC - D1 at possible preceding blanks before
             characters to convert.
 2) CNVWUC - D1 at 1st character to convert.
             P=0.
 3) CVUCW  - A contains characters to convert.
              (it may contain any no. of characters).
             P=0.
```

**Exit:**

```
P=0
Carry clear
Every byte in A has bit 5 zeroed.
CNV2UC:
 D1 points at the first non-blank character
 A contains the following eight bytes with
  bit 5 zeroed in every byte.
CNVWUC:
 Same as CNV2UC, except D1 is preserved from entry.
CVUCW:
 D1 preserved from entry
```

**Calls:** GNXTCR, BLANKC

```
Uses:       A,C, D1   -  CNV2UC entry
            A,C       -  CNVWUC, CVUCW entries
```

**Stk lvls:** 1

**NOTE:**

only works if characters are upper- or lower-case chars to begin with

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | S.W. | Added documentation |

## 5.7 CONVUC - Convert To Upper Case

Category: CONVRT File: MN&ED::MS

Name:(S) CONVUC - Convert To Upper Case
Name:(S) CNVUCR - Convert To Upper Case

Purpose:
  Convert char in A(B) to upper case if lower case
  Read a byte into a first if CNVUCR entry point used.

Entry:
  A(B) = Character to be converted
  P = 0
  HEX mode.

Exit:
  Carry set if no conversion required
  A(B)=converted letter, not changed if carry set at exit.
  P = 0

Calls: Range

Uses.......
 Exclusive: C(3-0),A(B)
 Inclusive: C(A),A(B)

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/16/82 | BS | Updated documentation |

## 5.8 RJUST - Unfloat A Floating-Point Number

Category: CONVRT File: MN&TM::MS

Name:(S) RJUST - Unfloat A Floating-Point Number

Purpose:
Unfloat a 12-digit form floating-point number.

Entry:
12-digit floating-point number in A
(sign ignored).

Exit:
Error exit (Inv Arg) if NaN passed.
A[W] = Right-justified decimal integer version of argument.
Carry set: Arg was infinity; result=1E16 - 1.
Carry clear: Arg was finite; arg >= 1E16 returned as 1E16 - 1.
DEC mode.
P=0.

Calls: None.

Uses.......
A,C,P.

Stk lvls: 0

Detail:

| | | | |
|---|---|---|---|
| Input: | 0123000000000002 | Output: | 0000000000000123 |
| Input: | 0123500000000002 | Output: | 0000000000000124 |
| Input: | 0123456789870007 | Output: | 0000000012345679 |
| Input: | 0987000000000998 | Output: | 0000000000000000 |
| Input: | 0987000000000050 | Output: | 9999999999999999 |

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/18/82 | NM | Added documentation |

## 5.9 HXDCW - Hex To Decimal Conversion

Category: CONVRT File: MN&UTL::MS

Name:(S) HXDCW - Hex To Decimal Conversion
Name:(S) HEXDEC - Hex To Decimal Conversion

Purpose:
Convert a full-word HEX# or an A-field HEX # to a DEC#.

Entry:
HEXDEC: Argument in A[A].
HXDCW: Argument in C[W] (HEX).
Mode doesn't matter.

Exit:
Result in A,B,C (DEC).
DEC mode.
Carry clear.
P unaffected.

Calls: MPY (falls through)

Uses.......
A,B,C

Stk lvls: 0

Algorithm:
HEXDEC: C=0 W, C=A A
HXDCW: A=0000000000000001
SETDEC
Fall through to MPY for mixed-mode multiply.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/03/82 | NM | Added documentation |
| 10/15/82 | SA | Added HEXDEC entry point |

## 5.10 DCHXW - Full Word Decimal To Hex Conversion

Category: CONVRT File: MN&UTL::MS

Name:(S) DCHXW - Full Word Decimal To Hex Conversion

Purpose:
Convert full-word DEC to full-word HEX number.

Entry:
Argument in C.
Mode doesn't matter.

Exit:
Result in A, B and C.
HEX mode.
Carry clear.
P=0.

Calls: None.

Uses.......
A,B,C,P.

Stk lvls: 0

Algorithm:
```
Clear register for result.
For q=15 downto 0 do
  begin
    Multiply result by 10.
    Add digit Nq of argument to result
  end.
```

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/03/82 | NM | Added documentation |

## 5.11 VARNBR - Pop and Test Variable Number

Category: CONVRT File: PM&STA::MS

Name:(S) VARNBR - Pop and Test Variable Number
Name:(S) VARNB- - Pop and Test Variable Number

Purpose:
Rounds decimal floating point real value on top of math stack to a hex integer, then tests for a valid variable number.
A NaN input will fall through; an out-of-range input will create a NaN -- both set carry.

Entry:
decimal value to be converted on top of math stack
D1 ----- points to top of math stack
R2(S) -- N statistical variables

Exit:
```
     Carry=Set:   invalid input, NaN output in registers A/B
                  XM=1: If NaN created
     Carry=Clear: A(A) -- rounded hex integer
                  XM=0
                  HEXMODE
                  P=0
```

Calls: DCHXF,IVARG,POP1R,SPLTAX,finita

Uses.......
Inclusive: VARNB-: A,B,C,P,XM
VARNBR: same, unless fatal error

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/09/82 | PM | Documented routine |
| 08/12/82 | " | Changed entry points |
| 12/14/82 | " | Added signaling NaN test |
| 02/10/83 | " | Fixed neg var nbr problem |

## 5.12 STR$SB - Convert Number to String

Category: CONVRT File: SB&IO::MS

Name:(S) STR$SB - Convert Number to String
Name:(S) STR$00 - Convert Number to String(Generic)

Purpose:
Pops a number off stack and pushes a string on stack containing ASCII representation in current display setting.
STR$SB is a subroutine which returns a string without leading and trailing blanks surrounding the number.
STR$00 is a generic routine which will either return when done or jump to EXPR. It may or may not output leading and trailing blanks.

Entry:
P = 0
D1 points to top of stack

STR$00:
Return (S0) set iff return is desired otherwise jumps to EXPR when done.
Blanks (S1) set iff leading and trailing blanks are desired.

Exit:
P = 0
D1 points to string
Exits to MEMERR if memory overflows

Calls: POP1N,FMTNUM,STKCHR,NAN?,FMTPRP,ADHEAD,D=AVMS, DSFORM

Uses.......
Exclusive: D1,S0,S1,C(A),D(A)
Inclusive: A,B,C,D(A),S0,R0,R1,R2

Stk lvls: 2

Detail:
Pops an numeric item off expression stack and

checks the current display format.

Standard format: If the number can be represented without losing accuracy in 12 digits plus optionally a decimal point it will be, else scientific notation will be used and all significant digits will be shown.

FIX n: Display n places past the decimal point with rounding. If result is longer than 13 digits, defaults to SCI n.

SCI n: Display n+1 significant digits in scientific notation with rounding.
(1. <= mantissa <= 9.999999...)

ENG n: Display n+1 significant digits in engineering notation with rounding.
(1. <= mantissa <= 999.9999...;
exponent divisible by 3)

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/20/82 | B.S. | Updated documentation |

```
+-----------------------------------------+------------------+
|                                         |                  |
|  DSPUTL  -  Display Utilities           |  CHAPTER  6      |
|                                         |                  |
+-----------------------------------------+------------------+
```

## 6.1 NOSCRL - Request No-display-scrolling

Category: DSPUTL    File: MN&ED::MS

Name:(S) NOSCRL - Request No-display-scrolling

Purpose:
    Request that main loop bypass scrolling of current display contents.

Entry:
    None.

Exit:
    C[A]=0.
    D0=NEEDSC.

Calls: None.

Uses.......
    C[A],D0.

Stk lvls: 0

Detail:
    Clears (NEEDSC). This prevents main loop from calling SCRLLR so user can stare at display.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/31/83 | NM | Added documentation |

## 6.2 VIEWD1 - View A Buffer While Keys Down

Category: DSPUTL    File: MN&ED::MS

**Name:(S) VIEWD1 - View A Buffer While Keys Down**

**Purpose:**
This entry point takes a 22 character buffer pointed to by D1 and builds a bit pattern in display inside the WINDOW setting. This display is held until all keys are up.

**Entry:**
P = 0
D1 points at a 22 character buffer.

**Exit:**
P = 0

**Calls:** BLDBIT

**Uses.......**
A,B,C,D,D0,D1

**Stk lvls:** 2

**Detail:**
This routine looks at the current WINDOW settings to set up the first character position and the number of characters to be displayed. Since this may be (and usually is) 22 characters, the buffer to be viewed should be at least 22 characters. It should be padded with either blanks or nulls to prevent unwanted "junk" at the end of the display.

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | BS | Updated documentation |

## 6.3 CURSFR - Move Cursor To Far Right

Category: DSPUTL File: MN&ED::MS

Name:(S) CURSFR - Move Cursor To Far Right

Purpose:
Send CURSOR FAR RIGHT to display.

Entry:
P = 0
HEX mode.

Exit:
P = 0
Carry clear

Calls: ESCSEQ (falls through)

Uses.......
A,B,C,D,D0,D1.

Stk lvls: 4

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/15/82 | BS | Added documentation |

## 6.4 CURSFL - Move Cursor To Far Left

Category: DSPUTL File: MN&ED::MS

Name:(S) CURSFL - Move Cursor To Far Left

Purpose:
    Send CURSOR FAR LEFT to display.

Entry:
    P = 0

Exit:
    P = 0
    Carry clear

Calls: ESCSEQ (falls through)

Uses.......
    A,B,C,D,D0,D1.

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | BS | Added documentation |
| 11/04/82 | NM | Packed a little. |
| 12/09/82 | NM | Packed a lot. |

## 6.6 SETFMT - Set Display Format

Category: DSPUTL    File: MN&UTL::MS

Name:(S) SETFMT - Set Display Format

Purpose:
    Set FIX, SCI, ENG or STD display format.

Entry:
    C[0] = C for STD, D for FIX, E for SCI, F for ENG.

Exit:
    Carry clear.

Calls: None.

Uses.......
 Inclusive: A[A],C[A],D[A]

Stk lvls: 0

Algorithm:
Read DSPFMT nibble from system flags.
Set lower 2 bits thereof.
AND with argument passed in C[0].
Write out DSPFMT nibble.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/26/82 | NM | Wrote. |

## 6.6 UPDANN - Update Annunciator

Category: DSPUTL File: PM&FLG::MS

Name:(S) UPDANN - Update Annunciator
Name: UPDANX - Update Annunciator

Purpose:
Updates annunciators corresponding to user and system flags.

Entry:
user and system flags
HEXMODE

Exit:
appropriate annunciator(s) turned on/off
Carry=Clear
P=0
HEXMODE

Calls: DBLUP,SNGLUP,UPDAN1

Uses.......

Inclusive: CPU: A(B),B(A),C(A),D0,P
RAM: ANNAD1-4

Stk lvls: 1

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/11/82 | PM | Documented routine |
| 10/04/82 | PM | Changed for annunciator revision |
| 01/05/83 | PM | Revised documentation |

## 6.7 ASCII - ASCII Bit Pattern Tables

Category: DSPUTL File: SB&BIT::MS

Name:(S) ASCII - ASCII Bit Pattern Tables

Purpose:
Bit patterns for built in character set.

Detail:
The bit pattern for each character requires 10 nibbles. Each of the 5 pairs of nibbles defines one display column. Each column has 8 bits where the lsb of the byte is the top row and the msb is the bottom row. The bit pattern for an ASCII char may be found by reading 10 nibbles at the address ASCII + 10 * (Char#).

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/29/83 | B.S. | Updated documentation |

## 6.8 CMDPR" - Text for command stack prompt

Category: DSPUTL     File: SB&CMD::MS

Name:(S) CMDPR" - Text for command stack prompt

Purpose:
This is the text for the command stack prompt, it is the following sequence: CR, LF, cursor off, backslash, cursor on. The text string is terminated by a FF byte as expected by BF2DSP.

Entry:
Don't enter

Exit:

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | B.S. | Added documentation |

## 6.9 MAKEBF - Make ASCII Buffer from Display Buffer

Category: DSPUTL     File: SB&CMD::MS

Name:(S) MAKEBF - Make ASCII Buffer from Display Buffer

Purpose:
Builds an ASCII buffer containing all readable characters in the display and appends it to the command stack (between CLCBFR and RAWBFR).

Entry:

Exit:
    P = 0
    C(A) points at first char of text
    D0 points past text
    A(A)=Buffer length + 3 nibbles

Calls: OUT1TK,OUTBYT,OUTNBC,INITPT,STKCMD,CHKSPC,MOVEU2,
D=AVME

Uses.......
 Exclusive: D0,D1,A,B,C,D(A)
 Inclusive: D0,D1,A,B,C,D(A)

Stk lvls: 2

Detail:
D0 is initialized to contents of RAWBFR, a 3 nibble length field is output, then for each readable character in display, a byte is added to the buffer by calling OUT1TK. After buffer is built, a CR is written to the end of the buffer. STKCMD is called to edit the command stack. Pointers from RFNBFR to AVMEMS are updated to point to new end of buffer. If there is less than LEEWAY memory left, commands in the command are crushed, starting with the oldest, until LEEWAY available memory exists or all but the most recent command have been crushed.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/19/82 | B.S. | Updated documentation |

## 6.10 BLDDSP - Build Display Pattern from Buffer

Category: DSPUTL File: SB&DSP::MS

Name:(S) BLDDSP - Build Display Pattern from Buffer
Name:(S) BLDLCD - BLDDSP Except Display Status Active

Purpose:
Uses the display buffer and related status information to build the display bit pattern.

Entry:
Hexmode

Exit:
P = 0
Hexmode

Calls: GETSTA,D0=FC,FCALC?,D10=FC,BLDBIT,BLDB40,
WRITM1,SETSTA

Uses.......
Inclusive: A(W),B(W),C(W),D(W),D0,D1

Stk lvls: 2

Algorithm:
If cursor is on then adjust FIRSTC so that cursor will be in display window.
Turns left arrow annunciator on or off depending on whether FIRSTC is zero or not.
Sets up registers and calls BLDBIT to build display.
Turns on right arrow annunciator iff display buffer contains characters to the right of last character in the currently displayed window.
If cursor is on then sets the cursor phase so the cursor will appear "on" first and falls into code for display update (ie cursor blink).
If cursor off then disables display timer and returns.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/19/82 | B.S. | Updated documentation |

## 6.11 BLDBIT - Build Bit Patterns in Display

Category: DSPUTL File: SB&DSP::MS

Name:(S) BLDBIT - Build Bit Patterns in Display

Purpose:
    Used to put a given number of character's bit patterns
    in display given an arbitrary ASCII buffer.

Entry:
    P = 0
    D(A)=Display starting position (ie WINDST)
    D(14,15)=Number of positions to display minus 1
    C(A) points to buffer of characters

Exit:
    P = 0

Calls: IOFNDO

Uses.......
 Inclusive: A(W),B(W),C(W),D(W),D0,D1

Stk lvls: 1

Algorithm:
```
For each character to be displayed
   If the high bit is on then
      Look for an alternate charset buffer.
      If one is found then
         Check for indirect character set and
            change pointers if found
         Multiply character number by 12
         If this number is less than the length
               of the charset buffer then use that
               buffer
         else use the default bit pattern table
      else use the default bit pattern table
   else use the default pattern table
   If using the default bit pattern table
       then multiply the character number by 10
   Add the offset (char number times 10 or 12)
         to the start of the table being used
         and read in bit pattern.
   Read 3 nibble table entry from LCDTAB
   Double table entry to set carry if this
       char crosses a display driver boundary
       and to generate the lower 3 nibbles
       of the starting address of this display
       position.
   Write out bit pattern to display.
   If display driver boundary is crossed then
       shift the bit pattern 4 columns and
```

move to next display driver and write
out remainder of character.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/19/82 | B.S. | Added documentation |

## 6.12 DSPUPD - Display Update

Category: DSPUTL File: SB&DSP::MS

Name:(S) DSPUPD - Display Update

Purpose:
Process service request for display code.
Service request can be generated by TIMER1 and is used
either for:
1) Cursor blink, or
2) End of display delay.

Entry:
P=0.

Exit:
P=0.

Calls: GETSTA,D1=FC,BLDBIT,BLDB40,WRTTM1,SETSTA

Uses.......
Inclusive: A(W),B(W),C(W),D(W),D0,D1,RAM(DSPSTA)

Stk lvls: 2

NOTE:
Saves contents of ST on entry into DSPSTA. Restores
on exit.

Algorithm:
Stores callers status bits in DSPSTA and recalls

display status.
Sets TimOut bit to indicate timer has timed out.
If UpdOff then display doesn't need updating so set timer to a long time and return.
If CurOff then cursor is off and thus display doesn't need updating so set timer to a long time. TimOut was set above which notes the fact that the timer has timed out. This is used for display delay during line feed.
If BitsOK then the LCD reflects the display buffer and doesn't require rebuilding just to change cursor. If not BitsOK then we need to rebuild the LCD to make sure cursor will make sense, this code will fall back through DSPUPD once display has been updated.
Now need to change cursor.
The position of the cursor in the display is calculated by looking at CURSOR, WINDST and WINDLN.
If the cursor isn't in display then set the timer to a long time and return.
Depending on the Phase, either
* Rebuild the character that belongs in cursor position and toggle phase.
* Check if replace or insert cursor is required, build it in display, toggle phase and return.

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/25/83 | NM | Added documentation |
| 06/07/83 | B.S. | Enhanced documentation |

## 6.13 GETMSK - Get Mask for Character Protection Bitmap

Category: DSPUTL File: SB&DSP::MS

Name:(S) GETMSK - Get Mask for Character Protection Bitmap

Purpose:
Point at location if protection bitmap and return a

mask for isolating bit corresponding to current cursor position.

Entry:
P=0.

Exit:
P=0.
B[0]=C[0]=Mask nibble.
D0 points at nibble in bitmap for current cursor position. Mask can be used to isolate proper bit.

Calls: None.

Uses.......
B[A], C, P, D0.

Stk lvls: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 02/25/83 | NM | Added documentation |

## 6.14 AVM2DS - Buffer to Display

Category: DSPUTL File: SB&DSP::MS

Name: AVM2DS - Buffer to Display
Name:(S) BF2DSP - Buffer to Display
Name: BF2DS+ - Buffer to Display
Name: BF2DPP - Buffer to Display

Purpose:
AVM2DS: Send buffer at AVMEMS to display
BF2DPP: Send PROMPT to display
BF2DSP: Send buffer at D1 to display
BF2DS+: Send buffer at (D1) to display

Entry:

```
        P      =  0
        BF2DPP:  Set D1 @ PROMPT
        BF2DSP:  D1 points at first char of buffer
        BF2DS+:  D1 points at address of start of buffer addr
        AVM2DS:  none
```

Exit:

```
        P      =  0
        Carry set
```

Calls:

```
        DSPCHA
```

Uses.......

```
 Exclusive: D1,C(A),A(B)
 Inclusive: A(W),B(W),C(W),D(W),D0,D1
```

Stk lvls: 4

Detail:

In each case above the buffer is terminated by an FF byte.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/19/82 | B.S. | Updated documentation |

## 6.15 DSPCHA - Display Character

Category: DSPUTL File: SB&DSP::MS

Name:(S) DSPCHA - Display Character
Name:(S) DSPCHC - Display Character

Purpose:

Accepts a byte for pseudo-device display driver. The routines take data from A or C and send the character to the display.

Entry:
    P = 0
    DSPCHA: A(B) contains character
    DSPCHC: C(B) contains character

Exit:
    P = 0
    Carry clear

Calls: BLDDS*, BLDDSP, CKSREQ, CLEARD, D0=CUR, D0=FC, DSPCH., DSPTIM, FINDA, GETMSK, GETSTA, MOVC00, MOVCUR, MOVED3, MOVEU3, NOKEYS, PUTSTA, R<RSTK, RCLSTA, RSTK<R, SCNRT, SCRLLR, SETFCA, SLEEP, cksreq, TBLJMC, USRSTA.

Uses.......
 Inclusive: A(W),B(W),C(W),D(W),D0,D1,RAM(See note below)

Stk lvls: 2

NOTE:
This routine will call CKSREQ if CR or LF is sent. This implies that a poll may happen. That will cause certain RAM locations (SNAPSHOT) to be altered. This routine uses R<RSTK / RSTK<R to preserve stack levels--this also uses RAM.

This routine may also transfer control out to HPIL. The HPIL ROM may not have exactly the same register usage for a given character, ie don't assume a certain character will leave a certain register preserved just because this code doesn't seem to use it. For any character, A,B,C,D,D0,D1 may be used.

Detail:
This routine provides the mechanism to access the pseudo-device that controls the display. This device has 3 nibbles of status that are defined as follows:
- S0 -- Miscellaneus uses
- S1 -- Set iff LCD currently matches display buffer
- S2 -- Cursor blink phase
- S3 -- Display needn't contain cursor
- S4 -- Disable cursor update
- S5 -- Display buffer needs to be cleared(1)
- S6 -- Cursor on(0)/off(1)
- S7 -- Insert(1)/Replace(0) mode
- S8 -- Cursor/FirstC need to be cleared(1)
- S9 -- Suppress Delay(1) (Auto-clears)
- S10 -- Display has timed out

The pseudo-device accepts the following escape sequences:

```
  Esc Q -- Insert cursor
  Esc R -- Replace cursor
  Esc C -- Cursor right
  Esc D -- Cursor left
  Esc H -- Home cursor
  Esc J -- Clear Display
  Esc K -- Delete through end of line
  Esc > -- Cursor on
  Esc < -- Cursor off
  Esc E -- Reset display
  Esc P -- Delete char
  Esc % <col> <row> -- Set cursor position absolute
  Esc Ctrl-C -- Cursor far right
  Esc Ctrl-D -- Cursor far left
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/19/82 | B.S. | Updated documentation |
| 02/25/83 | NM | Updated "CALLS" section |

## 6.16 DSPCL? - Clear display buffer if necessary

Category: DSPUTL  File: SB&DSP::MS

Name:(S) DSPCL? - Clear display buffer if necessary

Purpose:
Clear display buffer if Clear bit set in display status
Reset cursor position if ResetC bit set in display status

Entry:
P = 0

Exit:

```
Calls:       GETSTR,CLEARD,PUTSTR

Uses.......
 Inclusive: A(M),C(B)

Stk lvls:    2

History:

   Date      Programmer               Modification
 --------    ----------    ----------------------------------
 11/01/83    B.S.          Added documentation
```

## 6.17 CRLFOF - Send cursor off/CR/LF to disp w/o delay

Category: DSPUTL     File: SB&DSP::MS

```
Name:(S) CRLFOF  -  Send cursor off/CR/LF to disp w/o delay
Name:(S) CRLFND  -  Send CR/LF to display with no delay
Name:(S) CRLFSD  -  Send CR/LF to display with delay

Purpose:
     CRLFOF:  Send Cursor off, Replace Cursor, CR, LF to
              display with delay suppressed.
     CRLFND:  Send Replace Cursor, CR, LF to display with
              delay suppressed.
     CRLFSD:  Send Replace Cursor, CR, LF to display with
              delay.

Entry:
     P       = 0

Exit:
     P       = 0

Calls:       CRLFOF:  ESCSEQ, XDELAY, BF2DSP
             CRLFND:  XDELAY,BF2DSP
             CRLFSD:  BF2DSP

Uses.......
 Inclusive: A,B,C,D,D0,D1
```

Stk lvls: 5

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/01/83 | B.S. | Added documentation |

## 6.18 SCNRT - Point Cursor Past Unprotected Field

Category: DSPUTL File: SB&DSP::MS

Name:(S) SCNRT - Point Cursor Past Unprotected Field

Purpose:
Scans to right of cursor and returns A(A) pointing past end of unprotected field, a null byte or end of display buffer whichever comes first.

Entry:
P = 0

Exit:
P = 0
A(A)=Points past unprotected display character
Carry set if pointer points past DSPBFE (i.e. buffer is full and protected to end of buffer).
B contains value of A at time of call.
D(A) points past cursor position.

Calls: CA=CUR,DO=CAA

Uses.......
Exclusive:
Inclusive: A,B,C,D(A),DO

Stk lvls: 1

History:

Date Programmer Modification

| | | |
|---|---|---|
| 10/19/82 | B.S. | Updated documentation |
| 03/17/83 | B.S. | Packed by calling subroutines |

## 6.19 ESCSEQ - Send Escape Sequence to Display

Category: DSPUTL File: SB&DSP::MS

```
Name:(S) ESCSEQ  -  Send Escape Sequence to Display

Purpose:
     Sends an escape to display followed by a specified
     character.

Entry:
     P      =  0
     C(B)=Character to follow escape character.

Exit:
     P      =  0

Calls:      DSPCHC

Uses.......
 Exclusive: C(B)
 Inclusive: A(W),B(W),C(W),D(W),D0,D1,RAM(See DSPCHA)

Stk lvls:   4

History:

  Date      Programmer               Modification
 --------   ----------   ----------------------------------
 07/15/82   B.S.         Added documentation
```

## 6.20 DSPRST - Display reset

Category: DSPUTL File: SB&DSP::MS

Name:(S) DSPRST - Display reset

Purpose:
Resets display driver pseudo-device: clears buffer, display mask, cursor position, first character, status, and window.

Entry:

Exit:
P = 0

Calls: None

Uses.......
Inclusive: C(W),P,D0

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/25/83 | B.S. | Added documentation |

## 6.21 LCDINI - Initialize LCD display

Category: DSPUTL File: SB&DVR::MS

Name:(S) LCDINI - Initialize LCD display

Purpose:

```
        Initialize LCD row driver and contrast, turn display on

   Entry:
        P       = 0

   Exit:
        P       = 0
        Carry clear

   Calls:       None

   Uses.......
    Inclusive:  C(W)

   Stk lvls:    0

   History:

      Date      Programmer                  Modification
    --------    ----------     ------------------------------------
    10/25/83    B.S.           Added documentation
```

## 6.22 SENDWD - Send Out Width-Sized Chunks to Device

```
        Category:  DSPUTL     File:  SB&IO::MS


   Name:(S) SENDWD  -  Send Out Width-Sized Chunks to Device
   Name:(S) SNDWD+  -  Send Out Width-Sized Chunks to Device

   Purpose:
        Send out width-sized chunks to display/printer device.

   Entry:
        STMTRO must have been set up correctly by CKINFO
        Status bit InhEOL (4):
            0= send out initial CR-LF if buffer won't fit in
               first width-sized chunk
               (only if position .ne. 0)
            1= start sending out buffer immediately, regardless
               if the buffer won't fit on the first line.
        SENDWD:
```

```
          A(A)= #characters (#bytes) in output buffer.
          D1 points to output buffer.
        SENDW+:
          B(A)= #character (#bytes) in output buffer.
          C(A) points to output buffer.

   Exit:
        P      = 0
        Carry set
        A(A)   = 0

   Calls:      CSLWP9, CSRWP9, SENDEL, SEND20, D1@POS, B2C95,
               CSLWP, CSRWP

   Uses.......
    Exclusive: A(A), B(A), C(W), D(A), P, D1,     R2
    Inclusive: A(W), B(W), C(W), D(W), P, D1, R1, R2
               Does not change D0, Status

   Stk lvls:   4

   NOTE:
        DO NOT CHANGE D0 OR STATUS BITS!!!

   Detail:
                15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
     R2 usage:    |#chars in bfr |  entry D1     |              |

   History:

     Date      Programmer              Modification
    --------   ----------    ------------------------------------
    08/26/82   M.B.          Wrote routine.
```

## 6.23 DSP$00 - Create String of Readable Characters

Category: DSPUTIL File: SB&IO::MS

Name:(S) DSP$00 - Create String of Readable Characters

Purpose:

Adds a string to stack containing all readable chars in display buffer.

Entry:

```
P      = 0
D1 points to top of stack
S0 set implies append CR and RTN when done.
S0 clear implies no CR on end and jump to EXPR when
   done.
```

Exit:

```
P      = 0
D1 points to new string on top of stack
If Return(S0) set then CR will have been appended
Exits to EXPR if S0 clear.
```

Calls: STKCHR,ADHEAD

Uses.......
 Exclusive: R1,D1,A(A),B(W),C(14-0),D(A)
 Inclusive: R1,D1,A(A),B(W),C(W),D(A)

Stk lvls: 3

Detail:

Examines display buffer and copys all "unprotected" characters into a string on the math stack. If S0 is set then a CR is appended following the last char in the string. A standard string header is attached with D1 pointing to it. If S0 is clear then the routine will jump to EXPR to continue expression execute instead of returning.

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/20/82 | B.S. | Updated documentation |

## 6.24 FINLIN - Finish line in display/video

Category: DSPUTL File: SB&IO::MS

Name:(S) FINLIN - Finish line in display/video

Purpose:
Finishes line in display and video by moving the cursor to the far right then sending CR/LF with no delay.

Entry:
P = 0

Exit:
Carry clear
P = 0

Calls: CURSFR,CRLFOF

Uses.......
Inclusive: A(W),B(W),C(W),D(W),D0,D1,ST(11-0)

Stk lvls: 4

Algorithm:
Unprotect last display buffer character
(This is needed to guarantee that even if the entire display line is protected the cursor can be moved past the last character on the video monitor line which will allow a CR/LF sent to the monitor to position the cursor past the last video line of this display line.)
Send cursor far right.
Restore protection bit of last character.
Send replace cursor, CR/LF with no delay.
Return with carry clear.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/01/83 | B.S. | Added documentation |

## 6.25 PRPSND - Prepare to send buffer to display

Category: DSPUTL File: SG&SYS::MS

```
Name:(S) PRPSND  -  Prepare to send buffer to display

Purpose:
     Sends buffer to ascii to display device

Entry:
     P       =  0
     HEXMODE
     B(A)    =  # of characters in buffer
     OUTBS   =  pointer to start of buffer
     RO      =  pointer past end of line
     S-R1-1     contains pointer to end of file

Exit:
     P       =  0
     buffer sent to display
     C(W)    =  RO

Calls:       SENDWD, SENDEL, CKINFO

Uses.......
 Inclusive: A,B,C,D,D1,DO,R1,R2

Stk lvls:    5

NOTE:
     This routine's integrity requires that for sending a
     buffer to a display device, SENDWD,SENDEL,CKINFO
     do not touch RO,R3!!!

History:

  Date      Programmer               Modification
 --------   ----------   ----------------------------------
 10/14/82   S.W.         Wrote routine
```

## 6.26 LSTLEN - Calculate #chars to list in display buf

Category: DSPUTL    File: SG&SYS::MS

```
Name:(S) LSTLEN  -  Calculate #chars to list in display buf

Purpose:    Calculates number of chars in (display) buffer.

Entry:      (OUTBS) = Address of buffer start
            DO      = Address past last character in buffer
          2 ENTRY POINTS:
            1) LSTLN+ - 1st calls OUTBYT; preserves 1st
                        5 nibbles of RO.
            2) LSTLEN - Ptr to save in C(A)

Exit:     B(A) = number of characters in buffer
          Carry clear
          Pointer saved on entry is restored into RO
           via OBCOLL (collapse of OUTPUT buffer)

Calls:    OUTBYT, AVS=DO, OBLCMP, MFWRNQ

Uses:
          exclusive... A(A), B, C(A), RO
          inclusive... A-D, P, D1,DO, RO

Stack lvls: 5

Detail:     If #chars to output >=95, then 95 returned
            as number of characters in buffer and a
            "Line Too Long" warning is sent out.

History:

  Date      Programmer   Modifications
 --------   ----------   -------------
 07/06/82   S.W.         Improved documentation
 12/21/82   S.W.         Added 'Line too long'
```

## 6.27 DONNA - Re-prompt input line

Category: DSPUTL  File: TI&ERD::MS

Name:(S) DONNA - Re-prompt input line

Purpose:
Re-display an input line, with a prompt, and position the cursor to any desired point in the line.

Entry:
R3(A)= Address of prompt. The prompt can be any ASCII string, delimited with two matching bytes (delimiters can be any byte value).
R3(9-5)= Number of cursor-rights to position the cursor within the input stream (counted from the first input character).
INBS contains the address of the input buffer; the length of the input buffer is contained in the three nibbles preceding the buffer.

Exit:
P = 0
Carry set.
D1=FFFFF.

Calls: CKINF-, DSPBUF (SENDWD), ESCSEQ, DSPCNA, DSPCHA, CURSFL, CURSRR.

Uses.......
Exclusive: A(W),B(W),C(W),D(W),D0,D1,P
Inclusive: same plus R1,R2 (in SENDWD), STMTR0 (in CKINF-)

Stk lvls: 4

NOTE:
The prompt is built in the display observing WIDTH; the input line is displayed without observing WIDTH. Any single-character prompt will not have to worry about this, but a multi-character prompt may be split between two lines if WIDTH is short.

The length of the input buffer (found in the three nibbles preceding the buffer) must be one greater than the number of characters (usually this length includes a 0D terminator at the end of a BASIC input line).

Example of prompt:
Say an editor uses the prompt "Cmd:". The address in R3(A) would point to the characters xCmd:x where the x's are delimiters, any matching byte value.

Algorithm:
Turn off cursor.
Set up CKINFO.
Display prompt.
Redisplay input line.
Send out a null character (in case input line had zero length, this clears display buffer)
Cursor far left.
Count cursor-rights, using count in R3(9-5).

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/05/82 | MB | Documentation |

## 6.28 CURSRT - Count cursor-rights

Category: DSPUTL File: TI&ERD::MS

Name:(S) CURSRT - Count cursor-rights

Purpose:
Send out a cursor-far left, then send out a given number of cursor-rights.

Entry:
C(A)= count of cursor-rights.

Exit:
P = 0
Carry set.
D1=FFFFF.

Calls: CURSFL, CURSRR

Uses.......
 Exclusive: D1
 Inclusive: A(W),B(W),C(W),D(W),D0,D1,P

Stk lvls: 3

Algorithm:
 Copy counter to D1
 Cursor far-left
 Count cursor-rights until D1 carries

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/05/82 | MB | documentation |

## 6.29 AVS2DS - AvMemSt to display

Category: DSPUTL File: TI&ERD::MS

Name:(S) AVS2DS - AvMemSt to display

Purpose:
 Send ASCII stored at AvMemSt to display.

Entry:
 P = 0 (P is used to select options, must =0!)
 ASCII characters reside in memory starting at
 AvMemSt; an FF byte must immediately follow
 the characters.

Exit:
 P = 0
 Carry clear.

Calls: D0=AVS, DSPBUF

For all other details, see DSPBUF.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/25/82 | MB | documentation |

## 6.30 DSPCNA - Display by count

Category: DSPUTL File: TI&ERD::MS

Name:(S) DSPCNA - Display by count
Name:(S) DSPCNB - Display by count
Name:(S) DSPCNO - Display by count

Purpose:
Send ASCII characters to display, by count.
DSPCNO -- Counter in B(A), use Output Buffer.
DSPCNB -- Counter in B(A), use DAT0.
DSPCNA -- Counter in A(A), use DAT0.

Entry:
DSPCNO -- #characters-1 in B(A), output resides in
Output Buffer (address in OUTBS).
DSPCNB -- #characters-1 in B(A), D0 points to output
DSPCNA -- #characters-1 in A(A), D0 points to output

Exit:
P = 0
Carry clear.

Calls: D0OUTBS (DSPCNO only), DSPBUF

For all other details, see DSPBUF

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/25/82 | MB | documentation |

## 6.31 DSPBUF - Send a buffer of chars to display

Category: DSPUTL File: TI&ERD::MS

```
Name:(S) DSPBUF  -  Send a buffer of chars to display

Purpose:
     Send a buffer of characters to display, allowing
       1) terminate buffer on count or FF byte.
       2) observe WIDTH or not.

Entry:

  (1)--------------------
   |
   | P= 0     Send out characters until a terminator
   |          byte is encountered (terminator byte is
   |          passed in A(B)).  Do not observe WIDTH
   |          (i.e., do not split up display into
   |          WIDTH-sized chunks).
   |
   |    2     Count characters.  Send out characters
   |          until counter decrements (counter passed
   |          in A(A)).  Do not observe WIDTH.
   |
   |    4     Send out characters until a terminator
   |          byte is encountered (terminator byte is
   |          passed in A(B)).  Observe WIDTH.
   |
   |   Note: The combination "Count characters and
   |         observe WIDTH" is performed by SENDWD.
   |


  (2)--------------------
   |
   |  If P=2 (send by count):
   |    A(A)= #characters in buffer.
   |
   |  If P=0 or 4 (send until terminator):
   |    A(B)= terminator byte
   |    B(A)= 0  (used for separate counter).
```

```
    |

   (3)---------------------
    |
    |  D0 points to output buffer.
    |
```

Exit:
```
    P      = 0
    Carry clear.
```

```
Calls:      CSLWP9
            DSPCHA (for entry P=2 only)
            SENDWD (for entry P=0 or 4 only)

Uses.......
 Exclusive: P,A,C,D1,R0(10-5)
 Inclusive: B,D,D0
            R1,R2 and STMTR0 (in SENDWD) for P=0 or 4 only

Stk lvls:   3
```

NOTE:
    R0(15-11) and R0(4-0) are not touched by this routine.

Algorithm:
```
    Swap P (options) into ST1, swap ST1 into R0(10).
 1) If by count, decrement counter; if carry, goto 2).
    If by terminator, test match; if match, goto 2).
    If observe width, count buffer length in B(A),
      go to 1).
    Save counter or match in R0.
    Send out character (DSPCHA).
    Fetch counter or match in R0.
    Go to 1).
 2) If observe width, call SENDWD with length in A.
    Restore ST1 from R0(10).
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/25/82 | MB | documentation |

```
+-------------------------------------------+-----------------+
|                                           |                 |
|  DCMUTL  -  Decompile Utilities           |  CHAPTER  7     |
|                                           |                 |
+-------------------------------------------+-----------------+
```

## 7.1 CURSRU - Cursor Up

**Category:** DCMUTL **File:** JP&MEM::MS

```
Name:(S) CURSRU  -  Cursor Up
Name:(S) CURSRD  -  Cursor Down
Name:(S) CURTOP  -  Cursor Top
Name:(S) CURBOT  -  Cursor Bottom
Name:(S) DCPLIN  -  Decompile line and display it
Name:(S) DSPLI+  -  Display line with cursor on;calc cursor pos.
Name:(S) DSPLIN  -  Display line with cursor on;pass cursor pos.

Purpose:  Cursor UP, Cursor DOWN, Cursor TOP, Cursor BOTTOM
            FETCH "next line" in program memory
            Scroll Cursor Up | Cursor Down
          Decompile and Display line with Cursor on

Entry:    CURBOT: Sets Cursor Bottom flag        sCURBT
                  Clear Cursor Up flag           sCURUP
                  Displays last line of non-null program
          CURTOP: Clears Cursor Bottom flag      sCURBT
                  Displays first line in a non-null program
          CUR020: Entry for FETCH w/ CURRL=0
                  Assumes sCURBT=0
                  Avoids CR/LF and Poll for Cursor UP/DOWN
          CURSRU: Sets Cursor Up flag            sCURUP
          CURSRD: Clears Cursor Up flag          sCURUP
          DCPLIN: Decompile & Display Line Entry
                  D1 @ Line to decompile
          DSPLI+: Display line entry in output buffer
                  #cursor rights needed will be calculated
                  FETCH KEY entry
          DSPLIN: Display Line Entry in Output Buffer
                  AUTOX entry
                  A(A) = #backspaces for cursor position
                       = #cursor rights
                  OUTBS @ Start of line to decompile
```

```
          CURRL = Current line# referenced
          After FINDL call:
                SO=0 if Line# > found
                SO=1 if Line# not found
                S1=1 if Null program memory
                DO = Previous Line found
                D =  End of current program

Exit:     If Private program
             Error Exit <-- eFPROT
          If not BASIC program
             Poll on pCURSR
             If no response:
                Error Exit <-- eFTYPE
          If no CURRent Line # or Null Program file
             Return to Main Loop
          else
             D1 = Start of line to Decompile
             Decompile & Display line w/ Cursor
             Return to MAIN30 to preserve display

Calls:    FINDL,LDCOMP,BF2DPP,DSPCHO,NXTLIN,RDCHDR
          NULLP,BLDDSP,FPOLL,CURRLO,DO=OBS,CURSRT,
          CRGTPR (CRLFSD & GETPeF)

Uses:     A-D,DO,D1,CURRL,RO-R2,SO,S1,S5-S8
          For Cursor entry: sCURUP (S2), sCURBT (S3)

            sCURUP = Cursor Up
            sCURBT = Cursor Bottom
            RO=  # backspaces for cursor position after line#

Stk Lvls:   5

Detail:

  sCURBOT, sCURUP set/cleared for pCURSR to guarantee
  unique determined of Cursor key.


  CURBOT: Clear Cursor Up flag                      (sCURUP)
          Set Cursor Bottom flag                    (sCURBT)
          goto 0:
  CURTOP: Clear Cursor Bottom flag                  (sCURBT)
          Set Cursor Up flag
      0:  Send Carriage Return / Line Feed          (CRGTPR)
          If Private Program                        (GETPeF)
             Error Exit                             (eFPROT)
          Set status to check file type & error     (S9)
          If non BASIC program                   (Carry set)
    0.5:     Poll for Cursor keys                   (pCURSR)
```

```
                Error exit if no response             (eFTYPE)
    CUR020:  If NULL Program
                golong MAINLP
             If CURTOP
                Position to line# of First line       (D1+1EOL)
                go Decompile and Display line         (DCPLIN)
             else
                go Find Last Line in file
                   Line# <--- FFFF                    (goto 3)
     CURSRU: Set Cursor Up flag                       (sCURUP)
             Set Cursor Botton flag
             goto 1;
     CURSRD: Clear Cursor Up flag                     (sCURUP)
             Clear Cursor Bottom flag
         1:  Send CR/LF                               (CRGTPR)
             If Private program                       (GETPeF)
                Error Exit                            (eFPROT)
             If non-BASIC program                  (Carry set)
                Issue pCURSR poll                   (goto 0.5)
             Read current Line#                       (CURRLO)
         3:  Find Line#                               (FINDL)
             If Line# NOT found     (Carry clear)
                If Cursor Down
                   If Line# > found                   (S0=0)
                      go Decompile & Display line     (DCPLIN)
                   else
                      If NULL program                 (S1=1)
                         goto Main Loop
                      else
                         go Decompile previous line  (goto 4)
                If Cursor Up                          (sCURUP)
                   If NULL program                    (S1=1)
                      goto Main Loop
                   else
                      Get First line of File          (RDHDR1)
                      go Decompile previous|first line (goto 4)

             If Line# found    (Carry set)
                If Cursor Down
                   Save current line position         (B)
                   Get next line                      (NXTLIN)
                   If next line >= End of program
                      Next line <-- Saved current line
                   go Decompile & display line        (DCPLIN)
                If Cursor Up                          (sCURUP)
       4:          If previous line # 0               (D0)
                      Next line = Previous line
                   else
                      D1 @ Line to Decompile & Display

     DCPLIN: Decompile line @ D1                      (LDCOMP)
```

```
          Calculate # backspaces to space after line#
  DSPLIN: Display line
              Send prompt                            (BF2DPP)
              Send buffer                            (DSPCNO)
              Send backspaces (cursor rights)        (CURSRT)
              Build display                          (BLDDSP)
          Return to Main, Keep display               (MAIN30)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/01/83 | JP | Added pCURSR poll on File Type |
| 04/12/83 | JP | Ignore CURRL=0 |
| 04/12/83 | JP | CUR020 entry point for FETCH |
| 07/15/83 | JP | Send CR/LF before Private check |

## 7.2 EXPRDC - Expression Decompile

Category: DCMUTL File: SB&EXD::MS

Name:(S) EXPRDC - Expression Decompile
Name:(S) EXDCLP - Funny function decompile reentry point

Purpose:
EXPRDC: Decompile expression lists
EXDCLP: This is the point where funny function decompile routines should reenter the expression decompiler.

Entry:
EXPRDC:
D0=Output stream pointer
D1=Input stream pointer
D(A)=End of avail mem pointer
A(B)=Contents of MEM(D1)
P=0
EXDCLP:
D1 is current input pointer(past FFN tokenization)
D0 is current output pointer(past FFN text)

Exit:
    D0=Updated output pointer
    D1=Updated input pointer(First unused byte)
    A(B)=First unused token
    Carry clear
    P = 0

Calls:      VARDC,MOVEDO,RANGE,DRANGE,OUT1TK,OUTNBC,MEMERR

Uses.......
 Inclusive: A,B,C,R0,R1,R2,S0,S3,S8,S10,S11,D0,D1

Stk lvls:   4

Detail:
        R0 = Output pointer @ entry
        R1 = Temporary input pointer, Sign holds text len
        R2 = Function text

Explanation of terms used:

Nullop -- This a 00 byte which is used to preserve a spot in the output stream to insert an operator later. It also is used as a marker to help find the spot later.

Denature -- Once operators have been enclosed in parentheses or in a function call, the of that operator is no longer of any consequence to the rest of the expression. To prevent operators so enclosed from affecting precedence, they are changed (denatured) in such a way that they do not look like operators but can be recogized later when the time comes to expand the operator token into the text that corresponds to the token.

The expression decompiler keeps track of whether the expression has the form of a reference expression. To do this, it uses two status bits, NewVal and OldVal. Each pass through the decompile loop, the NewVal flag is copied to the OldVal flag, and the NewVal flag set, then if the token being decompiled is a variable or an array token, the NewVal flag is cleared. When the loop finally hits a token which terminates the expression, the OldVal flag will be clear only if the last token in the expression is not a variable or an array. This is equivalent to whether the expression has the form of a value expression. If the token that terminated the expression was a call by value token and the OldVal flag was left clear, then an extra set of parenthesis is placed around the entire expression. This feature is used in SUB decompile.

Algorithm:

Expression decompile converts an RPN string of operands, operators and functions to an algebraic stream of characters. The RPN stream is examined an item at a time starting at the beginning (lowest address). There are several types of items which may be encountered in the stream. The following summarizes what happens for each type:

- Operands -- output a nullop followed by text for constant
  - Single digit constants
  - Integer constants (2-12 digits)
  - Floating point constants (1-12 digits)
  - String constants (single or double quoted strings)
  - Alpha variables
  - Alpha-digit variables
  - String alpha variables
  - String alpha-digit variables
- Monadic operators -- search back for a nullop, insert operator token just after nullop, insert parentheses around that area if an operator of lower precedence is there. If parentheses were inserted then denature any operators enclosed therein.
  - Unary minus
  - NOT
- Dyadic operators -- search back for a nullop, replace this nullop with the operator token, insert parentheses around that area if an operator of equal or lower precedence is there. If parentheses were inserted then denature any operator enclosed therein.
  Now search back for another nullop, insert parentheses if any operator of lower precedence was encountered and denature all operators within these parentheses.
  - ^
  - *
  - /
  - %
  - DIV
  - +
  - &
  - Relops
  - AND
  - OR
- Functions -- Determine number of parameters.

If the function has no parameters then treat it like an operand. If it has more than one parameter then for each "extra" parameter look back for a nullop and replace it with a comma. Now search for a nullop and insert the function name and a left parenthesis just after it and denature all operators between it and the end of the output where a closing parenthesis is appended.

Funny functions -- Output a nullop and the text for the function name then call the functions "decompile" routine.

LPRP token -- Output a left and right parernthesis.

When any token other than one of the above is encountered, that marks the end of the expression. The entire output stream is moved from the beginning of available memory to the end of available memory. Now the copy back process begins. The first byte of the output stream should be a nullop and is ignored. Each remaining byte in the output stream is copied back to the beginning of available memory except that operators (and denatured operators) are expanded to their full text representation and any embedded nullops are converted to commas. When a single or double quote is encountered, bytes are copied verbatim until the corresponding closing quote is found.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/24/82 | B.S. | Updated documentation |
| 11/09/82 | B.S. | Merged Dummy array decompile into expression decompile |
| 08/30/83 | B.S. | Fixed bug in DNATUR (39-1017(3)) |
| 09/06/83 | B.S. | Added to documentation |

## 7.3 HXDASC - Hex to decimal ASCII conversion

Category: DCMUTL    File: SB&EXD::MS

Name:(S) HXDASC - Hex to decimal ASCII conversion

Purpose:
Converts a byte to a 3 character decimal ASCII string.
Output string contains leading zero(s) if <100.

Entry:
A(B) contains byte to convert

Exit:
B(15-10) contain 3 ASCII decimal digits
P = 0

Calls: None

Uses.......
Inclusive: B(W),A(B),C(S),C(B)

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/06/83 | B.S. | Added documentation |

## 7.4 ARITH - Get Text For An Arithmetic Operator

Category: DCMUTL    File: SB&EXD::MS

Name:(S) ARITH - Get Text For An Arithmetic Operator

```
Purpose:
     Returns text for an arithmetic operator

Entry:
     P       =  0
     A(B)=arithmetic operator

Exit:
     P       =  WP length of text of arithmetic operator
     C(WP)   =  Text for arithmetic operator
                (First ASCII char is in low C, last in high)
     Carry clear

Calls:       None

Uses.......
 Inclusive: A(X),C(W),P

Stk lvls:    0

History:

  Date      Programmer                Modification
 --------   ----------   ------------------------------------
 08/01/82   SA           Wrote routine
 10/19/82   B.S.         Added documentation
```

## 7.5 LDCOMP - Line Decompile Driver

Category: DCMUTL    File: SG&LDC::MS

```
Name:(S) LDCOMP  -  Line Decompile Driver
Name:(S) LDCM10  -  Line Decompile Driver
Name:(S) LDCEXT  -  Line Decompile Driver
Name:(S) LDSST1  -  Line Decompile Driver
Name:(S) LDSST2  -  Line Decompile Driver

Purpose:
     LINE DECOMPILE DRIVER

Entry:
```

```
      P=0
      D1 @ BEGINNING OF COMPILED LINE IN RAM.

      LDCOMP:  1) Updates current line
               2) Clears SST flag
               3) Decompiles entire BASIC line
      LDCM10:  Does 2 & 3 above
      LDCEXT:  Same as LDCM10, EXCEPT that this is
               used to 'externally invoke' decompile.
               Any memerr will return control to caller
               with carry set.
      sSSTdc=1 => only decompiles 1 stmt at a time
      LDSST1:  SST entry for Decompile w/ Line#
               Assumes sSSTdc (S1) set appropriately
      LDSST2 : SST entry for Multi-stmt Line
               Assumes sSSTdc (S1) set appropriately

Exit:
     Normal entry:
     Carry Clear   (through LSTLEN exit)
     Decompiled Line sent to Input/Output Buffer
     R0 past tEOL
     If LDSST1/LDSST2 entry
        D1 @ Tokenized Statement Terminator
     B(A) = BUFFER LENGTH (#CHARACTERS)
     Output Buffer collasped --> AVMEMS <-- OUTBS

     If LDCEXT entry is used:
     Carry clear => normal exit
                    D0 past ascii stream
                    OUTBS is start of ascii stream
                    A(A) past tEOL of line decompiled
     Carry set   => Memerr

Calls:      RTNSET, SAVELN, LDCSET, LIN#DC, GTXT+1, AD1+2,
            'DC (OUTBYT), ASCICK, !TEST, GTEXTI, OUTNBC
            LIN#AU

Uses.......
 Exclusive: A-D, D1,D0, S0,S3,S5,S6,S7,S8,sSSTdc (S1)
 Inclusive: A-D, D1,D0, S0,S3,S5,S6,S7,S8,S1, R0-R2,
            S-R0-2 & flRTN  (if LDCEXT entry used)

     sSSTdc = SST Decompile - GLOBAL throughout decompile
     S6 (VARDC),S8,CURRL

     R0 = Pointer past tEOL (provided LDCEXT not used)
     R1 = Preserved D1
     R2 = Main Table Address
     R3 cannot be used, it is used by "LIST"
```

SAVES NEW LINEN INTO CURRL (UPDATES CURRENT LINE)
CLEARS S8 FOR ALL BEGIN BASIC DECOMPILE STATEMENTS:
(CURRENTLY USED BY LISTDC & USER/BEEPDC)

Stk lvls: 6

Note:

No single Decompile routine can used more then 6 lvls
EXPRDC uses 4 subroutine levels

sSSTdc (S1) must not be used by individual Decompile routines

Any decompile routine that POLLs must set AVMEMS at the Current DO (call AVS=DO). This prevents the Poll Save area from overwriting the Output Buffer. Decompile, on exit, will set AVMEMS back @ OUTBS.

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | J.P. | Modified documentation |
| 08/30/82 | J.P. | Fixed SST/ELSE decompile |

## 7.6 GTEXT - Get Text for Keyword/Function

Category: DCMUTL File: SG&LDC::MS

Name:(S) GTEXT - Get Text for Keyword/Function
Name: GTEXTM - Get Text for Keyword/Function
Name: GTEXTX - Get Text for Keyword/Function

Purpose:
Get Text for Keyword or Function

Entry:
DO pointing into output buffer
D(A) contains available memory end (AVMEME)
GTEXTI: A = Main token | XWORD token | XFN token

```
                D1 = At Keyword | Function token
                D1 incremented by 2 on entry
                P=0

        GTEXT:  A = Main token | XWORD token | XFN token
                D1 = Past Keyword | Function token
                P=0

        GTEXTM: Mainframe Lex Table used
                D1 = Past Keyword | Function token
                P=0

        GTEXTX: XWORD Lex Tables used
                D1 = Past XWORD | XFN token

   Exit:
        P=0
        Carry Clear
             A = Text
          C(S)= # nibbles - 1
            D1 @ Execution address for token
            R1 = D1 on entry            (Past token)
                 D1+4 on entry if XWORD  (Past Lex ID and Entry#)

        Carry Set
            XWORD | XFN not found
            D1 @ D1 on entry             ( @ Lex ID )

   Calls:      XMTADR, MTADR+

   Uses.......
    Exclusive: A,B,C,R1,R2,D1
    Inclusive: A,B,C,R1,R2,D1

        R1 = D1 @ entry
        R2 = Main Table Address

   Stk lvls:   2

   Algorithm:

   GTEXTI: Increment D1 past token
   GTEXT:  If XWORD or XFN
                goto GTEXTX
   GTEXTM: Load MAINT address
           Save token --> B(A)
           goto 1;
   GTEXTX: Read LEX ID, Entry#
           Calculate Main Table address (XMTADR)
           If address NOT found ---> RTNC
           Skip over LEX ID and Entry#
```

```
1: Save D1        (R1)
   Save Main Table Address  (R2)
   Calculate Start of Text Table
     Txt Tbl Rel Addr Ptr @ Main Table Addr - oSPDn2 + 1
     Txt Tbl Start = Txt Tbl Ptr + (Txt Tbl Ptr)
   D1 <-- Text Table Start
   C  <-- Main Table Address  (R2)
   Calculate Entry into Main Table  (MTADR+)
   Read Text Table Offset
   Read Execution Address & Save it (R2)
   Compute Entry into Text Table
   Read # nibbles for text          (C(S))
   Read ASCII Text                  (A)
   Set D1 = Execution address       (R2)
   RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | J.P. | Modified documentation |
| 08/17/82 | S.W. | Added GTEXTI entry point |
| 12/06/82 | J.P. | Fixed XWORD not found exit conditions |

## 7.7 LIN#DC - Line number decompile

Category: DCMUTL     File: SG&LDC::MS

```
Name:(S) LIN#DC  -  Line number decompile
Name:(S) LIN#D+  -  Line number decompile
Name:(S) LIN#AU  -  Line number decompile
Name:    LIN#A+  -  Line number decompile
Name:    LIN#CK  -  Line number decompile
```

Purpose: Decompiles a line number & outputs it

Entry:

```
P=0
D(A) points to end of available memory (AVMEME)
D0 positioned at where decompiled line number to go
  5 ENTRY POINTS :
```

```
1) LIN#CK  - Returns with carry set if A(B) # tLINE#.
             Otherwise, falls into LIN#D+ entry pt.
2) LIN#D+  - Assumes D1 is 7 nibs prior to 1st
             digit of 4 nibble line number
             (2 nib line# token, 5 nib jump addr)
             Suppresses leading 0's
3) LIN#DC  - Same as above, except assumes that D1
             is pointing to 1st digit of line number
4) LIN#AU  - Used by TRACE - also suppresses leading
             zeroes.
             P=0 => 4 digits output, leading zeroes
                    suppressed.
             P=12 => Convert from HEX to DEC. 2
                    digits output, up to 4 leading
                    zeroes suppressed.
             P=14 => Convert from HEX to DEC. 2
                    digits output, up to 6 leading
                    zeroes suppressed.

5) LIN#A+  - Used by System command AUTO - same as
             above except line# already in B(3-0)
```

```
Exit:        D0 updated/ P=0/ Carry clear
             LIN#AU, LIN#A+ - D1 left intact
             LIN#DC, LIN#D+ - D1 stepped over 4 nibble line#

Calls:       DOASCI

Uses:        A, B, C, D(S), P

Stack lvls:  2

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/06/82    S.W.         Added documentation
 10/18/82    S.W.         Added P=0 entry condition
```

## 7.8 ASCICK - Ascii Stream Deconpiler

Category: DCMUTL File: SG&LDC::MS

```
Name:(S) ASCICK   -  Ascii Stream Deconpiler
Name:    ASC02    -  Ascii Stream Deconpiler

Purpose:    Outputs stream of ascii characters

Entry:      3 ENTRY POINTS :
           D0 points to where output to go
           D(A) contains end of available memory (AVMEME)
           1) ASCI+  - D1 at 2 nibs prior to alleged start
                       of stream.
           2) ASCICK - D1 at start of alleged ascii stream.
           3) ASC02 -  Same as (2) above, only 1st character
                       already known to be ascii & is in C(B)

Exit:       Carry clr
            D1 past the ascii stream
            C(B) contains 'terminating' 1-byte token

Calls:      OUTBYT

Uses:       A(B), C(B), D1, D0

Detail:     If there's no ascii characters, nothing will be
            output & D1 will be left at 1st token
            Interprets as ascii any 1 byte token in which
            bit 7 is clear.

Stack lvls: 2

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/06/82    S.W.         Inproved documentation
```

## 7.9 ARYDC - Array Decompile

Category: DCMUTL File: SG&LDC::MS

```
Name:(S) ARYDC   - Array Decompile

Purpose:    Decompiles Array compiled in ARRYCK format

Entry       P= 0
     D(A) contains available memory end (AVMEME)
     D0 points into output buffer
     2 entry points:
     1) ARYDC  - Assumes C(B)=a(
                 D1 at first subscript, S5=0
     2) ARYDC+ - Checks for substring declaration (tSEMIC)
                 in A(B).  If not found, returns w/carry set.
                 Else D1 stepped over tSEMIC & expression
                 decompiled enclosed in brackets.

Exit:
      Carry clear=>
            subscripts output between parens (or brackets)
              parens (or brackets)
            D1 at token following last subscript
            A(B) contains the token
            If ARYDC+ called, S5=1
      Carry set (ARYDC+ entry only) =>
            No subscript decl. found

Calls:    OUTBYT, DBEXPR

Uses:     A-C, D1,D0, S5
          R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Stack lvls: 5

History:

   Date     Programmer   Modifications
 --------   ----------   -------------
 07/06/82   S.W.         Improved documentation
 08/16/82   S.W.         Added ARYDC+ entry
```

## 7.10 GTEXT+ - GTEXT Preprocessor

Category: DCMUTL File: SG&LDC::MS

```
Name:(S) GTEXT+  -  GTEXT Preprocessor
Name:(S) GTXT++  -  GTEXT Preprocessor
Name:(S) GTEXT1  -  GTEXT Preprocessor
Name:(S) BLNKCK  -  Blank Check

Purpose:
     Given a keyword, GTEXT+, GTXT++, and GTEXT1 outputs
     the corresponding text.

     The BLNKCK entry point ensures that there is exactly
     one blank after the last item decompiled.

Entry:
     For all entry points:
     P      =  0
     D(A)   =  AVMEME
     D0     =  Ptr to output buffer

     BLNKCK entry:
     -------------
     No additional entry requirements

     GTEXT+, GTXT++, GTXT1 entry:
     ----------------------------
     S9=1 => Output a trailing blank
     D1      at keyword

       1) GTXT++ - Outputs a leading & trailing blank
                   Sets S9; Doesn't attempt to decompile
                   text if token < 7E
       2) GTEXT+ - Doesn't attempt to decompile text if
                   token < 7E
       3) GTEXT1 - Assumes A(B) already loaded with token
                   greater than 6A.  No leading blank output

Note:  Can't call 1 or 2 above if want to output text
        associated with a keyword in the range 6A-7D

Exit:
     GTEXT+, GTXT++, GTEXT1 entry:
     ----------------------------
```

```
P      = 0
S9 set if GTXT++ used
Carry set => Keyword not found, D1 intact
      clr => Text output, D1 past token, D0 advanced

BLNKCK entry:
-------------
Exactly 1 blank follows last item decompiled.
D0 points past that blank
```

```
Calls:      GTEXTI, OUTBYT, OUTNBS

Uses:       A-C, R1-R2, D1,D0       (GTEXT1, GTEXT+ entry)
            A-C, R1-R2, D1,D0, S9   (GTXT++ entry)
            A(B),C(B),D0            (BLNKCK entry)

Stk lvls:   2  BLNKCK entry
            3  All other entry points
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/12/82 | S.W. | Routine created |

## 7.11 OUTELA - Output End of Stmt Terminator From A

Category: DCMUTL    File: SG&LDC::MS

```
Name:(S) OUTELA  -  Output End of Stmt Terminator From A
Name:(S) OUTEL1  -  Exit for End of Stmt Decompile
Name:(S) EOLXC*  -  Check for End of Stmt Decompile
Name:(S) TRACDC  -  TRACE Statement Decompile
Name:    REMDC   -  REMark or DATA Statement Decompile
Name:    OUTEOL  -  Output End of Statement
Name:    ENDDC   -  Decompile END Statement

Purpose:  Entry points to handle end of statement decompile
          and misc statement decompile

Entry:
```

```
        P=0
        D(A) contains AVMEME
        D1 points into token stream
        D0 points  into ascii output buffer
             ENTRY POINTS:
               1) OUTELA - also STOPDC
                  D1 at statement terminator (already read
                  into A(B))
               2) OUTEL1 - End of statement decompile
                  D1 at statement terminator
               3) EOLXC* - Doesn't return if D1 is at stmt
                  end, else does
               4) TRACDC - TRACE and DEFAULT decompile
                           Outputs single keyword - no blanks
               5) REMDC  - also DATADC; D1 pointing after
                  tREM or tDATA.
               6) OUTEOL - D1 at tEOL
               7) ENDDC  - Looks for ALL token
                           Falls into OUTEL1
```

**Exit:**

```
        If not called externally, exits via LSTLEN
         with carry clear
        If upon entry, D1 at tEOL or t! :
         D1 at tEOL, D1 untouched
                    D0 pts past last decompiled char
                    B(A)=#chars in buffer

        If upon entry, D1 at tELSE or t@ :
         Decompile is continued, via ELSEDC &
         LDCM20, respectively.

        If SST decompile and ELSE
           ELSE statement NOT decompile
           Jump to tEOL processing

        If SST decompile and Multi-statement line
           Decompile does not continue
           Don't decompile past @

        REM/DATA entry points - statement decompiled
```

**Calls:** BLNKCK, OUT1TK, EOLDC, GTEXT1, !TEST, OUT2TK, TRNFCK, REMP10

**Stack lvls:** 4

**Uses:** sSSTdc (S1)

**History:**

| Date | Programmer | Modifications |
|---|---|---|
| 07/07/82 | S.W. | Improved documentation |
| 08/30/82 | J.P. | Added SST/ELSE checks |
| 10/27/82 | J.P. | Added END ALL Decompile |

## 7.12 VARDC - Variable Decompile

Category: DCMUTL File: SG&LDC::MS

```
Name:(S) VARDC  - Variable Decompile
Name:    VARDC+ - Variable Decompile

Purpose:    Decompiles variables

Entry:
            P=0
            D(A) contains available memory end (AVMEME)
            D1   input pointer
            D0   output pointer
            S8=1 => no attempt to decompile arrays
                   (used by EXPRDC)
            2 entry points:
              1) VARDC+ - D1 2 nibs before alleged variable
              2) VARDC  - D1 at alleged variable

Exit:     P=0
            Regardless of S8:
            Carry clr => Variable found & decompiled
                         D1 past variable token
                         A(B)=B(B)= following token

            S8 clr on entry:
            Carry clr => If S6 set, then decompiled
                         variable descriptor of array
            Carry set => no variable found

            S8 set on entry:
            Carry clr => 00 byte output prior to decompiled
                         variable.
            Carry set => either variable not found or
```

encountered tARRAY

Calls: ADDCHR, RANGE, OUTNBS

Uses: A, B, C(S), C(A), S6

Stack lvls: 1

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/06/82 | S.W. | Improved documentation |
| 10/18/82 | S.W. | Added P=0 entry condition |
| 06/09/83 | S.W. | Changed A=B B => A=B A (pack) |

## 7.13 LABLDC - Lable Decompile

Category: DCMUTL File: SG&LDC::MS

Name:(S) LABLDC - Lable Decompile

Purpose:
Decompiles label references

Entry:
D1 at tLBLRF
D0 output pointer
P=0
D(A) contains available memory end (AVMEME)

Exit:
P=0
Carry clear
D1 past string expression or literal
D0 past decompiled label

If string expression, through EXPRDC
else through OUTBYT

Calls: AD1+2, ASCICK, OUTBYT

Uses.....
 Inclusive: A,C, D1,D0
 Exclusive: A-C, D1,D0, R0-R2, S0,S3,S8,S10,S11 - EXPRDC

Stk lvls: 4

Detail:
    tLBLRF tLITRL <asci label>
    tLBLRF <string expression>

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | J.P. | Modified documentation |

## 7.14 FILDC* - File Decompile

Category: DCMUTL File: SG&LDC::MS

Name:(S) FILDC* - File Decompile

Purpose: Decompiles mainframe file specifiers & HPIL file specifiers if HPIL plugged in

Entry:
    P=0
    D(A) contains available memory end
    D0 output pointer
    2 entry points:
    1) FILDC+ - D1 hasn't yet been incremented.
    2) FILDC* - D1 already at file spec

Exit: D1 past file specifier
    File specifier decompiled, with D0 updated
    P=0

Calls: POLLD+, OUTNBS, ASCICK, EXPRDC, OUT1TK, GTEXT+, FINDA, D=AVME

Uses: S8,S9, A-C, D1,D0, R1,R2

A-C, D1,D0, R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Stack lvls: 5

Detail: Will check for tKEYS, tCARD, tPCRD

Assumes that non-mainframe file specs are tokenized with preceding tCOLON.

Must immediately precede SKIPDC code, since it falls into SKIPDC.

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/07/82 | S.W. | Improved documentation |

## 7.15 SKIPDC - Skip Rest of Statement Decompile

Category: DCMUTL File: SG&LDC::MS

Name:(S) SKIPDC - Skip Rest of Statement Decompile

Purpose:
When an unrecognized token is encountered, decompile of that statement cannot continue. SKIPDC skips D1 to the end of that statement.

Entry:
(INADDR) = Address of the statement length byte of the statement currently being decompiled.

Exit:
D1 points to the statement terminator byte in the token stream.
Exit is via OUTEL1.
A(A)= Statement Length for the statement skipped.
C(A)= D1

Calls: None

Uses: A(A), C(A), D1

Stk lvls: 0

Detail: Must immediately follow FILDC

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/08/83 | S.W. | Added documentation header |

## 7.16 LISTDC - Decompiles LIST, RENUMBER, SECURE, MERGE

Category: DCMUTL File: SG&LDC::MS

Name:(S) LISTDC - Decompiles LIST, RENUMBER, SECURE, MERGE

Purpose: DECOMPILES LIST, SECURE, MERGE STATEMENTS

Entry: P= 0
D1 past begin BASIC token
D0 output pointer
D(A) contains available memory end (AVMEME)

Exit: via OUTELA

Calls: FILDC, LINNDC, EOLXC*, COMTST, OUTBYT

Uses: A-C, D1,D0, S8,S9, R1,R2
A-C, D1,D0, R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Detail: EXPECTS THAT S8 WILL BE CLEAR UPON ENTRY

History:

| Date | Programmer | Modifications |
|---|---|---|
| 08/29/83 | S.W. | Added documentation header |

## 7.17 PRT#DC - Port# Decompile

Category: DCMUTL File: SG&SYS::MS

Name:(S) PRT#DC - Port# Decompile

Purpose:
Decompiles a port number

Entry:
P = 0
D(1)= Port#, D(0)=Extender#
D0 positioned for output (Next 10 nibs blank-filled)

Exit:
P = 0
D0 incremented by 10 (past trailing blank)

Calls: HEXDEC, CAT$70

Uses.......
Inclusive: A,B,C,P,D0

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/13/83 | S.W. | Added documentation |

## 7.18 FTYPDC - File Type Decompile

Category: DCMUTL File: SG&SYS::MS

**Name:(S) FTYPDC - File Type Decompile**

**Purpose:**

Decompiles File Type

FTYPD+ checks to ensure there's enough memory to output decompiled file type.

FTYPDC assumes there's enough memory.

**Entry:**

D0 past a blank (pointing to output buffer)
D1 pointing at 4 nibble file type#
2 ENTRY POINTS:
1) FTYPD+ - D(A) = AVMEME
2) FTYPDC - P=0

**Exit:**

5 character file type written to where D0 pointed;
D0 past outputted file type; D1 as it was upon entry
Carry clear
P=0

**Calls:** FTYPFD, CAT$90, CAT$95, OUTNBS, RDENTY

**Uses:** A-C, D0, R0, P

**Stk lvls:** 3

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 10/21/82 | S.W. | A=0 W <= A=0 A |
| 06/10/83 | S.W. | Call CAT$95 to output '-' |

```
+---------------------------------------------+-------------------+
|                                             |                   |
|  EXCUTL  -  Execute Utilities               |  CHAPTER  8       |
|                                             |                   |
+---------------------------------------------+-------------------+
```

## 8.1 SVTRC - Save Trace Information In Stmt Scratch

Category: EXCUTL     File: AB&ASN::MS

Name:(S) SVTRC - Save Trace Information In Stmt Scratch

Purpose:
Save trace information in stmt scratch.

Entry:
D0 = trace information.

Exit:
Copy of information in C[A].
Information saved in S-R1-2.

Calls: None.

Uses.......
C[A]

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |

## 8.2 EXPEXC - Evaluate Expression

Category: EXCUTL     File: AB&EXP::MS

```
Name:(S) EXPEXC  -  Evaluate Expression
Name:    EXPEX1  -  Evaluate Expression
Name:(S) EXPEX-  -  Evaluate Expression
Name:(S) EXPEX+  -  Evaluate Expression

Purpose:
     Initiate evaluation of an expression.

Entry:
     HEX mode.
     D0 pointing to start of expression.

Exit:
     Carry clear.
     D1 pointing at top of mathstack, which contains
       whatever results the expressions put there.
     D0 pointing past expression.
     A[W] = 16 nibbles at top of stack (**result if this
       is a REAL numeric expression).
     If the last item in the expression was a variable,
       information is left in certain registers for use
       by the DEST routine.  See the documentation for
       DYNAMC and STATIC in this module.

Calls:       COLLAP, GETST.  Exits through EXPR.

Uses.......
             Everything available to functions:
             ALL CPU REGS, Function Scratch, SCRTCH,

Stk lvls:    4 (4 levels available to functions invoked)

Note:
     EXPEXC and EXPEX1 are different names for same entry
     point.

Algorithm:
  EXPEX-: Collapse mathstack to forstk.
          Goto expexc.
  EXPEX+: Save CPU status bits in STSAVE.
  EXPEX1:
  EXPEXC: D1 = (MTHSTK).
```

Go to EXPR (i.e., evaluate expression).

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 8.3 FNRTN1 - Function Return

Category: EXCUTL    File: AB&EXP::MS

Name:(S) FNRTN1 - Function Return
Name:(S) FNRTN2 - Function Return
Name:(S) FNRTN3 - Function Return
Name:(S) FNRTN4 - Function Return
Name:(S) EXPR - Function Return

Purpose:
Return to expression execution controller after evaluation of a function or operator.

Entry:
```
  FNRTN1: D0 = PC.
          D1 = stack pointer.
          Number to be pushed on stack in C[W].
  FNRTN2: A[A] = PC.
          D1 = stack pointer.
          Number to be pushed on stack in C[W].
  FNRTN3: A[A] = PC.
          D1 = new stack pointer (pointer already
            decremented for storing result and stack
            collision check already performed).
          Number to be pushed on stack in C[W].
  FNRTN4: D0 = PC.
          D1 = new stack pointer (pointer already
            decremented for storing result and stack
            collision check already performed).
          Number to be pushed on stack in C[W].
  EXPR:   D0 = PC.
```

D1 = stack pointer.
Result has already been put on stack.

Exit:
Continues evaluation of expression. Returns to whomever called expression execution controller when expression is done.
Return conditions at that time:
Carry clear.
D1 at top of stack.
D0 = PC, is past expression.

Calls: None.

Uses.......
Everything available for functions.

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | MM | Attempted to document |

## 8.4 OUTRES - Round And Return Result

Category: EXCUTIL File: AB&FCN::MS

Name:(S) OUTRES - Round And Return Result

Purpose:
Round result according to IEEE rounding rules, put on mathstack and reenter expression execution controller.

Entry:
Result in (A,B), SB, XM and P as per uRES12 entry conditions.
D1 = top of math stack.

Exit:
Through EXPR.

Calls: uRES12.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |

## 8.5 LIMITS - Compute Dimension Limits In Decl Stmt

Category: EXCUTL File: AB&REG::MS

Name:(S) LIMITS - Compute Dimension Limits In Decl Stmt

Purpose:
Compute the dimension limits in a declaration statement (INTEGER, REAL, SHORT, DIM). Collapses the stack beforehand.

Entry:
D0 pointing at start of tokenized expression.

Exit:
D0 pointing past expression.
D1 @ top of math stack.

Calls: COLLAP, EXPEX+.

Uses.......
Everything available to expression execute.

Stk lvls: 5

History:

| Date | Programmer | Modification |
| --- | --- | --- |

| | | |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 8.6 HASH1 - Indexed Jump Through A GOTO Table

Category: EXCUTL File: AB&UTL::MS

Name:(S) HASH1 - Indexed Jump Through A GOTO Table
Name:(S) HASH2 - Indexed Jump Through A GOTO Table

Purpose:
Jump into a table of GOTOs (or other 4-nibble beasts) according to an index variable.

Entry:
A[A] = Hash byte (maximum 3FF).
HASH1: RSTK = Address of start of GOTO table.
HASH2: C[A] = Address of start of GOTO table.

Exit:
This routine exits by jumping to the A[A]'th entry in the GOTO table.

Calls: None.

Uses.......
A[X], C[A].

Stk lvls: HASH1: 0.
HASH2: 1.

Detail:
Typical use:

```
GOSBVL =HASH1
GOTO   L0
GOTO   L1
GOTO   L2
GOTO   L3
GOTO   L4
GOTO   L5
GOTO   L6
```

The GOSBVL puts the address of the GOTO table on RSTK; HASH1 peels it off. Note that this GOSBVL is actually acting like a GOTO; control will never return to the code in the vicinity of the GOSBVL.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 8.7 TRSFMu - Transform Utility Routine

Category: EXCUTIL File: FH&TFM::MS

Name:(S) TRSFMu - Transform Utility Routine

Purpose:
Transform a file using source/dest file info on Save Stack.

Entry:
```
P       =  0
/DFTYP  =  Destination file type
Save stack info set up by SVINFO as by COPYX or TRSFMX
```

Exit:
```
P        =  0
C(1-0)   =  Transformation option
C(6-2)   =  Dest file creation first parameter
C(11-7)  =  Dest file creation second parameter
Save Stack info cleared from Save Stack
Carry clear:
Transform completed successfully
Carry set:
C(3-0) =  Error code.  "Syntax" if all errors were
          recoverable.
```

**Calls:** FILFIL, EXPDEV, RDINFD/S, SVINFO, OPENF*, FINDFS, CRTF, RAM/OM, MEMCHE, POLL, PRGFIN, LOCFI+, RPLLIN, WRITNB, and a host of local utilities

**Uses.......**
**Inclusive:** All CPU registers, statement and function scratch, TRFMBF, S11-S0

**Stk lvls:** 6 (RSTKBF: 1 plus any used by handlers)

**Detail:**

**Status Used:**

| Val | Phase | Name | Meaning |
|---|---|---|---|
| (0) | ( 1 ) | sEXTDV | Source or destination is on HPIL device. |
| (0) | ( 2 ) | sTFREQ | If set, a transform is required. Otherwise it is a trivial case (file is already desired type). |
| (1) | ( 1 ) | sUNDEF | Indicates both file names are undefined. |
| (1) | (3,4) | sTFERR | If set, a fatal error has occurred somewhere in the transform. User will be notified at end. |
| (2) | ( 1 ) | sCARD | Indicates a card device on source or dest. |
| (2) | (3,4) | sTFWNG | If set, a recoverable error has occurred during the transform. This will become a fatal error after the transform is complete. |
| (3) | (all) | sDEST | If set, we are accessing the destination file info on the save stack. |
| (4) | (all) | sREADI | Used by RDINFO |
| (5) | (234) | sTFINP | If set, transform is in place. Else, the source and dest files are distinct. |
| (6) | ( 3 ) | sTFINV | If set, we are doing an inverse transform. A nonrecoverable error at this point will cause the file to be purged. |
| (7) | (all) | sEOF | If set, an EOF has been read on input operation. |

(8) ( 3 ) sTFEND  If set, the last line has been transformed. Must be used because sEOF can be set BEFORE the EOF is transformed.

(9) (234) sDRYRN  If set, transform results are not written to dest, but used to calculate required size of dest file.

(10)(all) sI/OBF  If set, last referenced opened file is external.

(11)(234) sPRGCF  If set, the transformed file was current file or was referenced on the stack (e.g., CALLer of current file)

Algorithm:

```
Save return address
Initialize FIB storage and status to zero
Fill in missing file names (e.g., :TAPE INTO TEXT A)
If either filename undefined, then error exit
Expand destination device code
If dest device not specific, then error exit
Open source file (exit if error)
Save away source FIB#
Build expanded source device code
Save away source file type
Clear status
If source device = dest device
If source name = dest name
   Set "Transform IN PLACE"
      Store source FIB# as dest FIB#
      If file is secure, then
         Error exit
If dest device is HPIL then
   If "Transform IN PLACE" then
      Error exit
Else
   If dest device is not RAM
      Error exit
Find transform handler
If no handler, then
If transform required, then
   Error exit
Else
   If transform "In place", then
      Return
   Else
      Copy file to destination using COPYu
If "Transform IN PLACE" then
   If there is no inverse transform then
      Error exit "Illegal Transform"
```

```
          Release file info, clear sDRYRN
          If file is current file then
             Close it out and open new workfile
       Else
          If dest file is external then
             Set "Dry Run"
          Else
             Search for dest file
             If file found, then
                Error exit "File exists"
             Else
                Create dest file
                Open dest file
                Store away dest file FIB#
                Release file info
       Initialize counts: NUMLINES, DESTLEN
       Save away true AVMEME
   3.1 Set up default output buffer
       Save status
       Verify minimal memory requirement
       Call Transform routine
       Restore status
       If Error, then
          If recoverable, then
             Issue warning message
             Set "Warning" status
       If no error, then
          If "Dry Run" then
             Adjust destination length counter
             If at EOF then
                Create dest file
                Open dest file
                Store away dest FIB #
                Clear "Dry run" status
                Rewind source file
             Go to 3.1  [go to next line]
          Else
             Read dest FIB
             If transform NOT IN PLACE, then
                Set old line length to zero for insertion
             Call WRITNB to copy output buffer to dest file
             If fatal error, then
                Go to 3.5
             Else
                Go to 3.1          [go to next line]
       Else {error}
          If recoverable error, then
             If in "Dry run" then
                Go to code sequence to process line
          Else
             Set error flag
```

```
            Save error code
            If in "Dry run" then
               Go to 4.0
            If memerr, substitute "Transform Failed" message
         Issue warning message
         If recovereble error, then
            Set warning flag
            Go to code sequence to process line
         Else [It's an unrecoverable error]
            If INVERSE transformation, then
               Save "Transform failed" error massage
               Go to 3.9
            Else
               If transform IN PLACE, then
                  Set "Inverse Transform" status
                  Rewind source file
                  Set up inverse transform address
                  Go to 3.1
   3.7 Collapse input, output buffer
      If Dry Run then
         Fetch Source FIB
      Else
         Fetch Dest FIB
         If NOT inverse transformation, then
            Truncate file to current position
      Rewind file, save status
      Call Chain Handler on file
      Call TFUSVE to hold error code, restore status
      If Dry Run, then
         Play it again, Sam
      If error, then
         Set error code
   3.9   Purge destination file
   4.0 Restore return address
      Read source FIB#
      If source FIB# not zero then
         If transform IN PLACE then
           Add file type and copy code to header
         Close source FIB
      Read dest FIB#
      If dest FIB# not zero then
         Close dest file
      If not "Fatal error" status, then
         If not "Warning" status, then
            Exit successfully
         Else
            Set "Syntax" error code
      Issue error code message
      Exit with error condition
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/20/82 | FH | Split off from TRSFMX code |

## 8.8 COPYu - COPY Utility

Category: EXCUTL File: JP&EXC::MS

Name:(S) COPYu - COPY Utility

Purpose:
    COPY Utility
    COPY Mainframe/PORTs
    COPY CARD
    COPY External

Entry:
    File information in SAVSTK area
    (Through SVINFO utility)

    SAVSTK- 5 = Source Device Information
    SAVSTK-25 = Source Filename
    SAVSTK-30 = Destination Device Information
    SAVSTK-50 = Destination Filename

    See SVINFO utility
        Device Info - Nib 0 = Device type
                      Nib 1-4 = Device specific info
        Filename      Up to 10 chars
                      Blank filled

Exit:
    Save area is RELEASED

    Carry clear - Good COPY
      R1 = Start of file just copied
           If destination into Mainframe/IRAM

    pCOPYx Poll issued if either Source or Destination

```
                is External (not Mainframe/IRAM/CARD)

        Carry set - Error Return
          C(4) = Error number

          Error Returns:

          eMEM      - No memory to create destination file
          eFSPEC    - No response to COPY Poll
                      Non Mainframe for CARD
          efnFND    - Source file not found
          eFPROT    - Private source file
          eFEXST    - Destination file exists
          eFTYPE    - Non KEYS file for KEYS copy
          eFACCS    - Destination is unknown PORT device
          eDVCNF    - PORT device not found

  Calls:       FILFIL,POLL,FINDF (FINDFS),GETPR1,MFDVC+,RDHDR1,
               CRETF+,MOVEUO,RDINFD,WFIND-,LEXBF+,RLINFO,CRDFIL,
               FILCRD,CHAIN-,BASCHA,FLDEV+,MFDEVC,D1=SRO

  Uses.......
   Exclusive: A-D,D0,D1,SAVSTK (50 nibs),R0,R1,R2,
              S0-S7,S8,S9,S12,STMTR0 (5 nibs),SCRTCH (32 nibs)
              4 levels of RSTKBF (if Copying LEX file)
                                    See LEXBF+

   Inclusive:

       sDEST  = Destination Execute flag  (S3)
       sREADI = Read file information     (S4)
       sKEYS  = COPY to KEYS              (S5)
       sPCRD  = Private CARD              (S8)
           D  = Device information
                D(0) = Device Type
                       F = No device
                dMAIN  0 = :MAIN
                dPORT  1 = :PORT
                            D(1,2) = Extender#, Port#
                                   = FF if all ports
                dCARD  7 = CARD      D(B)=0
                dPCRD  7 = :PCRD     D(B)#0
                    >= 8 = PIL / Non-mainframe Device

         R1  = Destination file start     (CREATF)
         R3  = Start of source file


            POLL   uses B,C,AVMEME,XM
            FINDF  use  A-D,D0,D1,S6,S8,S9,R2,R3
            MOVEUO uses A,C,D0,D1,P
```

```
          RDINFO
          SVINFO use  A,C,D,S3,S4
          GETPR1 uses A,C,D1
          CREATF uses A-D,D0,D1,R0,R1,SCRTCH (32 nibs)
          tKYSck uses A,C,D(S)
          LEXBF+ uses A-D,R1,4 lvls RSTKBF

 Stk lvls:   6

 Algorithm:

 COPYu:
     Get source information, Fill missing names (FILFIL)
     If device # Mainframe        (sEXTDV) or ext PORT
 0:     POLL for COPY external device      (pCOPYx)
        If carry set
           Error Return
        If no response
           If external device              (D(0)>7)
              Error <-- eFSPEC
           If unknown PORT device
              Error <-- eFACCS
        else
          RTN

 1:  If MAIN | PORT            (sCARD=0)
        If source
           Find source file                (FINDF)
           Save pointer to file start      (R3)
           Check file protection           (GETPR1)
             Error Return if private       (eFPROT)
           If BASIC file                   (BASCHA)
              Chain file                   (CHAIN-)
           Set Destination flag
           Get Destination device info     (MFDVC+)
           If CARD
              go COPY to CARD              (goto 5)
        else  (destination)
           If "keys" filename
               Set KEYS File flag
           Save source start               (STMTR0)
           If PORT destination not found   (FLDEV+)
              Error                        (eDVCNF)
           If Not Mainframe destination
              Convert Dest. filename to Uppercase (CVUCW)
              Save updated Dest. File infor (SVINF+)
           If not Independent RAM or MAIN
              go Poll for COPY to unknown dev (goto 0)
           Find destination file           (FINDF)
           If file found
              Error Return                 (eFEXST)
```

```
              Restore source start
              Read Source file header           (RDHDR1)
              If KEYS Copy                      (sKEYS)
                 If source file type # KEYS
                    Error Return
              Compute file length
              Create Destination file           (CREIF+)
                Error if not created           (Carry set)
              Copy source to destination        (MOVEU3)
              Read destination information      (RDINFD)
              Write new filename
              Write new creation date & time
         4:   If LEX file copy   (Dest. filetype = LEX)
                 Save file start (R1 --> RSTK)
                 Regenerate LEX Buffer          (LEXBF+)
                 Restore file start             (R1)
                 goto Done;

        If CARD | PCRD device
           If source
              Set destination flag
              Read destination device           (MFDVC+)
              R3 <-- Source Filename
              R2 <-- Destination filename
              If destination device = MAIN
                 Copy CARD to File              (CRDFIL)
                 Set R1 = Last file in Mainframe(EOFLCH)
                 Position to File type
                 go Check if Lex File copy      (goto 4)
              else
                Error Exit                      (eFSPEC)
     5:    If destination = CARD
              If Private Card                  ((D(1-2)#0)
                 Set Private Card flag
              If source device = MAIN | PORT
                 R1 <-- Destination Filename
                  C <-- Source file start
                 Copy file to CARD              (FILCRD)
              else
                 Error Exit                     (eFSPEC)

  Done:  Release File Informatin Save area      (SRLEAS)
         Return CC

  CPYERR: Save error message on stack
          Release File information save area
          Restore error message
          Return SC

  History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |
| 11/20/82 | JP | Fixed COPY TO CARD |
| 12/18/82 | JP | Combined pCOPYd with pCOPYx |
| 12/18/82 | JP | Added chain source if BASIC |
| 03/21/83 | JP | Test if PORT not found after FLDEV+ |
| 03/21/83 | JP | Using S-R0-0 to save Source start |
| 05/11/83 | JP | Packed CVUCW,SVINF+ calls @ CPY135 |

## 8.9 CK"ON" - Check ON / ATTN Key

Category: EXCUTL File: JP&SYS::MS

```
Name:(S) CK"ON"  -  Check ON / ATTN Key

Purpose:
     Check if ON/ATTN key hit    (CK"ON" entry)
     This routines needs to be called after
          each statement execute

Entry:

Exit:
     Carry set
       ATTN key Not hit
     Carry clear
       ATTN Key hit
       NoCont (S14) set if ATTN key hit

Calls:      None

Uses.......
 Exclusive: A(S),D1,NoCont(S14)
 Inclusive: A(S),D1,NoCont(S14)

 S14 = ATTN key hit, NoCont flag

Stk lvls:   0
```

## 8.10 FINDLB - Find Label in Current Program

Category: EXCUTL File: JP&SYS::MS

```
Name:(S) FINDLB  -  Find Label in Current Program
Name:    ATCHK   -  Find Label in Current Program

Purpose:
     Find Label in current program.  This routine is for run
     time only.  To find a label across a file call FCHLBL.

     ATCHK: Late entry point to check if at an "@"

Entry:
     FINDLB:
       P=0
       B = Label to find
           Right justified with trailing blanks
             ("ABC" = 2020202020434241)
       File already chained

     ATCHK:
       DO @ Possible "@" (multi-statement line)

Exit:
     FINDLB:
       P=0
       B = Label to find
       Carry Clear - Label found
         DO @ EOL or @ preceding the statement with Label
       Carry Set   - Label not found
     ATCHK:
       DO @ "@" or EOL

Calls:       LBLNAM

Uses.......
 Exclusive: A,C(A),DO
 Inclusive: A,C,DO,P

Stk lvls:   2

Detail:   Starting from label chain head (PRGMEM-5)
```

```
            Jump by Label Link looking for LABEL token
            When a LABEL token is found
                Call LBLNAM to get label into A
                If label matches the label in B
        ATCHK:      Position to EOL | @
                    Return CC
                else
                    Continue until End of Label Chain reached
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |
| 04/08/83 | JP | Test for @/line# using A(XS) = F |

## 8.11 LBLNAM - Get Label Name into Register A

Category: EXCUTL  File: JP&SYS::MS

Name:(S) LBLNAM - Get Label Name into Register A

Purpose:
Get label name into Register A

Entry:
DO  @ Beginning of Label in Memory

Exit:
Carry clear
P = 0
A = Label name, Right justified with trailing blanks

"ABC" = 2020202020434241 (hex)

Calls: BLANKC

Uses.......
Exclusive: A,C,P
Inclusive: A,C,P

Stk lvls: 1

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/30/82 | JP | Modified documentation |
| 10/08/82 | JP | Added BLANKC call |

## 8.12 PRSCOP - Compute Program Scope

Category: EXCUTL File: JP&SYS::MS

```
Name:    PRSCOP  -  Compute Program Scope
Name:    PRSCKB  -  Compute Program Scope; Return if SUSP
Name:(S) PRSCOO  -  Compute Program Scope; GETSTC exit cond

Purpose:
  Compute Program Scope:Program Start,Program End,Sub Links

Entry:
     Assumes: CURRST, CURREN pointing at current file

     PRSCKB:  If program suspened  --> Return
              P=0
     PRSCOP:  Calls GETSTC to position in file and
                check file type
              Error Exits if non-BASIC file
     PRSCOO:  Assumes positioning = GETSTC exit conditions
              File type must be BASIC;Binary or Same structure
              P=0
     PRSCO-:  Get program start/end w/o File Type error exit
              Allows Program scope set for Binary programs
     PRSC60:  Set Program Start and End only
              D1 @ PRGMST

Exit:
     If program already running on entry:
        This routine does nothing
```

```
        If PRSCOP entry:
           If current file not BASIC
              Error Exit ---> eFTYPE

        A   = PRGMST  (Program Start)
        C,D = PRGMEN  (Program End)
        D1  @ PRGMST
```

Calls: GETSTC,GETSTe,CHAIN*,RUSUS?,SCOPEN

Uses.......
 Exclusive: A(A),B(A),C(A),D(A),D0,D1,R2
 Inclusive: A,B(A),B(S),C,D(A),D0,D1,R2

Stk lvls: 3

NOTE:
  PRSCKB will not set program scope if running or suspended
  PRSCOP will always set the program scope if program not
    running

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |
| 09/15/82 | JP | Changed GETSTC to error return |
| 01/04/83 | JP | Added PRSCKB entry point |
| 02/11/83 | JP | Deleted PRSC55 entry point |

## 8.13 CkLoop - IMAGE parse loop to check for edit chars

Category: EXCUTIL     File: MB&IMG::MS

Name:(S) CkLoop - IMAGE parse loop to check for edit chars
Name:(S) CkLpNC - IMAGE parse loop, no symbol count

Purpose:
     This is the main parsing routine for IMAGE parsing.
     It first accepts spaces and multipliers in the image

string, then parses the next image character for correct syntax.

Entry:

```
P        = 0
R1(A) points to last IMAGE symbol which was parsed
D1 points to current position in BldIMG stream
D(A)=AvMemEnd
Return address contains a four-nibble mask used
  to parse the next character (see USING header)
```

Exit:

```
CkLoop does not return!  It jumps to
    1) IMerr (if multiplier=0 is found)
    2) CkDlim (if no match found in parse table)
    3) To appropriate parse routine if match found
        (these routines are fixed in the parse table;
         they cannot be added to)
CkLoop leaves the RSTK in a mess...  The parse mask
address is left in the RSTK (no problem, since USING
can never be called as a subroutine).
```

Calls:      IMmlt+, PRSsc+, DRANGE, TBLJMP

Uses.......
 Exclusive: A,B,C,D,D0,D1,P
 Inclusive: Can use anything when exits to parse handlers

Stk lvls:   3 (before exit to parse handler)

NOTE:

This parser is used only for the following IMAGE symbols, with the corresponding parse handler routines:

| | |
|---|---|
| X | IM"X" |
| D | IM"D" |
| A | IM"A" |
| . or R | IMrdx |
| " or ' | IMstr |
| S or M | IMsign |
| Z | IM"Z" |
| E | IM"E" |
| C or P | IMsep |
| * | IM"*" |
| unit's digit Z | IM1"Z" |
| H,K,B or ^ | IMHKB^ |

Only those symbols included in the parse mask (found at the RSTK address) will be accepted. Any other character will cause a jump to CkDlim; if the character is not a delimiter, CkDlim will issue a pIMCHR poll.

Algorithm:
```
  CkLoop: Increment digit symbol count (in R2(A))
  CkLpNC:
    1)    Fetch next IMAGE character, convert to uppercase
          Check for digit (DRANGE); if not digit, goto 3).
          Else (digit) if digit already found (sMULT=1)
             goto 2).
          Else (digit not found yet) set sMULT=1,
             write out multiplier fields to BldIMG.
    2)    Check for digit overflow (more than 4 digits),
             error if overflow.
          Write out current multiplier.
          Goto 1).
    3)    Check char for ASCII space. If so, goto 1).
          If multiplier pending, test for:
             if mult= 0, then error.
             if mult= 1, then ignore (back up over
                  multiplier fields)
          Fetch parse mask from RSTK address.
    4)    Read next character from fixed parse table.
          If end of table, jump to CkDlim.
          Check mask bit for valid char; if not,
            go to 4).
          Else (valid char), compare with IMAGE symbol:
            if no match, go to 4)
            else (match), jump to parse handler for
              that symbol
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Wrote routine, documented. |

## 8.14 BOPNM- - Process uOPNM- token during backup

Category: EXCUTIL File: MB&IMG::MS

Name:(S) BOPNM- - Process uOPNM- token during backup

Purpose:

To process uOPNM- token during IMAGE parse backward search.

Entry:

P = 0
D1 points to uOPNM- token in BldIMG stream
R0(A)=current position in BldIMG stream (any new token will be written below this address)
R1(A)=address if symbol which caused backward search: a right parenthesis (to close a field), or the end-of-image (to check for unmatched parentheses).
S5=1 if end-of-image search; S5=0 if closing field.

Exit:

P = 0
Carry clear.
A uLOOPP token, a 5-nibble offset pointing to the left parenthesis location, and a uJMP() token will have been written to the BldIMG stream.
D1=current position in BldIMG stream (address passed in R0(A) minus 9)

Calls: COPYm1, EndBck, IMoffs, BldIMG

Uses.......
Exclusive: B(A),C,D1,P
Inclusive: B(A),C,D1,P,S8

Stk lvls: 1

NOTE:

This backward search during IMAGE parsing is performed to find an open field (a field defined by parentheses). The search is performe either to close the field (when a right parenthesis is found), or to check for unmatched parentheses at the end-of-image.

Algorithm:

Set B(A)=1 (for COPYm1)
If S5=1 ("end-of-image"), report "Invalid IMAGE" error.
Copy multiplier from reserve field to decrementer field (adding one to the reserve, from B(A))
Write uLOOPP token to BldIMG
Compute offset to left parenthesis position, store it in BldIMG
Write uJMP() token to BldIMG.

History:

| Date | Programmer | Modification |
|---|---|---|

12/08/82 MB Documentation

## 8.15 IMinit - Initiate IMAGE output field

Category: EXCUTL File: MB&IMG::MS

Name:(S) IMinit - Initiate IMAGE output field
Name: IMin01 - Backup to field delimiter (close field)

**Purpose:**
To back up through the BldIMG token stream to the pending delimiter and re-write a field delimiter, in order to identify the type of field for the execution routines.

**Entry:**
P = 0
C(B)=new delimiter token (see detail, below)
D1=current position in BldIMG stream (any new tokens will be written below this address)
A(B)=IMAGE symbol which caused the initialization (in uppercase)
D(A)=AvMemend
S3=0 if field has not already been initialized;
S3=1 if field has already been initialized.

**Exit:**
If pending fields need to be executed (S0=1), then exits to IMGxqt.
Else,
- P=0
- Carry clear
- D1=current position in BldIMG stream
- C=address of delimiter token
- Delimiter token has been re-written to identify new field.

**Calls:** D12R0A, BACK, CSL9R0
FPOLL (pIMcpi) if S7=1
IMGxqt if S0=1.

Uses.......
 Exclusive: A,B(A),C,R0(A),R2
            IMinit also uses S0,S2,S3,S10
 Inclusive: If S0=0: A,B,C,R0(A),R2
            If S0=1: can use anything in execution routines.

Stk lvls: 3 (unless S0=1: execution routines can use 7)

NOTE:
Whenever a new field begins, a delimiter token (uDELIM) is written to the BldIMG stream, along with two 4-nibble fields used for digit counters. Also, S3 is set=0 to indicate that the field has not yet been initialized (type of field not yet discovered). IMinit is called whenever an output character is found; if S3=1, it returns immediately. Otherwise, S3 is set=1, and the BldIMG tokens are scanned (backwards) until the uDELIM token is found. It is then replaced with the appropriate token to identify the type of field.

However, if pending fields need to be executed (S0=1), the token is replaced with a uRESTP (restart parse) token, and IMGxqt is invoked to execute the fields.

IMin01 is called to find the field delimiter at certain times, for the following actions:
1) when a radix symbol (. or R) is found, one of the 4-nibble counter fields is filled with the number of digits before the radix
2) when a numeric field ends, the other 4-nibble counter field is filled with the total number of digit symbols.
3) when a sign symbol (S or M) is found, the field delimiter is adjusted to indicate that a sign is specified.
4) when the E symbol is found, the field delimiter is replaced with one which indicates that the exponent is to be displayed.

At these times, S0=0 so that execution will not start.

Fast poll for pIMcpi may change S0, or the flag in R2(XS) (see C(XS) detail below), if necessary.

Detail:
At entry to IMin01, C(XS) is used as a flag to indicate whether to re-write the delimiter. In cases (1) and (2) above, the field delimiter is not overwritten; in these cases, C(XS) is nonzero as a flag.

At entry to IMin01, C(B)=new delimiter token to re-write, or C(B)=0 if delimiter merely has to be adjusted (case (3) above).

Algorithm:
```
  IMinit: Set S2=1 ("count digits")
          If S3=1, return. ("Field already initialized")
          Set S3=1, S0=1 ("execute pending fields"),
            S10=1 ("output field found")
          Set C(XS)=0 (flag for "re-write delimiter")
          Save symbol in R2.
  IMin01: Save D1 in R0.
          If S7=1 and S0=1, fast poll (pIMcpi)
     1)   Back up through tokens:
            if uJMPst, then D1+12, go to 1)
            if uJMPdl, then D1+6, go to 1)
            if uDELIM, then go to 3)
            if other delimiter, go to 4)
            if uRESTP, then go to 2)
            else go to 1)
     2)   Set S0=1 (don't execute)
          Copy D1 to R0(9-5) (new execution address)
     3)   Clear R2(A) (digit count)
          If S0=1, jump to IMGxqt: re-write delimiter
            with uRESPT token and execute pending fields.
          If "don't re-write delimiter", go to 5)
          If "write new token", go to 4)
          If S9=1 ("sign"), then increment delim+1
     4)   Re-write delimiter
     5)   Restore D1 from R0(A).
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.16 BldIMG - Put tokens from C into BldIMG stream

Category: EXCUTL File: MB&IMG::MS

Name:(S) BldIMG - Put tokens from C into BldIMG stream
Name:(S) BldIMA - Put 1 or 2 tokens from A into BldIMG

Name:(S) BldIM+ - Put tokens from C into BldIMG stream

Purpose:
To put IMAGE tokens into parse stream.

Entry:
```
BldIMA:   A(B)=token and P=0
       or A(3-0)=2 tokens and P=2
BldIMG:   C=tokens and P=2*(#tokens-1)
BldIM+:   C(WP)=tokens and P=2*(#tokens)-1
         D1=current position in BldIMG stream
         D(A)=AvMEmSt
```

Exit:
```
P     = 0
Carry clear
Exits to MEMERR if D1 moves below AvMemSt
```

Calls: none

Uses.......
Exclusive: P,D1 moved below write
BldIMA: also does ACEX A

Stk lvls: 0

NOTE:
The "BldIMG" stream refers to the token stream used for IMAGE execution. This routine can be used by any code which needs to write bytes or nibbles to Available Memory.

Exampl: for entry into BldIMG, say C(7-0) contains 4 tokens. Then enter with P=6.

Detail:
```
=BldIMA ACEX   A
=BldIMG P=P+1
=BldIM+ C=-C   A
        C+P+1
        C=-C   A
        ?C<=D  A
        GOYES  MEMERR
        CD1EX
        DAT1=C WP
        P=     0
        RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|

```
--------   ----------   -----------------------------------
12/08/82   MB           Documentation
```

## 8.17 IMoffs - Store offset from D1 in BldIMG stream

Category: EXCUTL     File: MB&IMG::MS

```
Name:(S) IMoffs  -  Store offset from D1 in BldIMG stream

Purpose:
     Store a 5-nibble offset from D1 in the BldIMG stream.

Entry:
     P= at least 4.  If C(15-5) contains more tokens to
       write into the BldIMG stream, then set P such that
       a P=P+1 will define the entire write field in C(WP).
     C(A)=address-2 for which offset will be computed.

Exit:
     P      =  0
     Carry clear

Calls:      BldIMA

Uses.......
 Exclusive: C(A)
 Inclusive: P,D1 (does not use A)

Stk lvls:   0

Detail:
    =IMoffs AD1EX
            C=C-A  A
            AD1EX
            C=C+1  A
            C=C+1  A
            ACEX   A
           <falls into BldIMA>

History:
```

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 8.18 PRSscn - IMAGE parse scan

Category: EXCUTIL File: MB&IMG::MS

Name:(S) PRSscn - IMAGE parse scan
Name:(S) PRSsc+ - IMAGE parse scan, increment D0 first

Purpose:
Read a byte from address in R1(A), scan a table of values for a match. If match found, jump to corresponding routine.

Entry:
P = 0
R1(A)=address of byte to match
Address in RSTK points to table of bytes and relative offsets (see FINDA for table structure)

Exit:
P = 0
Carry clear
Exits to desired routine if byte match. If no match, returns to address past table.

Calls: CONVUC, FINDA

Uses.......
Exclusive: C(W),D0,A(B)
PRSsc+ also increments R1(A) by 2.
Inclusive: C(W),D0,A(B)

Stk lvls: 2

NOTE:
The byte from the address found in R1(A) is read into A(B) and converted into upper case before the jump to FINDA.

See FINDA for description of table of bytes and offsets.

Detail:

```
=PRSsc+ GOSUB  IMDO+2       Increment R1(A) by 2.
=PRSscn C=R1
        DO=C
        A=DATO B
        GOSUBL =CONVUC      Convert to upper case.
        GOVLNG =FINDA
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.19 IMxq27 - Return to IMAGE token executor

Category: EXCUTL  File: MB&USG::MS

Name:(S) IMxq27 - Return to IMAGE token executor

Purpose:
Return to IMxq12 (main IMAGE token execution routine) after restoring D1 (token pointer).

Entry:
C(A)=address+2 of next IMAGE token to execute.
S5=0
S6=0

Exit:
May jump to any execution routines.

Calls: May jump to any execution routines.

Uses.......
Inclusive: May jump to any execution routines.

Stk lvls: May jump to any execution routines.

NOTE:
Some IMAGE poll handlers will use this entry point after handling a poll. Since the FPOLL routine does not preserve D1, this allows a poll handler to jump to the IMAGE token executor with D1 pointing to the appropriate token.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 8.20 USst03 - Output characters from address in C

Category: EXCUTL File: MB&USG::MS

Name:(S) USst03 - Output characters from address in C
Name:(S) USst05 - Output characters from address in D1

Purpose:
To output a character during USING execution; character display observes WIDTH.

Entry:
USst03: D1=address of current token being executed
C=address of characters to be output
USst05: A=address of current token being executed
D1=address of characters to be output
P=0
B(A)=#characters to output
CKINFO must have been called previously to set up the output information (see CKINFO)
S5=0 to exit to IMxq12, S5=1 to return.

Exit:
P = 0
If S5=0, exits to IMxq12
If S5=1, does a "return", carry clear.

Calls: SENDWD

```
Uses.......
 Exclusive: A,B,C,R0,S4
 Inclusive: A,B,C,D,R0,R1,R2,P,S4,D1

Stk lvls:   5

NOTE:
     If you want to display only one character, call USGch+

Detail:
     Before call to SENDWD, sets S4=0 to inhibit EOL before
     item is displayed.

    =USst03 AD1EX
            D1=C
    =USst05 R0=A
            A=B    A
            ST=0   4
            GOSBVL =SENDWD
            C=R0
            D1=C
            ?ST=0  5
            GOYES  IMxq12
            ST=0   5
            RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.21 USGch+ - Display character during USING execution

Category: EXCUTIL     File: MB&USG::MS

Name:(S) USGch+ - Display character during USING execution
Name:(S) USGch- - Display character during USING execution

Purpose:
To display one character during USING execution.

```
Entry:
     USGch-: RSTK address contains table of ASCII characters
             P=pointer into ASCII table
     USGch+: P=0
             C(A)=address of ASCII character
        D1=address of current IMAGE token being executed.

Exit:
     See USst03

Calls:      USst03

Uses.......
 Exclusive: A(W),B(A),C(A),P
 Inclusive: A,B,C,D,R0(A),R1,R2,P,D1

Stk lvls:   5

NOTE:
     For USGch- entry, the ASCII table must have a 00 byte
     as the first entry.  A value of P=0 would point to the
     first byte past this 00 byte.

Detail:
    =USGch- C=RSTK                  Address of ASCII table.
            C+P+1                   Pointer into table.
            C+P+1
            P=      0
    =USGch+ B=0     A               B(A)=1=#characters
            B=B+1   A                  to display.
            A=R0                    Preserve R0(9-5).
            GOTO    USst03

History:

   Date     Programmer             Modification
 --------   ----------   ----------------------------------
 12/08/82   MB           Documentation
```

8.22 USGrst - Suspend USING execution, restart parse

Category: EXCUTL    File: MB&USG::MS

Name:(S) USGrst - Suspend USING execution, restart parse

Purpose:
Halt IMAGE execution and restart parsing of IMAGE fields.

Entry:
P = 0
R3(A)=Program Counter
RAM storage at AvMemEnd is as shown in IMGxqt header.

Exit:
To Nxtf13 (parse next field).

Calls: GETSTA, C+A2D1, R2=D1+, CA2D1., IMDO--, Nxtf13

Uses.......
Exclusive: A(A),C,D1
Inclusive: IMAGE parse routines at Nxtf13 can use anything

Stk lvls: 2 (before exit to Nxtf13, which can use all 7)

NOTE:
Most pIMXQT poll handlers will return to USGrst, after they have taken care of their execution.

Algorithm:
Restore status bits from RAM.
Restore address of start of IMAGE string to R3(9-5)
Restore length of IMAGE string to R0(9-5)
Restore address of next parse symbol to R0(A).

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

8.23 USGnum - Evaluate and execute numeric IMAGE field

Category: EXCUTL File: MB&USG::MS

Name: USGnum - Evaluate and execute numeric IMAGE field
Name:(S) USnm05 - Execute numeric IMAGE field

Purpose:
To evaluate (through EXPEXC) and execute numeric IMAGE field.

Entry:
RAM locations as specified in IMGxqt header.
USGnum:
P=0
D1=address of current token in BldIMG stream
A(B)=delimiter token which defined numeric field
USnm05:
P=0
A(W)=numeric expression (real or imaginary part)
D1 points to AvMemEnd-16, which also contains a copy of the expression in A.

Exit:
Exits to IMxq12.

Calls: SET-ST, FPOLL (pIMcpw), GetEXP, C+A2D1, DECP=C, RND-12, ExpEXP, CHKFLT

Uses.......
Inclusive: GetEXP calls EXPEXC, which may use anything

Stk lvls: GetEXP calls EXPEXC, which may use all 7

NOTE:
USGnum is the routine which formats all numeric fields. The value of the delimiting token determines the status bit settings, which in turn define the type of formatting (sign field, exponent field, etc.).

USnm05 is a return point for the pIMcpw poll ("complex field working").

Algorithm:
Set status bits as specified by numeric delimiter.
Fetch expression, store at AvMemEnd-16.
Copy expression to B.
Read #digits in field, store in D.
Read #digits before radix, store in C.
Allow 1 digit position for sign, if sign not specified.
Expand exponent to 5 digit form.
Calculate #zeroes before first nonzero digit.
Calculate position to round; round expression.
If exponent changed in rounding, decrement #zeroes.

If insufficient digits, "IMAGE Ovfl" warning/error.
Store #zeroes in R1.
Store rounded expression back in AvMemEnd-16.
If floating field (D's), go to CHKFLT
  else go to IMxq12.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 8.24 ENDIMG - Process end of IMAGE string

Category: EXCUTL    File: MB&USG::MS

Name:(S) ENDIMG - Process end of IMAGE string

Purpose:
Process uIMend token at end of IMAGE string.

Entry:
P = 0
RAM storage as shown in IMxqt header.

Exit:
If "not ouput field found" (S10=0), generates
  an "Invalid USING" error.
Else:
  P=0
  D1=AvMemEnd+5
  C(A)=address of start of IMAGE string. If there
    are more output fields, the IMAGE string can now
    be recycled.
  S0=0,S1=0,S2=0,S2=0,S6=0

Calls: GETSTA, CLOST+, RCVOFS

Uses.......
 Exclusive: D1
 Inclusive: D1,A(A),C(A),D(A),S0,S1,S2,S3,S6

Stk lvls: 2

NOTE:
During IMAGE execution (output or enter), when the end-of-image is encountered the routine TstEnd should be called to determine if at the end of the output list (or enter list). If so, exit to NXTSTM. If not, call IMGEND to recycle the image string.

Detail:
```
    =ENDING GOSUB  GETSTA          Get status bits from RAM
            GOSUB  =CLOST+         Set S0,S1,S2,S3,S6=0
            ?S1=0  10              Output field found?
            GOYES  <Invalid USING error>   No. Error.
            D1=D1+ 8               Gives D1+3 in RCVOFS
        ... fall into RCVOFS...    Recover offset to start
                                     of image string.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.25 GetEXP - Expression execute for IMAGE output list

Category: EXCUTL File: MB&USG::MS

Name:(S) GetEXP - Expression execute for IMAGE output list

Purpose:
Call EXPEXC for items in IMAGE output list, screen expression for valid type.

Entry:
```
P       = 0
R3(A)=Program Counter
RAM storage as shown in IMGxqt header.
S3 and S6 determine valid expression types:
   S6=1 means "numeric expression acceptable"
   S3=0 means "string expression acceptable"
```

```
        S3#S6 means "complex acceptable"
          valid type     S3  S6
          ----------     --  --
          numeric         1   1
          string          0   0
          complex         1   0
          any (K or M)    0   1
```

Exit:
    If expression is not of valid type, "Invalid USING".
    Else:
        P=0
        Carry clear
        S6=0
        If numeric or complex expression:
          RES register has been updated
          A(W)=numeric expression (or =real part, in
            the case of complex)
        If string expression, A(W)=string header except
          that A(B)=00.

Calls:    TstEnd, NXTEXP, CKINFO, POPMTH, AVE=D1, GETST1,
          POPTST, PUTRES

Uses.......
    Calls EXPEXC, which may use anything.

Stk lvls:  Calls EXPEXC, which may use anything.
           5 levels available to EXPEXC.

Algorithm:
    Test output list for end-of-list.  If so, to NXTSTM.
    Call NXTEXP, which stores status bits and offset
      to D1 in RAM, jumps to EXPEXC.
    Pop math stack.
    Restore status bits from RAM.
    If numeric expression:
2)      If S6=1, then go to 4). Else go to 3).
    If string expression:
        If S3=0, then return. Else go to 3).
    If complex expression:
        If S3=0, then go to 2).
        Else (S3=1) if S6=0 then go to 4).
3)  Exit to "Invalid USING" error.
4)  Put expression in RES register. Return.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.26 TstEnd - Test IMAGE output list for end of list

Category: EXCUTL File: MB&USG::MS

```
Name:(S) TstEnd  -  Test IMAGE output list for end of list

Purpose:
     Test IMAGE output list for end-of-list.  If not,
     positions D0 to next expression.

Entry:
     P      = 0
     R3(A)=Program Counter
     RAM storage as shown in IMGxqt header

Exit:
     P      = 0
     Carry clear: end of output list (D0 points past EOL,
          "@" or "!")
        D1 points to first image token
        A(B)=first image token
        C(B)=ASCII "#" for test of first image token.
     Carry set: D0 points to next expression in output list

Calls:       EOLXCK
             If end-of-list, also calls: SetAVE, C+A2D1

Uses.......
 Exclusive: A(B),C(W),D0,D1
 Inclusive: A(B),C(W),D0,D1

Stk lvls:   1

NOTE:
     If end-of-list, A(B) and C(B) are ready to test
     first image token for "#".  If the first token
     is a "#", then a CR-LF should not be sent out.

Algorithm:
     Fetch Program Counter from R3(A), copy to D0.
  1) Read byte from DAT0.
```

If A(B) not "," or ";" then return carry set
  (A(B) must be first byte in expression)
Increment D0+2
Test A(B) for EOL, "@" or "!". If no match,
  go to 1) (must be another "," or ";")
(Match with EOL, "@" or "!"):
Recover offset to start of IMAGE string,
  put address in D1.
Read first image token into A(B).
Load ASCII "#" into C(B).
Return carry clear.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.27 USloop - Loop on IMAGE multiplier

Category: EXCUTL File: MB&USG::MS

Name:(S) USloop - Loop on IMAGE multiplier

Purpose:
To process a loop-on-multiplier token while executing an IMAGE statement. Repositions D1 back to start of multiplier loop.

Entry:
  For a fixed jump (jump back a fixed number of nibbles),
    P=#nibbes-1 to jump
      P=3 for uLOOPB (loop on byte -- 4 nibble jump)
      P=15 for uLOOPS (loop on string -- 16 nib jump)
  For a jump whose length is calculated by a 5-nibble field,
      P=0 for uLOOPP (loop on parentheses)
  D1=address of loop token in BldIMG stream

Exit:
  Carry clear

```
      (P unchanged)
      If multiplier has not expired:
        loop counter has been decremented.
        D1 points to start of multiplier loop.
      If multiplier has expired:
        the reference counter has been copied into the
          loop counter.
        D1 is left as it was passed (points to loop token).
```

**Calls:** CK"ON"

**Uses.......**
Exclusive: C(A),D(A),D1
Inclusive: C(A),D(A),D1,....

**Stk lvls:** 1

**NOTE:**

USloop checks if the ATTN key has been hit; if so, it exits through PART3 (output handler), which goes to NXTSTM. Thus, an image string like "9999X" will allow the user to abort it with the ATTN key.

**Algorithm:**

```
      Copy D1 to D(A).
      Check ATTN key; if pressed, exit.
      Increment D1 by P+1.
      If P#0 (loop on byte or string), go to 2)
      Else (loop on parentheses):
        Move D1 to offset storage
        Recover offset to start of loop
   2) Decrement loop counter
      If counter not expired, return.
      Else (counter expired):
        Copy reference counter to loop counter.
        Restore D1 from D(A), return.
```

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.28 DCRMNT - Decrement multiplier in IMAGE string

Category: EXCUTL    File: MB&USG::MS

Name:(S) DCRMNT - Decrement multiplier in IMAGE string

Purpose:
To decrement loop counter in IMAGE string. An image symbol with a multiplier causes a loop which must decrement the counter each time.

Entry:
P = 0
D1 points to uMULT token (multiplier)

Exit:
P = 0
Carry clear
D1 ponts to next executing token (D1-8 from entry)
Loop counter has been decremented.
If an open parentheses loop, see note below.

Calls: none

Uses.......
Exclusive: A(B),C(A),D1

Stk lvls: 0

NOTE:
If the loop counter is for a parentheses loop which has not been closed yet (execution of the fields was started before the parse routines found the closing parentheses), then a uOPNUM token (open parentheses loop with multiplier) is found in the reference counter field. If such is the case, the uOPNUM token is replaced with a uOPNM- token to indicate that the loop counter has been decremented.

Algorithm:
Move D1-4 to reference counter.
If uOPNUM token in reference counter field, re-write with uOPNM-.
Move D1-4 to loop counter.
Decrement loop counter (DEC mode), replace; return.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 8.29 NXTEXP - Store pointers, execute next expression

Category: EXCUTL File: MB&USG::MS

Name:(S) NXTEXP - Store pointers, execute next expression

Purpose:
Store pointer and status bits, call EXPEXC for IMAGE output items.

Entry:
P = 0
D0=Program Counter (points to expression to be executed)
D1=address of current BldIMG token
RAM storage as shown in IMGxqt header

Exit:
Through EXPEXC:
D0=new Program Counter
D1=points to item on math stack

Calls: SetAVM, D1C-A, EXPEXC

Uses: EXPEXC can use anything

Stk lvls: EXPEXC can use all levels (5 availble at call)

Algorithm:
Save status bits in RAM at AvMemEnd+5.
Save offset to D1 (current IMAGE token address) in RAM at AvMemEnd.
Jump to EXPEXC.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.30 COUNTC - Count output characters in IMAGE field

Category: EXCUTL    File: MB&USG::MS

Name:(S) COUNTC - Count output characters in IMAGE field

Purpose:
To count the number of output symbols in an IMAGE field. Operates on individual symbols, checking to see if accompanied by a multiplier. If not, increments count by 1; if so, adds multiplier value to count.

Entry:
P = 0
D1 points to symbol which needs to be counted.
B(A)=current count of symbols.

Exit:
DEC mode!
Carry clear
If no multiplier accompanied symbol:
  P=0
  D1=same as entry (address+2 of next token to execute)
  B(S) incremented by 1
If multiplier accompanied symbol:
  P=14
  D1 points to uLOOPB token (address+2 of next token to execute)
  B(A) incremented by multiplier value

Calls: TstEn5

Uses.......
Exclusive: A(B),B(A),C(A),D(A),P,D1
Inclusive: same

Stk lvls: 1

NOTE:
An application which processes the uMULT token by decrementing the loop counter will want to call the COUNTC subroutine as follows. The HPIL ROM, for ENTER USING, is an example of an application which needs to call COUNTC this way.

```
GOSBVL =COUNTC       Count symbol.
SETHEX
?P=    0             Multiplier?
GOYES  ..<exit>..    No.
P=     0             Yes. Reset P.
D1=D1+ 4             Fetch reference counter,
DAT1=C 4               copy it into loop
D1=D1- 4               counter.
..<exit>..
```

Algorithm:
Move D1-2, to possible uLOOPB token.
Test token for uLOOPB; if no match, reset D1+2, goto 2)
(uLOOPB token found -- accompanying multiplier):
Move D1+6 to reference counter.
Read multiplier value into C(A).
Reset D1 to uLOOPB token.
Set P=14 to nullify LCHEX 1
2) LCHEX 1 for incrementing count
Add B=B+C A for new count, in DEC mode
Return, carry clear.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 8.31 MGOSUB - Execute A GOSUB From Movable Code

Category: EXCUTL File: MN&GSB::MS

Name:(S) MGOSUB - Execute A GOSUB From Movable Code

Purpose:
Allows code which may move (such as code within a LEXfile in RAM) to GOSUB to a utility which may move it (such as a file expand utility). The utility will return to the LEXfile properly even if it moved.

Entry:
Instead of GOSBVL <address of desired routine>, call the routine as follows:

```
GOSBVL =MGOSUB
CON(5) <address of desired subroutine>
  .
  .
  .
```

All registers and modes should be set up as required by the subroutine.

Exit:
Execution resumes at the location following the CON(5) at the call to MGOSUB.
All registers, modes and carry are as returned by the subroutine.

Calls: STRALL, PSHMCR, POPGSB, RCLALL (falls through)

Uses.......
RAM: SCRTCH, SCREX0, SCREX1, SCREX2
Register usage is dictated completely by the requested subroutine.

Stk lvls: MAX ( 3, <#levels used by requested subroutine>)

NOTE:
The scratch RAM is used before and after this code calls the requested subroutine, but not during. Thus the subroutine can use the scratch RAM locally, but not to pass information back to the calling routine. The calling routine obviously cannot keep anything there which is expected to survive =MGOSUB.

=MGOSUB acts transparently for everything, including CARRY and SB.

Because the return address is kept in RAM, the called subroutine will see the return address of MGOSUB, not of the calling code. So MGOSUB cannot be used to call a subroutine which uses the return address as a pointer to data (such as FINDA, TBLJMP, CALBIN and FPOLL). Neither POLL nor FPOLL can be called through MGOSUB.

Callers to POLL can breathe easily despite this caveat. POLL also updates the calling address, and so can be called directly from movable code. This is not the case for FPOLL.

Detail:

Calling sequence:

```
GOSBVL =MGOSUB
CON(5) <address of desired subroutine>
<execution resumes here after return>
```

Algorithm:

Stores the return address (address past the CON(5)) on Gosub stack.
Executes subroutine; address on Gosub stack will be adjusted as necessary if subroutine does a RFADJ.
Retrieves return address from Gosub stack.
Returns to code which called us.

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/31/82 | NM | Wrote |

## 8.32 STRHDR - String Header

Category: EXCUTL File: MN&UTL::MS

Name:(S) STRHDR - String Header

Purpose: Ensures there's enough memory to push string on the math stack, then writes out string header

Entry: C(A)=#NIBS IN THE STRING
D1 at top of math stack
P=0

Exit: R1[A] points to string header on stack
D1 points past the header (where string will go)

R1[15-5] = A[15-5] on entry.
A[15-5] = C[15-5] on entry.
C[A] preserved.
Carry Clear.
ERROR EXITS IF NOT ENOUGH MEMORY

Calls: none

Stack lvls: 0

Uses: A, C, D1, R1

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 07/04/82 | S.W. | Added documentation. Modified code to use AVMEME, instead of TFORM, as place to push string. |
| 10/22/82 | NM | Rewrote |

## 8.33 SENDEL - Send EndLine to Device via Handler

Category: EXCUTL File: SB&IO::MS

Name:(S) SENDEL - Send EndLine to Device via Handler

Purpose:
Transmit an "EndLine" to a device by calling the the appropriate handler routine.
Updates column count by the number of characters in buffer.

Entry:
Statement scratch set up by CKINFO

Exit:
P = 0

Calls: Device handler specified in statement scratch

Uses.......
 Exclusive: A(W), C(A), D(A), D1
 Inclusive: A(W), B(W), C(W), D(W), D1, P, R1(W), R2(A)
 Does not use D0, Status.

Stk lvls: 3

Note: DO NOT USE D0 OR STATUS BITS!!!!

Detail:
This routine calls the Part 2 handler by entering the SENDIT code.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/25/82 | B.S. | Updated documentation |

## 8.34 SENDIT - Send Buffer to Device via Handler

Category: EXCUTL File: SB&IO::MS

Name:(S) SENDIT - Send Buffer to Device via Handler
Name:(S) SEND20 - Send Buffer to Device via Handler

Purpose:
Transmit a buffer of 8-bit ASCII characters to a device by calling the appropriate handler routine. Updates column count by the number of characters in buffer.

Entry:
Statement scratch set up by CKINFO
SENDIT:
 D1 points to first byte of buffer
 Buffer end is at (AVMEME)
SEND20:
 D(A) point to first byt of buffer
 A(A) is length of buffer (in bytes)

Exit:
    P = 0

Calls:      D1@POS,
            Device handler specified in statement scratch

Uses.......
 Exclusive: A(W), C(A), D(A), D1
 Inclusive: A(W), B(W), C(W), D(W), D1, P, R1(W), R2(A)
            Does not use D0, Status.

Stk lvls:   <4

Note: DO NOT CHANGE D0 OR STATUS BITS!!!!

Detail:
    For the IO handler, the following are the entry
    conditions:
      D(A)=Starting address of buffer,
      A(A)=Length of buffer(in bytes).

    The handler may use any CPU registers
      except D0, R0 and the status bits.

    The handler has 3 stack levels (RSTK) available.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/25/82 | B.S. | Updated documentation |

## 8.35 DPART2 - IO Handler For Built-In Display

Category: EXCUTL    File: SB&IO::MS

Name:(S) DPART2 - IO Handler For Built-In Display

Purpose:
    Sends output to display devices at execution time

```
Entry:
     P      = 0
     D(A)=Start address of buffer
     A(A)=Length of bufer (in bytes)

Exit:
     P      = 0
     D1 points past last char sent (to next output char)

Calls:     CSRWP9,DSPCHA,CK"ON"

Uses.......
 Exclusive: R1(W), R2(A), A(W),       C(W),       D1
 Inclusive: R1(W), R2(A), A(W), B(W), C(W), D(W), D1

Stk lvls:   3

Detail:
               15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
  R1 usage:                      | entry DO    | buffer D1    |

  R2 usage:    |                 |             | counter #chr|

History:

   Date     Programmer              Modification
 --------   ----------   -----------------------------------
 10/19/82   B.S.         Updated documentation
 01/27/83   M.B.         Documented exit conditons
```

## 8.36 DPART3 - Finish up DISP line

Category: EXCUTL     File: SB&IO::MS

```
Name:(S) DPART3  -  Finish up DISP line

Purpose:
     Puts finishing touches on a DISP statement line,
     specifically, causing the display to be built and
     the line to be scrolled if necessary.
```

Entry:
    P = 0
    InhEOL(ST4) set if CR/LF has not just been sent to display

Exit:
    P = 0

Calls: DOSCRL

Uses.......
 Inclusive: A,B,C,D,D0,D1

Stk lvls: 5

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/01/83 | B.S. | Added documentation |

## 8.37 PUTRES - Put Numeric Result Into RES

Category: EXCUTL    File: SB&IO::MS

Name:(S) PUTRES - Put Numeric Result Into RES

Purpose: Put numeric expression in RES register.

Entry: D1 points to start of numeric expression on stack (or any desired location).

Exit: Carry clear: real. Carry set: complex.
D1= same value as entry.
P=0. Sets HEX mode.

Calls: POP1N

Uses: P, A(W), B(O), D0
RO if complex.

Stk lvls: 1

Algorithm:
Call POP1N (express purpose of checking numeric arg)
Set D0= RESREG
If complex, read 34 nibbles from the Math stack to put in the RES register.
If real, simply write A(W) into the RES register.
Returns D1 to original value.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 08/26/82 | M.B. | Wrote routine |

## 8.38 CKINFO - Check Handler Information

Category: EXCUTL File: SB&IO::MS

Name:(S) CKINFO - Check Handler Information
Name:(S) CKINF- - Specify DISP Stmt & Set Handler Info

Purpose:
Guarantees that info in STMTR0,STMTR1 is correct for the statement that is being executed.

Entry:
P=0,HEXMODE

Exit:
P=0,Carry clear

Calls: POLL

Uses.......
Exclusive: A, C
Inclusive: A,B,C,D,FUNCD0,FUNCD1,FUNCR0,FUNCR1,STMTR0

Stk lvls: <4

NOTE:

Function RAM is NOT preserved through CKINFO!!!

If MLFFLG is not clear, MLFFLG,STMTR0 and STMTR1 are updated

Detail:

RAM utilization:

```
----------------
           Pos/Width  EOLLEN (Number of nibs)
                |      |
     Handler    |      | Char#1
           |    |      | |
 Statement|     |      | |  Char#2
      Type|     |      | |   |
        | |     |      | |   |  Char#3
 MLFFLG | |     |      | |   |   |
     |  | |     |      | |   |   |  Reserved for polled
     v  v v     v      v v   v   v  v          handlers
    ----------------------------------------------------
    |1|1| 5  |  5  |1|2 |2 |2 |        14           |
    ----------------------------------------------------
       ^                         ^
       |                         |
     STMTR0                    STMTR1
```

If MLFFLG is clear then routine returns quickly otherwise a handler address and other information is set up to transfer information to the device which is appropriate for the statement. The states are coded as follows:

```
MLFFLG          0 --> Information okay
                F --> Information not reliable

Statement type  0 --> DISP
                1 --> PRINT
                2-F --> POLL for setup
                          2 --> OUTPUT
                          3 --> PLOT
                          4-F --> Reserved
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/82 | N.Z. | Updated documentation |

## 8.39 EXCPAR - Execution Time Expression Parse

Category: EXCUTL    File: SB&IO::MS

```
Name:(S) EXCPAR  -  Execution Time Expression Parse

Purpose:
     Parses an expression in the constraints of an
     executing statement.

Entry:
     Carry clear:  D1 contains pointer to input stream
     Carry set:  A(A) contains pointer to input stream
     The pointer to the input stream is also used as
       a starting point for the parse stack.
     (AVMEMS) is start of output buffer
     P      =  0

Exit:
     P      =  0
     (AVMEME) = D1 on entry
     See exit conditions for EXPPAR

Calls:      AVE=D1,EXPP10

Uses.......
 Exclusive: C,D0,D1,R3,(AVMEME)
 Inclusive: A,B,C,D,D0,D1,R0,R1,R3

Stk lvls:   3

History:

   Date     Programmer              Modification
 --------   ----------   -----------------------------------
 11/01/83   B.S.         Added documentation
```

## 8.40 REPROM - Reprompt for input

Category: EXCUTIL File: SB&IO::MS

Name:(S) REPROM - Reprompt for input

Purpose:
Sends buffer to display following prompt and positions cursor to start of line.

Entry:
C(A) = Pointer to buffer to be displayed
R3(A) = Pointer to quoted string that is prompt

Exit:
Exits via DONNA

Calls: DONNA

Uses.......
Inclusive: A,B,C,D,D0,D1,R3

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/01/83 | B.S. | Added documentation |

## 8.41 INPOFF - Restart statement after DSLEEP

Category: EXCUTIL File: SB&IO::MS

Name:(S) INPOFF - Restart statement after DSLEEP

Purpose:
    Allows a statement to set itself to be restarted if continue is pressed, then turns off machine. ATTN key will send machine back to BASIC interpreter which will suspend execution.

Entry:
    P = 0

Exit:
    Exits through MFERRS

Calls: FINLIN,DSLEEP,MFER42,MFERRS

Stk lvls: 6

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/01/83 | B.S. | Added documentation |

## 8.42 VALOO - Parse and Execute a String on Stack

Category: EXCUTL    File: SB&VAL::MS

Name:(S) VALOO - Parse and Execute a String on Stack

Purpose:
    System VAL function. Converts a string into a number. Any valid numeric expression may be passed.

Entry:
    P = 0
    D1 points to string on top of math stack.
    ST10 (=ValSub) set iff VAL is being called as a subroutine.
        Will cause "Data Type" error instead of "Invalid Argument" and will require the valid expression to be followed by a CR.

**Exit:**

P = 0

String on top of stack has been replaced by the value obtained by parsing and executing the string.

**Calls:** XXHEAD,STKCHR,ADHEAD,REVPOP,EXCPAR,OUTBYT, MOVED2,PSHSTK,EXPR,POPSTK,POP1N,AVE=D1,MFERR

**Uses.......**
**Inclusive:** A,B,C,D,R0,R1,R2,R3,R4,D1

**Stk lvls:** 4

**NOTE:**

This routine calls expression execute which may call a user defined function; this may alter a lot of RAM locations. The D0 that is passed in is kept on the GOSUB stack so it will be updated if memory moves.

**Algorithm:**

Appends a CR to string on stack.
Reverses string.
Parses string and verifies it is a valid numeric expr.
Appends an EOL to parsed code.
Moves parsed code onto stack, covering original string.
Saves 2 RSTK levels and D0 (PC) on GOSUB stack.
Calls EXPR to evaluate expression.
Pops value from stack.
Collapses parsed code from stack.
Checks validity of pointers saved on GOSUB stack
  and jumps to MFERR(eMMCOR) if any are not valid.
Restores 2 RSTK levels and D0 (PC) from GOSUB stack.
Pushes value on stack.
Returns

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 02/04/83 | B.S. | Added documentation |
| 04/08/83 | B.S. | Modified routine to observe S10. |

## 8.43 CHKEOL - Check if at End of Statement

Category: EXCUTL File: SC&DAT::MS

Name:(S) CHKEOL - Check if at End of Statement

Purpose: When processing the PRINT or READ list, check to see if just past the last variable on the list.

Entry: DO = Program counter

Exit: Carry set => Not at end of statement yet.
Carry clear => PC is at end of the statement

Uses: A(B), C(B)

Stk lvls : 0

## 8.44 NXTVAR - Get next Variable from READ list

Category: EXCUTL File: SC&DAT::MS

Name: NXTVAR - Get next Variable from READ list
Name:(S) NXTVA- - Get next Variable from READ list

Purpose: Get the next variable from the READ list, the variable will be created if it does not yet exist.

Entry: DO @ the next variable token

Exit: The updated DO (past the variable) saved in STMTDO
MTHSTK is set to current top of stack.
The variable value or its dope vector is on top of math stack.
DEST has been called ( DEST will save all the information in STMTR0 & STMTR1 that need to assign a

value from math stack to the variable).

Calls: EXPEX-

Uses: All CPU registers, scratch RAM and status.

Stk lvls: 5

## 8.45 STKVCT - Process Array Dope Vector

Category: EXCUTL    File: SC&DAT::MS

Name:(S) STKVCT - Process Array Dope Vector

Purpose: Process an array dope vector on math stack. When printing or reading an array to or from a data file, it is done one element at a time. The array dope vector will remain on the stack until done, so it can be used to keep track of the next element addr and number of elements left to be done.
The dope vector on the math stack will contain :

| Nibs | Meaning |
|---|---|
| 0 | Variable type. A-Int, B-Short, C-Real.... |
| 1 | Dimensions. (1 or 2) |
| 2 | Option base. |
| 3-6 | Maximum string length if is string variable |
| 7-10 | Number of elements left to be done. |
| 11-15 | Next element address. |

Entry: D1 @ stack pointer
If S8 =1, rewite dope vector

EXIT:
Following status bit will be set properly :
Notnum(S0) - Not simple real
Array (S1) - Numeric or String array
String(S2) - String or string vector
Cmplex(S3) - Complex number or Complex array
If is an array element(S1=1):
Carry clear => All elements done

```
            STACK(10-7)= Number of elements left
            STACK(6-3) = String max. length in bytes
            STACK(15-11)= Next element address
Used:   A,C,D0,D1
```

## 8.46 NXTADR - Get Address of Next Array Element

Category: EXCUTL    File: SC&DAT::MS

```
Name:(S) NXTADR  -  Get Address of Next Array Element

Purpose:  Get the address of next element of an array

Entry:
        MTHSTK pts at the array dope vector(top of stack)
        S8 = 1 If to get the address of the first element
           When the dope vector is first time recalled to
           the math stack, the address field already
           point to the next element address. Set S8 will
           it been moved to next element address.

Exit:  Carry clear:
         D1 @ Top of stack
         S-R0-3 = Data type:  0- real, 1-short, 2-integer
                  E- complex, F- short complex, D-STRING
         S-R0-0 = next element address
         If is a string vector:
          R3 = Max. string length
          S-R1-1 = Max. string length

Used A,C,D0,D1, STMTR0, STMTR1, R3 (if string vector)

Stk lvls: 1
```

## 8.47 NXTELM - Get Next Array Element

Category: EXCUTL File: SC&DAT::MS

**Name:(S) NXTELM - Get Next Array Element**

**Purpose:** Get next array element
While printing or reading an array, the array vector on the stack is used to keep track of next element address and # of elements left. This routine will get the next element and update the vector information.

**Entry:**
The dope vector on the math stack will contain :

| Nibs | Meaning |
|---|---|
| 0 | Variable type. A-Int, B-Short, C-Real, D-S.Complex E- Complex, F- String |
| 1 | Dimensions. (1 or 2) |
| 2 | Option base. |
| 3-6 | Maximum string length if this is string variable |
| 7-10 | Number of elements left to be done. |
| 11-15 | Next element address. |

**Exit:** Carry set => All done, there is no next element
Carry clear => Not done yet, there are more elements.
S5 = 1 if no room on math stack to recall the value of next element.
If numeric array:
B = Next element
The element count and next element address will be updated in the array dope vector on math stack.
If still room on stack, the element will be written to the stack on top of the array dope vector and the MTHSTK will be updated
If is a complex array:
D = Imaginary part
B = Real part
The two number will be written to stack too
If string: D0 @ string start
A= Address past the string element
C= String length in nibs + 4

Note:
The data type, such as real, string or complex, should still be indicated by S2 and S3 :
S2 = 1 - String
S3 = 1 - Complex

Used: A,B,C

Stk lvls: 1

## 8.48 STRHED - Generate String Head on Stack

Category: EXCUTL File: SC&DAT::MS

Name:(S) STRHED - Generate String Head on Stack

Purpose: Generates string header on stack

Entry:
The string data is sitting on top of MTHSTK
D1 @ top of the string
(MTHSTK) @ end of the string (beyond last character)

Exit: String header will be written on top of the string.
D1 @ string header.
(MTHSTK) @ string header.
If not enough memory to generate the header(16 nibs), it will direct exit to MFERR error routine.

Calls: STK16?

Uses: A,B,C(A),D0,D1

Stk lvls: +1

## 8.49 GETCHN - Get Channel Number

Category: EXCUTL File: SC&FIL::MS

Name:(S) GETCHN - Get Channel Number

Purpose: Get the Given channel for a statement

Entry: D0 points at the channel number token.

Exit: A(B) = Channel number in binary
D0 past channel number
CHNNSV = Channel #
Error exit if channel # > 255 or <= 0

Uses: All CPU registers, status. scratch RAM except
All scratch RAM except STMTR0, STMTR1
(Expression execution is called)

Calls: EXPR

Stk lvls: +5

## 8.50 D1MSTK - Set D1 at MTHSTK (AVMEME)

Category: EXCUTL File: SC&SUB::MS

Name:(S) D1MSTK - Set D1 at MTHSTK (AVMEME)

Purpose: Set D1 to point to available memory end (top of
math stack)

Entry: None.

Exit:
D1 @ Top of math stack (available memory end)
C(A) = Address of AVMEME

Calls: None.

Uses: C(A)

Stk lvls: 0

## 8.51 D1FSTK - Set D1 to FORSTK

Category: EXCUTL File: SC&SUB::MS

Name:(S) D1FSTK - Set D1 to FORSTK

Purpose: Set D1 to top of FOR/NEXT stack.

Entry: None

Exit: D1 points at FOR/NEXT STACK

Uses: C(A)

Stk lvls: 0

## 8.52 TRFROM - Trace Line Number

Category: EXCUTL File: SC&TRC::MS

Name:(S) TRFROM - Trace Line Number

Purpose: Routine to generate the "Trace nnnn to" in display.
The current line number is computed from PCADDR.

Entry: PCADDR @ current line length
P=0

Exit: Send "Trace nnnn to" to display buffer
(Via AVS2DS)

Calls: TRCLIN

Uses: A,B,C,D,D0,D1,D0, R0, P

Stk lvls: +4

Note: Will exit to error routine if not enough memory to buffer the display line.

## 8.53 TRTO - Generate Trace Message

Category: EXCUTL File: SC&TRC::MS

Name: TRTO - Generate Trace Message
Name:(S) TRTO+ - Generate Trace Message
Name: TRTO- - Generate Trace Message
Name: TRTO* - Generate Trace Message

Purpose: Generates "to nnnn" for TRACE FLOW mode.
The line number is computed from D0 on entry.

Entry :
D0 is pointing at some where in the current line.
(A line can have multiple statements)

TRTO+: D0 pts at EOL/@ preceding a statement
P=0
TRTOr: D0 pts at the line length of a statement.
TRTO-: D0 pts at middle of a statement
TRTO*: D0 pts at EOL preceding the current line.

Exit: Via CRLFSD
TRTO+: R1 = D0 on entry.

Calls: CPLN10, D0=PCA, DSBFCK, DSINTR, TRFM20.

Uses:
A,B,C,D,D0,D1,S9

TRTO+ also uses R1 to save the D0 on entry.

Stk lvls: +5

History:

| Date | Programmer | Modification |
|---|---|---|
| 3/11/83 | SC | Document |

## 8.54 LINSKP - Line Skip

Category: EXCUTL    File: SG&EXC::MS

Name: LINSKP - Line Skip
Name:(S) LNSKP- - Line Skip

Purpose: Skips to next statement

Entry: 2 entry points:
1) LNSKP- - PCADDR points to stmt length byte
2) LINSKP - D0 points to stmt length byte

Exit: D0 points to end of statement token (t@ or tEOL)
A(A) = D0
B(B) = Statement length
Carry Clear

Calls: D0=PCA (LNSKP- entry only)

Stack lvls: 1 (LNSKP- only)
0 (LNSKP entry)

Uses: A(A), B(A), D0

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 10/15/82 | S.W. | Call to D0=PCA to save code |

## 8.55 NXTSTM - Scan to Next Stmt/Jump to BASIC Loop

Category: EXCUTL File: SG&EXC::MS

```
Name:(S) NXTSTM  -  Scan to Next Stmt/Jump to BASIC Loop

Purpose:  Next statement scan & jump to BASIC loop @ RUNRTN

Entry:    ENTRY POINTS:
            NXTSTM - entry point to go on to the following
                     statement.  No assumptions made.
                     PCADDR must be current.
                     sENDx flag will be explicitly cleared.
                     entry point for IMAGE & REM.
            NXTST1 - Entry point for END execute. (sENDx=1)
                     PCADDR must be current.
            NXTST2 - D0 points at statement length byte.
                     Assumes sENDx is clear
            NXTST3 - D0 points at EOL token
                     Assumes sENDx is clear
            NXTST5 - D0 already points at EOL token
                     Explicitly clears sENDx
                     Entry pt for routines which may
                     have inadvertantly set sENDx, perhaps
                     via EXPEXC

            LABEL  - Label 'execute' (NOP)
            DATA   - DATA statement execute (NOP)
            BANG   - REM (!) execute (NOP)

Exit:     D0 POINTS TO @ OR EOL TOKEN
          Through RUNRTN

          LABEL:
            Skips ASCII Label
            If Multi-statement line   ("@")
               Through RUNXLP (to avoid SST between Labels)
            else
               Through RUNRTN (with D0 @ EOL)

Calls:      none
```

Stack lvls: 0

Uses: A(A), B(A), C(A), D0, S1 (sENDx)

Detail: USED TO 'EXECUTE' REM, LABEL, DATA STATEMENTS

The END Execute flag is ALWAYS cleared by NXTSTM
END enters at NXTST1 with sENDx set
This is necessary when a program is NOT to continue

Label Execute:
@EOL return to BASIC loop

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 03/30/83 | J.P. | Shift C(B) for ASCII check |

## 8.56 TKSCN+ - Token Scan

Category: EXCUTL File: SG&EXC::MS

Name:(S) TKSCN+ - Token Scan
Name: TKSCN4 - Token Scan
Name:(S) TKSCN7 - Token Scan

Purpose: Search program memory (or statement buffer) for a specific 2 nibble begin BASIC token

Entry: C(B) contains token to match on
P=0
D(A)= PRGMEN if in a program
= end of statement buffer, otherwise
3 Entry points:
1) TKSCN+ - D0 at tEOL before search start
2) TKSCN4 - D0 at some statement length byte
3) TKSCN7 - D0 at tEOL or t@ before search start.

Exit: CARRY SET => Token found & D0 points to it.

```
          CARRY CLR => Searched to program end
                       (or statement buffer end) without
                       finding a match.
```

Calls: none

Stack lvls: 0

Uses: A(A),B(A),C(A),D0

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 07/07/82 | S.W. | All references to F-R0-0 & S9 to save CURRL have been eliminated. |

## 8.57 EOLSCN - tEOL Scan

Category: EXCUTL    File: SG&EXC::MS

```
Name:(S) EOLSCN  -  tEOL Scan
Name:    EOLSN5  -  tEOL Scan
Name:    EOLSN7  -  tEOL Scan

Purpose:   Scans to tEOL (as opposed to t@ OR tEOL)

           3 entry points:
             1) EOLSCN -  PCADDR at current stmt len byte
             2) EOLSN5 -  D0 at t@ or tEOL
                          C(B)=tEOL
             2) EOLSN7 -  D0 at t@
                          C(B)=tEOL

Exit:     D0 POINTS TO EOL; A(B) = EOL TOKEN;  CARRY SET
          If EOLSCN entry point used, P=0.

Calls:    LINSKP
```

```
Uses:     A(A), B(A), C(B), D0

Stack lvls: 2

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/01/82    S.W.         Added documentation
 01/17/83    S.W.         Added EOLSN5 entry point
```

## 8.58 KEYFND - Key Assignment Find

Category: EXCUTL     File: SG&EXC::MS

```
Name:(S) KEYFND  -  Key Assignment Find
Name:    KYFND+  -  Key Assignment Find

Purpose:  FINDS SPECIFIED KEY ASSIGNMENT IN keys FILE

Entry:    P=  0
     2 entry points:
            1) KEYFND - B(A)=keycode
            2) KYFND+ - D(A)=keycode
                        A(A) points to header of keys file

Exit:     CARRY CLR=> NO MATCH
                      D1 points past last entry which
                      had a smaller keycode value
                SET=> MATCH FOUND. D1 AT ENTRY.
                      C(A)=Entire entry length
                      D0 points to file header end
          P=0
          B(A)=KEYCODE
          If entry point KEYFND was used then:
            S8=1=> NO keys FILE
               =0=> D0 POINTS TO FILE HEADER END
                    R3 POINTS TO FILE START

Calls:    FILEF, LAKEYS - only KEYFND entry point
```

Uses:
 exclusive... A, B(A), C, D, D1, D0
 inclusive... A, B(A), C, D, D1, D0, S6,S8, R3 - KEYFND

Stack lvls: 1 KEYFND entry
            0 KYFND+ entry

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |

## 8.59 KEYDEL - Key Assignment Delete

Category: EXCUTL     File: SG&EXC::MS

Name:(S) KEYDEL - Key Assignment Delete

Purpose: If there's an assignment string associated with specified key, delete it.

Entry: B(A) = Keycode
P=0

Exit: P=0
Carry Clear
Any assignment to that key is deleted
  via RFAD--

Calls: KEYFND, MOVEUM, KYPRCK

Uses: A-D, D1, D0, R0,R1,R3, S6,S8

Stack lvls: 3

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |

```
12/29/82   S.W.        Eliminated call to RFAD94
```

## 8.60 GTKYCD - Get Keycode

Category: EXCUTL    File: SG&EXC::MS

```
Name:(S) GTKYCD  -  Get Keycode
Name:(S) GTKYC+  -  Get Keycode

Purpose:  Evaluates string expression & returns keycode

          The GTKYCD entry assumes that D0 points to the
          expression to be evaluated.  It errors if the
          string is null.

          GTKYC+ assumes that the evaluated expression
          is already on the stack.  A status bit setting
          on entry indicates whether or not a null string
          should cause an error exit.

Entry:    2 entry points:
          1) GTKYCD - D0 at expression.
          2) GTKYC+ - Evaluated string on stack.
                      S10=1 => Null string doesn't cause
                               error exit.

Exit:     CARRY CLR => B(A) = Keycode - between 1 & A8
                       A(A) = Shift value  (0,56,112)

          If error encountered, error exits through MFERR
          with eDATTY or eIVARG

Calls:    EXPEXC, POP1S, DECHEX, CONVUC, DRANGE, MEMBER

Uses:
  Exclusive... A-D, D1,D0, S8,S9,S10
  Inclusive... Above + R0-R3, S0-S11, all of function scratch

Stack lvls: 5

History:
```

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 12/17/82 | S.W. | When key assigned using ascii char (not key#), now erroring on alternate characters; for example those with ascii val less than 32 (blank) or greater than 125 (}). Was making assignments to keys in non-obvious way. |
| 01/26/83 | S.W. | Between ascii values 32 & 125 are 4 values which aren't represented on our keyboard - these are now trapped out. |
| 02/22/83 | B.S. | Changed GTKYC* entry point to allow returning with carry set and B(A)=0 if null string passed. |

## 8.61 STMBUF - Collapse statement buffer check

Category: EXCUTL    File: SG&EXC::MS

Name:(S) STMBUF - Collapse statement buffer check
Name:(S) STMBCL - Collapse statement buffer check

Purpose : Some statements need to collapse the statement buffer when executed from the keyboard. These statements are: CONT, RETURN, ENDSUB, ENDDEF They call the entry point STMBUF.

STMBUF - Collapses Statement Buffer only if no program is running
STMBCL - Collapses Statement Buffer, unconditionally

Entry : S13 = 0 if the statement is executed from keyboard
STMBCL: Alway collapses

Exit : Carry set

Calls : I/OCOL, STMBFD        May exit via FORUPD

Uses : A-D, D0, D1, S14 (STMBUF entry only)

Stk lvls : 2

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 01/27/83 | S.W. | Added call to RFADJ- to zero references to collapsed buffers. Additionally uses R0,R1 |
| 05/19/83 | J.P. | Set NoCont if not running so ENDSUB,ENDDEF,RETURN will SUSP |

## 8.62 SCOPCK - Scope check

Category: EXCUTL File: SG&EXC::MS

Name:(S) SCOPCK - Scope check

Purpose: Verifies if an address is in current program scope

Entry : A(A)= ADDRESS TO BE VERIFIED

Exit:
- A is preserved from entry
- Carry clear - Address in current program scope
- Carry set - Address out of current program scope

Calls: none

Uses: C(A),D0

Stk lvls: +0

## 8.63 KEYNAM - Return key name string from keycode

Category: EXCUTL File: SG&KEY::MS

Name:(S) KEYNAM - Return key name string from keycode

Purpose:
Returns string representing a keycode

Entry:
A(B)=Keycode to be named.

Exit:
A(WP)=ASCII for keycode.
P=Word thru pointer length of text
UseQuo(S0) set iff double quotes should be used
to surround string.

Calls: RANGE, HXDASC

Uses: A,B,C,R0,S0,S1,S2,D0

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/10/83 | B.S. | Updated documentation |

## 8.64 MFER42 - Position D0 to start of BASIC stmt.

Category: EXCUTL File: TI&ERD::MS

Name:(S) MFER42 - Position D0 to start of BASIC stmt.

```
Purpose:
     To position D0 to start of BASIC stmt -- to either
     an "@" character, or the line number.

Entry:
     PCADDR pointer must be updated already (points
        to the first token in the BASIC statement).
     S13=0 if program not running
        =1 if program running.

Exit:
     (P unchanged)
     Carry set: program not running (S13=0 at entry)
     Carry clear: program running (S13=1 at entry)
            D0 points to either the "@" character
              or to the line number at the start of
              the BASIC statement.

Calls:      D0=PCA, ATCHK

Uses.......
 Exclusive: D0
 Inclusive: A(A),D0

Stk lvls:   1

NOTE:
     This routine does not find the start of a BASIC state-
     ment -- call CPLN10 for that.  For MFER42, PCADDR must
     already point to the first token in the statement.
     This routine simply backs up D0 to the "@" (D0-2),
     or the line number (D0-6).

Algorithm:
     If S13=0 (program not running), return.
     Fetch PC from PCADDR, put in D0.
     Back D0 up 2 nibbles, to possible "@".
     ATCHK: If D0 points to "@", rtncc.
            Else, D0-4 to point to line number.

History:

  Date      Programmer            Modification
 --------   ----------   -----------------------------------
 12/08/82   MB           Documentation
```

## 8.65 TBMSTX - Find and Build Message From Lex Table

Category: EXCUTL     File: TI&ERD::MS

```
Name:     TBMSTX  -  Find and Build Message From Lex Table
Name:(S)  TBMSG$  -  Find and Build Message From Lex Table
Name:     MsgAvs  -  Build message from table, in AvMemSt

Purpose:
     Search LEX tables for desired message, and build it
     into a buffer at D0.

Entry:
     MsgAvs -- RAM location ERR# contains desired msg #
     TBMSTX -- D0 points to buffer to build message.
               R0(3-2)= LEX ID#, R0(B)= msg #.
               P= desired value to clear portion of R0.

Exit:
     D0 points to FF terminator at end of built msg.
     P=0, C(B)= FF.
     Carry cleared.

Calls:       LXTFND, D0ASCI, CSRWP9, CSLWP9, RANGE,

Uses:
 Exclusive: A,B,C,D,D1,D0,R0,P,
            R2 (if msg calls for text insertion)
 Inclusive: same

Stk lvls:   2

Algorithm:
  MsgAVS  Set D0=AvMemSt
          Copy ERRN (from ERR#) into C(3-0)
  TMBSG$  Set P=15 to disallow all text insertions
  TBMSTX  Save msg number in B
      (1) Clear R0(WP)
          Set D1=start of LEX I/O buffer (LXFND)
          If message is from LEX ID=00, go to (3).
      (2) Chain through buffer until:
              End of buffer: Send out null (msg #0000)
              LEX buffer match.
          Compute offset to LEX file message table.
          Check message table range; if no match,
              go to (2).
```

```
            (Range match:)
            Save address of table in D(A).
        (3) If searching for table title, set message
               number=00
            Search table for message number
            If no match, send out null (msg #0000)
            (Message match:)
        (4) Process cells:
               If cell id = C, go to (5).
               If cell id < B, then call DOASCI
                 to output #chars. Process next cell.
               If cell id = B, then read next nib,
                 call DOASCI. Process next cell.
               If cell id = D, store present table
                 address in R0, set D1=address in D(A),
                 go to (3).
               If cell id = E, set  D1=mainframe table
                 address, store present table
                 address in R0, go to (3).
               If cell id = F0, set B=new msg number
                 from table, go to (1).
               If cell id = F1, set B=new msg number
                 from R2, go to (1).
               If cell id = F2 or F3, fetch codes from
                 R2, store present table address in R0,
                 call DOASCI.  Process next cell.
        (5) If table address in R0 (from previous cell)
              set D1=that address, go to (4).
            Else, fall into D0=AVS, return.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/05/83 | MB | Documentation |

## 8.66 FLDEVX - Make Device Code Explicit

Category: EXCUTL     File: TI&UTL::MS

Name:(S) FLDEVX - Make Device Code Explicit

```
Name:     FLDEV+  -  Make Device Code Explicit

Purpose:
     Maps the FSPECx device code into the FIB device code
     without having to find the file using FINDF.  In certain
     cases maps unspecified device states to appropriate code.

   For SOURCE device:
     Does NOT map undefined device to MAIN.  Identifies port
     if explicit.  Returns carry set only for illegal port.

   For DESTINATION device:
     Maps undefined device to MAIN, explicitly identifies
     port.  Returns carry set for illegal or unspecified
     port.

Entry:
     S3(sDEST) = 0 if SOURCE file (see above), 1 if DEST file.
     P       = 0
   FLDEVX:
     D(S)    =  Device code returned from FSPECx.
     D(3-0)  =  Device code data returned from FSPECx.
   FLDEV+:  (for file info as returned by RDINFO)
     D(0)    =  Device code returned from FSPECx.
     D(4-1)  =  Device code data returned from FSPECx.

Exit:
     P       =  0
   Carry clear:
     Device code and data are sufficiently explicit.
     D(S)    =  See Detail
     D(A)    =  See Detail
   Carry set:
     Device code and data are illegal or not explicit:
       SOURCE:  Port ID is specified but illegal.
                D(A) = 0
       DEST:    Port ID unspecified or illegal.
                If PORT ID unspecified:  D(B) = FF
                   else                  D(A) = 0

     C(3-0) =  Error code: "Device not Found"

Calls:       ROMF-1, CSLW5, CSRW5

Uses.......
 Exclusive: C(S), C(A), D(A), R0(15-5),
 Inclusive: B, C, D, D1,      R0(15-5), R1, R2, R3, S2

Stk lvls:   3

Detail:
```

| ON ENTRY | ON EXIT | | | |
|---|---|---|---|---|
| D(S) | D(S) | D(4-3) | D(XS) | D(B) |
| F (Undef) | 0 (DEST) | 0 | 0 | 0 |
| | F (SOURCE) | 0 | 0 | 0 |
| 0 (MAIN) | 0 | 0 | 0 | 0 |
| 1 (PORT) | 1 (IRAM) | 0 | 0 | Port ID |
| | 2 (ROM) | 0 | 0 | Port ID |
| | 3 (EEPROM) | 0 | 0 | Port ID |
| 7 (CARD) | 7 (CARD) | entry | entry | PCRD flg |
| 8+ (HPIL+) | 8+ | entry | <device | address> |

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/19/82 | FH | Wrote. |
| 11/15/82 | FH | Completely rewrote for new device codes. |
| 03/21/83 | JP | Error Msg = eDVCNF |
| 03/21/83 | JP | Pack byte by calling ROMF- |
| 03/21/83 | JP | If PORT not found, set D(A)=0 |

```
+----------------------------------------------+-----------------+
|                                              |                 |
|  FILUTL  -  File Utilities                   |  CHAPTER  9     |
|                                              |                 |
+----------------------------------------------+-----------------+
```

## 9.1 TFHDLR - Find Transform Handler

Category: FILUTL    File: FH&TFM::MS

```
Name:(S) TFHDLR  -  Find Transform Handler

Purpose:
     Find the address of a transform handler capable of
     reading and transforming lines of the source type into
     lines of the destination file type.

Entry:
     P      =  0
     A(A)   =  Destination file type
     C(A)   =  Source file type
     S5     =  Set if transformation is IN PLACE (sTFINP)

Exit:
     P      =  0
     S5     =  Preserved (sTFINP)
    Carry clear: [Transform handler found]
     S0     =  Set if transform requires a handler (sTFREQ)
     C(A)   =  Destination file copy code
     C(S)   =  Transform handler address
    Carry set:
     Indicates that a transform handler NOT found, or that
       the source and destination file types are the same
       and no LEX file declared that a handler was needed
       (in this case, S0 will be clear; transform can be
       handled by COPY or by doing nothing if IN PLACE).

Calls:      FPOLL

Uses.......
 Inclusive: A,B,C,R0,D0,D1

Stk lvls:   5
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/01/83 | FH | Derived from in-line code |

## 9.2 LOCFIL - Locate File With FIB

Category: FILUTL File: FH&TFM::MS

```
Name:(S) LOCFIL  -  Locate File With FIB
Name:    LOCFIM  -  Locate File With FIB

Purpose:
     Find FIB for file given file number and return position
     information.

Entry:
   LOCFIL:
     A(B)    =  FIB file number (LOCFI+ will return it in R4)
   LOCFIM:
     R4(15,14)  =  FIB file number

Exit:
     P        =  0
     R4(15,14) = FIB file number (LOCFI+, LOCFIM only)
    Carry clear:  FIB entry found
     A(x-0) =  "Data Begin" field of FIB entry
      (S)   =  Protection nibble from FIB
     B(A)   =  Address of FIB entry
     C(A)   =  "Current Position" field of FIB entry
     D(S)   =  Device code
      (A)   =  D(X) = Dev addr if external device, rest 0
            =  D(B) = Port id if port, rest 0
            =  0 if MAIN
     D1     @  "Current Position" field of FIB entry
     S7     =  Set if current position is at EOF (sEOF)
     S10    =  Set if external device (sI/OBF)
     STMTD1 =  Address of File FIB
    Carry set:    Error encountered
     C(3-0) =  eFnFND if FIB entry not found
```

```
              = eNtIMP if external device

   Calls:     FFIBM

   Uses.......
    Inclusive: A,B,C,D,D1,P,S7(sEOF),S10(sI/OBF)

   Stk lvls:  2

   History:

       Date      Programmer              Modification
     --------    ----------    ---------------------------------
     06/07/82       FM         Designed and coded
```

## 9.3 PURGEF - Purge Internal or External File

```
        Category:  FILUTL      File:  FH&TFM::MS


   Name:(S) PURGEF  -  Purge Internal or External File

   Purpose:
        Purge file given its FSPECx information.

   Entry:
        P       =  0
        A(W)    =  First 8 chars of file name.
        R0(3-0)=  Last 2 chars of file name.
        D(S)    =  Device code
        D(3-0)  =  Secondary device info

   Exit:
        P       =  0
        File purged.  If file not found, error ignored.

   Calls:
        FINDF, PRGFMF, POLL

   Uses.......
    Inclusive: A-D,D0,D1,P,R0,R1,S-R0-0,S-R0-1,S7,S8
        If purging current file: also R2,R3,S9,S10,S11,S7-S0
```

```
        " " " LEX      "    also R2,R3,S9

   Stk lvls:   6

   History:

       Date    Programmer              Modification
     --------  ----------   ----------------------------------
     06/07/82     FH        Designed and coded
     06/09/83     FH        Expanded to include external files
```

## 9.4 ?PRFIL - Check File Protection

```
        Category:  FILUTL     File:  FH&TFM::MS


   Name:(S) ?PRFIL  -  Check File Protection
   Name:(S) ?PRFI+  -  Check File Protection

   Purpose:
        Checks file protection nib returned by LOCFIL for
        privacy (?PRFIL) or security (?PRFI+).

   Entry:
        P      =  0
        A(S)   =  Protection nibble

   Exit:
        P      =  0
      Carry set:
        C(3-0) =  File protection error code (eFPROT).

   Calls:      None.

   Uses.......
    Inclusive: C(S), C(3-0)

   Stk lvls:   0

   History:

      Date     Programmer              Modification
```

```
--------   ----------   ----------------------------------
08/24/82      FH        Designed and coded
```

## 9.5 RDBAS - Read Line From Basic File

Category: FILUTL     File: FH&TFM::MS

```
Name:(S) RDBAS   - Read Line From Basic File

Purpose:
     Read a line from a BASIC file given the file's FIB.
     For memory files, FIB is spaced past line but no data
     is copied to output buffer.  For external files, line
     read is copied to output buffer.

Entry:
     R4(15-14) = File FIB#
     OUTBS @  Start of output buffer
     (AVMEMS)  =  (OUTBS)

Exit:
     P      =  0
    Carry clear:  Line read
     S7     =  Set if file was positioned at EOF at operation
               start, hence no data read (sEOF)
     C(A)   =  Full len (nibs) of line in file counting line
               header.  Zero if S7(sEOF) set
     R3     =  Pointer to start of data read (in file or in
               output buffer) unless S7(sEOF) set.
    Carry set:
     C(3-0) =  Error code:

Calls:      READNB, RECNIB, TFUEOF, EOLSN7, FIBUPD, LOCFIN

Uses.......
 Inclusive: A-D,D0,D1,R0-R3,STMTR1,STMTD1,S11-S9,S7,S6,S4-S0

Stk lvls:   5

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 12/15/82 | FH | Designed and coded. |

## 9.6 RDTEXT - Read Line From Text File

Category: FILUTL  File: FH&TFM::MS

```
Name:(S) RDTEXT  -  Read Line From Text File

Purpose:
     Read a line from a text file into the output buffer
     given the file's FIB.  The line's length header or
     EOF mark are not copied into the output buffer.

Entry:
     R(15-14) =  File FIB#
     OUTBS  @  Start of output buffer
     AVMEMS @  (OUTBS)

Exit:
     P      =  0
     OUTBS  @  Start of output buffer.
     AVMEMS @  After last nib read.
    Carry clear:  Line read
     S7     =  Set if file positioned at EOF. (sEOF)
     C(A)   =  Full len (nibs) of line in file counting line
               header.  Zero if no EOF marker at end of file.
    Carry set:
     C(3-0) =  Error code:

Calls:      TFUEOF, READNB, RECNIB, SWPBYT, LIF>NB, OBPRD

Uses.......
 Inclusive: A-D,D0,D1,R0-R3,P,S11-S9,S7,S6,S4-S0

Stk lvls:   5 plus 1 RSTKBF level

History:

   Date     Programmer                Modification
```

```
--------   ----------   -----------------------------------
06/12/82   FH           Designed and coded.
09/21/82   FH           Revised to fix byte reversal in
                          line header
```

## 9.7 READNB - Read/Write Nibs To/From File

Category: FILUTL     File: FH&TFM::MS

```
Name:(S) READNB  -  Read/Write Nibs To/From File
Name:(S) WRITNB  -  Read/Write Nibs To/From File

Purpose:
     Write a line to a file given its FIB file number.  File
     may reside in memory or on external device.  File will
     be positioned to start of previous line before the line
     is written.

Entry:
     R4(15-14)  =  Number of file in FIB
     C(A)    =  #Nibs to read if reading
     R3(A)   =  Length of previous line in nibs if writing
                   into memory
     Output buffer contains line to write if writing

Exit:
     P       =  0
     R4(15-14) = FIB#
    Carry set:
     C(A)    =  Error code:
                   Insufficient Memory, etc.
                   End of file (file is not altered)
    Carry clear:
     R3      =  #Nibs read or written, or offset if writing to
                   memory.
     S7      =  Set iff file at EOF after operation (sEOF)
     FIB spaced past line in file
     Output buffer collapsed if writing
Calls:     NIBLIO

Uses.......
```

Inclusive: A-D,R0-R3,D0,D1,P,STMTR1,STMTD1
S11-S9,S8(WRITNB only),S7,S6,S4-S0

Stk lvls: 4 plus 1 RSTKBF level

NOTE:
NO CHECK IS MADE whether the file is protected or in ROM.

Algorithm:

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/15/82 | FH | Designed and coded. |

## 9.8 OBEDIT - Edit Output Buffer

Category: FILUTL File: FH&TFM::MS

Name:(S) OBEDIT - Edit Output Buffer

Purpose:
Move the trailing portion of the output buffer, between a specified address and (AVMEMS), up or down by a given offset. Update AVMEMS and perform memory check when offset is positive.

Entry:
A(A) = Start of block to move (SOURCE).
C(A) = Offset of move (DEST - SOURCE). If positive, memory check will be performed.
P = 0 if leeway is desired should a memory check be performed.

Exit:
P = 0
Carry clear:
A(A) = Start of block to move (SOURCE).
B(A) = Length of block moved (old (AVMEMS)-SOURCE).
C(A) = DESTination of move (new start of block).
(AVMEMS) updated, now old (AVMEMS) + offset.

```
     Carry set:
      C(3-0) =  eMEM error code (Insufficient memory)

  Calls:      MEMCL+, MOVE*M

  Uses.......
   Exclusive: A(A), B(A), C(A),     D1, P
   Inclusive: A(A), B(A), C(A), D0, D1, P

  Stk lvls:   1

  History:

     Date     Programmer              Modification
   --------   ----------   ----------------------------------
   09/21/82      FH        Designed and coded.
```

## 9.9 RPLSBH - Replace Memory File Subheader

Category: FILUTL     File: FH&TFM::MS

```
  Name:(S) RPLSBH  -  Replace Memory File Subheader

  Purpose:
       Replaces the subheader of a memory file with the data
       stored in the output buffer.  For external files, write
       the output buffer data to the subheader area of the
       file.  Does NOT update the subheader length field of
       the FIB, but for memory files it updates the Data Begin
       field.  If out-of-place transform in memory file, it
       replaces the old subheaderunconditionally with the new
       subheader in output buffer.

  Entry:
       R4(15-14) = FIB# of dest file; file rewound.
       R3(A)  =  Length of old subheader
       P      =  0
       S5     =  1 iff In-place Transform (sTFINP)
       Output buffer contains new subheader

  Exit:
```

```
        P            =  0
        R4(15-14)  =  File FIB#
     Carry set:
        C(3-0)  =  Error code; insufficient memory

Calls:       LOCFIL, RPLLI*, FIB#RS

Uses.......
 Inclusive: A,B,C,D(S),D(7-0),R0,R1,R2,R3,D0,D1

Stk lvls:    4

NOTE:
     File is ASSUMED to reside in memory (internal file).


Algorithm:
     Adjust FIB pointers to make old subheader appear to be
          first line
     Replace this line with new subheader
     Adjust FIB pointers beyond new subheader again

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 10/04/82 | FH | Designed and coded |

## 9.10 SWPBYT - Swap Bytes

Category: FILUTL     File: FH&TFM::MS

```
Name:(S) SWPBYT  -  Swap Bytes

Purpose:
     Reverses A(3-2) and A(1-0).

Entry:
     A(3-0)  =  2 bytes to be reversed

Exit:
```

```
      A(3-0) =  Reversed bytes

   Calls:       None

   Uses.......
    Inclusive: A(A),C(A)

   Stk lvls:    0

   History:

      Date      Programmer              Modification
    --------    ----------   -----------------------------------
    09/21/82       FH        Designed and coded
```

## 9.11 CREATF - Create File in MAIN

Category: FILUTL File: JP&EXC::MS

```
   Name:     CREATF  -  Create File in MAIN
   Name:(S) CRETF+   -  Create file in MAIN or in IRAM

   Purpose:
        Create a file in designated RAM device.

   Entry:
       CREATF:
        C(A) = Total memory size of new file in nibbles
               (must include length of file header)
       CRETF+:
        C(A) = Total memory size of new file in nibbles
               (must include length of file header)
        D(S) = 0 or F  => Create in mainframe
             = other   => Create in PORT
                 D(B) determines in which port to create:
                   D(1) = PORT #
                   D(0) = Extent #
                 D(B) = FF => Create on first avail. port

   Exit:     R1 @ Start of new file  (from WFTMDT)
```

```
          B(A) = Total memory size of new file
         CARRY SET => NEW FILE WAS NOT CREATED
          C(3-0)   =   Error number
         CARRY CLR => FILE CREATED SUCCESSFULLY
          The following header info filled in:
          Flag field and COPY code field zeroed
          Creation time and date
          File chain length

Calls:    MOVED3, RFADJ+, WFTMDT, EOFLC+, ROMF-1, WFLENG
          LSTADR, ROMCHK, ROMFND, MEMCKL, RC01, RAMROM

Uses:     A-D,D0,D1,R0,R1,SCRTCH (32 nibs),S0-S7 (YMDHMS)

Detail:
          B = Size of new file          (Offset for pointers)
          R0= Size of new file        (Saved during WFTMDT call)
          R1 = Start of new file


Algorithm:
          Save size of new file (R0)
          If not Mainframe create                D(S) >= 1
             If PORT not specified               D(B) = FF
    1:          Find first avail port            (ROMCHK)
                  Error if no ports
                Try to create file on port       (CRTPRT)
                If not successful
                   Try next port                 (goto 1)
             else
                Find specified port              (ROMF-1)
                  Error if not found
 CRTPRT:        Error if Port not RAM            (RAMROM)
                Calc end of file chain
                Calc last address on Port        (LSTADR)
                If enough memory
                   Write zero byte @ file chain end
                   Back up to file header
                   Write Date and time           (WFTMDT)
                   Write file length
          else
            Check if enough memory w/LEEWAY to create
            Read End of Source (AVMEMS) --->(D0)
            End of Destination AVMEMS + File size --> (D1)
            Length of Source = End of Source - Begin of Source
                             = AVMEMS - (MAINEN - 2)
            Begin Source @ Zero byte of File Chain
            Move memory down                      (MOVED3)
            Zero flags,write Time, Date to hdr(WFTMDT)
            Write File length chain to header
```

```
            Save PRGMEN, CURREN              (R1)
            Adjust memory & stack pointers   (RFADJ+)
            Restore PRGMEN, CURREN
```

Stack lvls: 5
4 if file created in MAIN

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | S.W. | Added documentation |
| 07/15/82 | JP | Modified D(S) entry conditions |
| 10/11/82 | JP | Added LEEWAY check for MemChk |
| 12/17/82 | S.W. | Eliminated check for ROM - Trapped out in poll, as with other non-RAM memory devices |
| 01/10/83 | S.W. | Eliminated poll to CREATE on non-RAM device |
| 01/31/83 | S.W. | Always uses 5 stack levels |
| 03/17/83 | JP | Packed D1=(5) =MAINEN |
| 06/23/83 | S.W. | When adding file to an IRAM, now we guard against 'wrap-around'. Replaced GOVLNG RMEM w/ GOLONG RMEM10. |
| 06/29/83 | S.W. | Don't save CURREN on RSTK before calling RFADJ+ - uses too many levels - use R1 instead. |

## 9.12 WFTMDT - Write Flags, Time, Date to File Header

Category: FILUTL     File: JP&EXC::MS

Name:(S) WFTMDT - Write Flags, Time, Date to File Header

Purpose:
Zero Flags, Write Creation Time & Date to file header

Entry:
D0 @ File start

```
        WFTMD-:  Set flag to prevent Nib 2 of Flags to be zeroed
        WFTMDT:  Clear flag: Nib 2 of flags is zeroed

        Nib 2 of flags = COPY code nibble

   Exit:
        DO @ Time field of file header
        P=0
        R1 @ File start

        In RAM:
           Flag: 00
           Time: mmhh
           Date: ddmmyy

   Calls:      ST01, YMDHMS, RC01

   Uses.......
    Exclusive: A(A),C,P,DO,R1
    Inclusive: A,B,C,D,P,DO,D1,RO,R1,SCRTCH (32 nibs),SO-S7

        R1 = File start
        YMDHMS uses A-D,RO-R1,DO,D1,SO-S7
        ST01   uses A,DO,SCRTCH (32 nibs)
        RC01   uses RO,R1,DO,A

   Stk lvls:   3

   Detail:
        ST01 called to save RO-R1 in SCRTCH
          YMDHMS uses these registers
          RC01 restores RO-R1

   NOTE:
        This routine could be shorter if another scratch
        register or the stack was used to save the position
        within the file header @ Time

        Since this is a utility I'm trying to minimize the
        usage of R registers and subroutine levels

   2:   The positioning from the File start to the TIME field
        is through LENGTHs not OFFSETs.

   History:

     Date     Programmer              Modification
   --------   ----------   ------------------------------------
   07/04/82   JP           Modified documentation
```

## 9.13 PEDIT - Program Edit

Category: FILUTL File: JP&MEM::MS

```
Name:(S) PEDIT   -  Program Edit
Name:(S) PEDITD  -  Program Edit to delete line
Name:    PEDITM  -  Program Edit not collapsing stacks

Purpose:    Edit/delete line in current program

Entry:      PEDIT  =>
                   Edit line into current program
                   Line in output buffer
                    S8 is cleared
                   Stacks/SUSP prog cleared after
                     Protection Check
            PEDITD =>
                   S8 must be set
                   Delete Line
                   Line# to delete in output buffer
                   Stacks/SUSP prog cleared after
                     Protection Check
            PEDITM =>
                   MERGE command entry point
                   S8 must be clear, to avoid delete
                   PRIVATE and SECURE have already been
                     checked
                   Stacks will NOT be collapsed

Exit:
         Carry Clear
            R3= offset of memory at higher address
            Memory pointers updated
         else
            Error Exit
             Non BASIC file type          eFTYPE
             File protected               ePROT
             Unsuccessful replace of line

Calls:     FINDL+, SAVEL+, RPLLIN, OBCOLL, CHKPSF, CLPSTK
           NXTLIN, D1=CRS, DOOUTB, CLLINK
```

```
Uses:     A, B, C, D, D0, D1, OUTBS, R0-R3, S8
          If GOTO/GOSUB links are cleared, S1 is used

Detail:

 PEDIT:   Clear Delete Line flag
 PEDITD:  If current file type not BASIC or protected
             Error Exit
          Colapse stack, zero addresses, clear SUSP annun.
 PEDITM:  Zero Label chain and all GOTO links in file
             Move Output Buffer to end of available memory
          Set D0 @ start of line to Edit (@ OUTBS)
          Update CURRL to new line # (SAVELO)
          If null line (S8=1)
             Collapse Output Buffer
          Call FINDL to find a match on line# >=
          Set D = End of program memory (MAINEM)
          Compute old line length
          Replace line
          If unsuccessful
            THEN MFERR

Stack lvls: 5

History:

   Date     Programmer    Modifications
 --------   ----------    -------------
 07/08/82   SW            Updated documentation
 01/11/83   SW            Eliminated poll on non-RAM device
 03/02/83   JP            Packed GETPRe to CHKPSF
 03/03/83   JP            Moved PEDITM entry, CLPSTK call
```

## 9.14 FINDL - Find Line# within a Program File

Category: FILUTL    File: JP&MEM::MS

Name:(S) FINDL - Find Line# within a Program File

Purpose:
    Attempt to find passed in Line# within program and

```
        Return with pointer to start of line

   Entry:
        FINDLR: Read Line# @ D0 into C(A)
        FINDL : C(A) =   Line# to find
        FINDL+: B(A) =   Line# to find
        FINDL0: B(A) =   Line# to find
                C(A) =   Start of Search
                D(A) =   End of Search

        Assumes: File type = BASIC

   Exit:
        D(A) = End of CURRENT file
        D0   = Previous line found
             = 0 if No previous line found

        Carry set
               Line# found
               D1 @ Line#
               S0=0, S1=0

        Carry clear:
               S1=1        --->   NULL program     - D1 past EOF
               S0=1        --->   Line# not found  - D1 past EOF
               S0=0,S1=0   --->   Line# > found    - D1 @ line#

        If line# found      - Carry set
                              D1 @ Line# found
                              S0=0, S1=0
        If line# > found    - Carry Clear
                              D1 @ Line# > found
                              S0=0, S1=0
        If Null Program     - Carry clear
                              D1 points past EOF on file
                              S0=1, S1=1
        If line# not found  - Carry clear
                              D1 points past EOF on file
                              S0=1, S1=0
        Error Exit -
          None

   Calls:    NULLP, NXTLIN

   Uses.......
    Exclusive: A(A),B(A),C(A),D(A),D0,D1,S0,S1
    Inclusive: A(A),B(A),C(A),D(A),D0,D1,S0,S1

   Stk lvls: +3

   Detail:   NULLP
```

```
              Carry Set if Null Program
              D1= First line of file
              D = End of current file
              C = oBSsod
     Assumes: When end of program test done in FINDL
              If not null program, NOT @ end of program
              C (00011) is ALWAYS < D (End of program)
```

History:

| Date: | Programmer | Modification |
|---|---|---|
| 01/04/83 | JP | Removed S9 usage |
| 03/01/83 | JP | Updated documentation<br>NULLP does not Error Exit |

## 9.15 NXTLIN - Scan to Next Line

Category: FILUTL     File: JP&MEM::MS

Name:(S) NXTLIN - Scan to Next Line

Purpose:
    Scan from Line Number to End of Line Token

Entry:
    D1 @ Line Number

Exit:
    Carry Clear
    D1, C(A) POINT PAST EOL TOKEN

Calls:    None

Uses.......
 Exclusive: A(A),C(A),D1
 Inclusive: A(A),C(A),D1

Stk lvls:    +0

Detail:

USES IMPLEMENTATION OF '@' FOR MULTI-STATEMENT LINES

## 9.16 RDCHDR - Read Current File header, File length

Category: FILUTL    File: JP&SYS::MS

```
Name:(S) RDCHDR  -  Read Current File header, File length
Name:(S) RDCHD+  -  Read Current File header,File length and typ??
Name:(S) RDHDR1  -  Read File header, File length

Purpose:
     Read file header, return File length, possibly File type

Entry:
     RDCHDR:  Sets D1 = Start of Current File @ Header
              Assumes:
                 If P=0; File type read into R2
                 If P#0; File type not read into R2
     RDHDR1:  D1 @ Start of File @ header
              Assumes:
                 If P=0; File type read into R2
                 If P#0; File type not read
     RDCHD+:  Set D1 = Start of Current File
              Explicitly sets P=0
              File type will be returned in R2

Exit:
     Carry Clear
     D1 @ File length of header
      A = File length
         Current D1 + (A) = Next File in Chain

     If P=0
        R2 = File type

     P is NOT reset; necessary for GETSTC to call RDHDR1
     Calling routine must reset P=0 if desired.

Calls:      None

Uses.......
```

Exclusive: A(A),P,R2 (if P=0),D1
Inclusive: A(A),P,R2 (if P=0),D1

Stk lvls: 0

Detail: File Header Format:

| | |
|---|---|
| File Name | 16 nibbles |
| File Type | 4 |
| Flags | 2 |
| Creation Time | 4 |
| Creation Date | 5 |
| File Chain | 5 |
| Implementation | 8 |

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified Documentation |
| 01/04/83 | JP | Change S9 usage to P=0/P#0 |

## 9.17 GETSTC - Get Start/EOF Curr File/check Filetype

Category: FILUTL File: JP&SYS::MS

Name:(S) GETSTC - Get Start/EOF Curr File/check Filetype
Name:(S) GETST- - Get Start/EOF Curr File/don't check Filetype
Name:(S) GETST* - Get Start/EOF any file/check Filetype
Name: GETSTe - Get start/EOF Curr File/Error exit not BASIC
Name: GETPeF - Check protection & get file start/EOF

Purpose:
GETSTC,GETSTe:
Return first line of BASIC/Binary file & EOF
If P=0
Verify that file is BASIC, Error Return if NOT

See GETSTe for Error Exit if non BASIC file

```
      GETPeF:
        Check File protection
          Error exit if file protected
        Fall into GETSTC code
          Get start/end of BASIC file
          Error return if non BASIC file

   Entry:

      GETPeF: Checks file protections
              Falls into GETSTC
      GETSTC: D1 gets set to start of Current file
              Sets P=0
                   File type read into R2; Check if BASIC
                   Falls into BASCHK
      GETST-: D1 gets set to start of Current File
              Assumes P set on entry
                   Used for PNO entry
                   File type not read into R2, not checked
      GETST^: A @ Start of file
              Assumes P value on entry

      GETST1: D1 @ File length field of file
              A(A) contains file length
              If P=0
                 Checks file type in R2 for BASIC file type

   Exit:
      If GETPeF entry:
         If file protected:
            Error Exit to MFERR   (eFPROT)

       P=0
       D0  @ First line of file (at initial tEOL)
       D   = End of file
       A   = File length

       If PNO
          Carry Clear
          File type NOT in R2, file type NOT checked

       If P=0
          Fall into BASCHK
          If BASIC filetype
             Carry Clear
             R2  = File type
          else
             Error Return - C(0-4) = eFTYPE

   Calls:     RDCHDR, RDHDR1, GETPRO (GETPeF entry only)
```

```
Uses.......
 Exclusive: A(A),C(A),D(A),D0,D1,P,R2 (if P=0)
 Ixclusive: A(A),C(A),D(A),D0,D1,P,R2 (if P=0)

Stk lvls:   GETSTC,GETPeF,GETST1,GETST*,GETST1: 1
            GETSTe:       2

Detail:
         Positions to first line of file assuming:

         oBSsod = Offset to BASIC start of data, which
                  includes the permanent EOL.

         Must subtract length of EOL to position @ first line

History:

  Date      Programmer              Modification
--------    ----------    -----------------------------------
06/30/82    JP            Modified documentation
09/15/82    JP            Changed to Error Return, not Exit
01/04/83    JP            Changed S9 usage to P=0/P#0
03/01/83    JP            Added GETPeF entry point
```

## 9.18 BASCHK - Verify File Type in R2 is BASIC

Category: FILUTL     File: JP&SYS::MS

```
Name:(S) BASCHK  -  Verify File Type in R2 is BASIC
Name:(S) BASCHA  -  Verify File Type in R2 is BASIC

Purpose:
     BASCHK:
       Verify that File type in R2(A) is BASIC
     BASCHA:
       Verify that File type in A(A) is BASIC
     Error return if not

Entry:
     P=0
     BASCHK:   R2(A) = File type
```

```
     BASCHA:   A(A)  = File type

  Exit:
     P       = 0

     If File type = BASIC
        Carry Clear
           R2(A) = File type
           A     = Preserved from Entry
     else
        Carry Set
           Error Return  C(0-4) = eFTYPE
           R2(A) = File type
           A(A)  = File type

  Calls:      None

  Uses.......
   Exclusive: C,R2
   Inclusive: C,R2

  Stk lvls:   BASCHK: 0
              GETSTe: 2

  Detail:     This code must IMMEDIATELY follow GETSTC

  History:
```

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |
| 09/15/82 | JP | Changed to Error return/not exit |
| 12/17/82 | JP | Added BASCHA entry |
| 01/04/83 | JP | Added P=0 at end, due to GETSTC |
| 03/01/83 | JP | Remove GETS-e entry due to NULLP |
| 04/25/83 | JP | If non BASIC, R2 = filetype |

## 9.19 FCHLBL - Find Label in Current BASIC File

Category: FILUTIL     File: JP&SYS::MS

Name:(S) FCHLBL - Find Label in Current BASIC File

Purpose:
Find a label in the current BASIC file
Assumes current file is BASIC

Entry:
Assumes current file is BASIC
R3 = Label to find
Right justified with trailing blanks

Falls into COMPL#

Exit:
P = 0

Carry Clear - Label Found
D0 @ EOL preceding line containing Label
D1 @ Line # of line containing Label

Carry Set - Label Not Found in Current file

Calls: GETSTC,TKSCN7,LBLNAM

Uses.......
Exclusive: A,B(A),C,D(A),D0,D1,R2,P
Inclusive: A,B(A),C,D(A),D0,D1,R2,P

Stk lvls: 2

Detail:
Fall into COMPL# to compute Line# after Label found
This code must IMMEDIATELY precede COMPL#

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |

## 9.20 COMPLN - Compute Line # with D0 @ line length

**Category:** FILUTL **File:** JP&SYS::MS

```
Name:     COMPLN  -  Compute Line # with D0 @ line length
Name:(S)  CPLN10  -  Compute Line # with D0 anywhere in stmt
Name:     CPLN15  -  Compute Line # with C anywhere in stmt

Purpose:
     Compute Line # from position within statement

Entry:
     COMPLN:  D0 @ Line length of statement
     CPLN10:  D0 @ anywhere within statement
     CPLN15:  C  @ anywhere within statement

Exit:
     Carry Clear => Line # found
        P       =  0
        D1      @  Line #
        D0      @  EOL preceding the Line#  (D0=D1-2)
     Carry set => The input pointer is not pointing at
        current file.

Calls:      GETST-,

Uses.......
 Exclusive: A(A),B(A),C(A),D(A),D0,D1
 Inclusive: A(A),B(A),C(A),D(A),D0,D1

Stk lvls:   2

Note:  This routine will not check file type, it assumes the
       current file is type BASIC.

Detail:     Do not call this routine if specified address is
            at initial tEOL.  If at low nib of initial tEOL
            will return with carry set; if at high nib of
            initial tEOL will not work properly - found in
            code review (S.W.)

History:

  Date      Programmer            Modification
 --------   ----------   ----------------------------------
 06/30/82   JP           Modified documentation
```

| | | |
|---|---|---|
| 01/04/83 | JP | Removed S9 usage |
| 03/30/83 | SC | Modified documentation |
| 04/15/83 | JP | Fixed check for @ (F4) |

## 9.21 PFINDL - Find Line# Within Program

Category: FILUTL File: JP&SYS::MS

```
Name:(S) PFINDL  -  Find Line# Within Program
Name:(S) PFNDZL  -  Find Line# Within Program
Name:    PFNDL*  -  Find Line# Within Program

Purpose:
     Find Line# between current program boundary

Entry:
     PFINDL:
       P = 0
       Assumes PRGMST, PRGMEN are current and updated
       D0 past Line# token
       Clears sXWORD (S9) flag to use Compiled Line# reference
     PFNDZL:
       P = 0
       Same entry as PFINDL
       D0 past Line# token
       Assumes sXWORD (S9) is set so:
         Will always search for Line#
         Allows XWORD entry, to search for Line# and not
         rely on compiled line# address, which may be bad.
     PFNDL*:
       P = 0
       D @ End of range to search for Line#
       D0 past Line# token
          Used by RENUMBER

Exit:
     P      =  0
     D0     =  D0 on entry  (past Line# token)

     Carry Set - Line# found
         D1 @ Line#
```

Reference to Line# (entry D0) is "compiled"
Relative address to line# is filled in

Carry Clear - Line# not found

Calls: SCOPCK,ISRAM?

Uses.......
Exclusive: A(A),B(A),C(A),D(A),D1,D0,
sXWORD (S9-PFINDL/PFNDZL only)
Inclusive: A,B(A),C(A),D(A),D1,D0,
sXWORD(S9-PFINDL/PFNDZL only)

Stk lvls: 2

NOTE:
This routine will search between PRGMST & PRGMEN only
if PFINDL or PFNDLZ is called.

PFNDZL will always search for Line# (if sXWORD set)

PFNDL* uses D(A) for boundary

Detail:
If not XWORD entry:
It will look at the compiled address field following
the line number first.
If the compiled field is non-zero
Compute the address of the Line#
else
Search the entire program
Write the compiled address to RAM if Line# found

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified/Added Documentation |
| 02/04/83 | JP | ISRAM? call uses all of A |
| 02/22/83 | JP | Added PFNDZL entry added |
| 02/22/83 | JP | Added S9 (sXWORD) usage |
| 03/08/83 | JP | If not running; always search |

## 9.22 NULLP - Null Program Check

Category: FILUTL    File: JP&SYS::MS

```
Name:(S) NULLP  -  Null Program Check

Purpose:
     Check if current BASIC program is NULL
     Position to First line | EOF

Entry:
     File type will be checked if P=0
     If P=0: File Type is returned in R2 (from GETST-)
     If P#0: File Type will not be returned, nor checked.

     Assumes: Length to data = Length to data of
              BASIC/ Binary file
     Assumes File type = BASIC or Binary or file with same
                         structure

Exit:

     Carry Set   - Null program
     Carry Clear - not Null program

       P  = 0
       D1 = First line of File (@ EOF)
       D = End of program

       From GETST-
         A  = File Length
         R2 = File Type          (if P=0 on entry)
         D0 = First line of File

Calls:      GETST-

Uses.......
 Inclusive: A(A),C(A),P,D0,D1
 Exclusive: A(A),C(A),D(A),P,D0,D1,R2 (If P=0 on entry)

Stk lvls:   2

Detail:   Get start and end of current file (GETST-)
          Move First Line of file pointer to D1
          If file length = Offset to BASIC start of data
             RTNYES  (HULL program)
```

Null Program File Length = Offset BASIC start of data

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |
| 01/04/83 | JP | Remove S9 usage |
| 03/01/83 | JP | Removed Hardwire Error Exit |

## 9.23 CHAIN+ - Chain Subprograms, Labels, DEF FNs

Category: FILUTL File: JP&SYS::MS

```
Name:(S) CHAIN+  -  Chain Subprograms, Labels, DEF FNs
Name:(S) CHAIN-  -  Chain Subprograms, Labels, DEF FNs
Name:    CHAIN*  -  Chain Subprograms, Labels, DEF FNs

Purpose:
     Chain all Sub-programs in a file
     Chain all Labels in a file
     Chain all Def FNs in a file

Entry:
     P      =  0
     Assumes Current file is BASIC

     CHAIN+:   Chain Current File
     CHAIN-:   A @ Start of file to chain
     CHAIN*:   D1 @ Sub-link of file
               D  @ End of file

Exit:
     P      =  0
     D(A) = End of file
     D1   @ Sub-link of file

     Error Exit - if file not in RAM
                  eFACCS - " Illegal Access"

Calls:     FNDDO+ (FINDA), ISRAM?
```

Uses.......
 Exclusive: A,B(A),B(S),C,D(A),D0,D1,R2
            CHAIN*: D1 is preserved

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | JP | Modified documentation |
| 01/17/83 | S.W. | Updated/Expanded documentation |

## 9.24 FILCRD - Copy File To Card

Category: FILUTL     File: MN&CD::MS

Name:(S) FILCRD - Copy File To Card

Purpose:
    Copy file from memory to card.

Entry:
    C[A] points to start of file header.
    R1 contains name to be used on card. Zeroes if no
      name specified (use name of file).
    S8=1 if private card requested.

Exit:
    Returns if write completed.
    NXTSTM if write aborted.
    Error exits:

Calls: ALIGN, BLANKC, CHKSUM, CMPTIM, CR??, CRDOFF, CSLW5, D1+13B, D1+21B, D1+29B, DAYYMD, FNDPRT, FROMDT, FTYPF#, IMPFLD, IOAL36, LCTRKS, MAXTRK, POLL, PREPDT, PREPHD, R1TOD0, RC01, RD8SV, RDSOC, RDYTRK, READCS, READFL, RTODP, ST01, STORG?, TOCARD, TODT, TRKDON, VFYCRD, WAITM+, WRIT8S, WRITE, WRITFL, WRMSG, WRT2-0, YMDDAY, aslw5, asrw5, crlfnd, csrw5, fpoll, idiva,

```
            noscrl.

   Uses.......
            A,B,C,D,P,D0,D1,ST,R0-R4,
            SNAPBF, SAVSTK, SCRTCH, 8 levels in RSTKBF.

   Stk lvls:  4

   Detail:
        Card format chosen for compatibility with HP-75.
        Card is divided into four fields, each preceded by a
        hardware recognized flag and followed by a zero byte.
        Fields are separated by a 66 ttfc (timetrack flux
        change) gap.  fields are as follows:

   ****  Start-of-Card: recorded at factory when timetrack
                         is recorded.
           SOC marker: "HP"  (2)
           format: "CV"  (2)
           size: # bytes available after write-protect field
                 (specific to Corvallis format)  (2)
                 for 10" cards: 2BC (=700 base 10)
           (reserved): 0000 (2)

   ****  Write-protect: 4-byte field:
           0000 for write-enabled cards.    \ (2)
           FFFF for write-protected cards.  /
           (reserved): 0000 (2)
           padding added by HP-75  (1)

   ****  Data Header: identifies file, contains security
                      information.

        % identifies fields which differ between HP-71 and
          HP-75 format.  HP-75 format is only used
          for LIF1 (text) files.

      % 0:  sub-format (1): 00 for LIF1 file (HP-75 subformat)
                            01 for HP-71 files (HP-71 subfmt)
        1:  track# (1)
        2:  # of tracks in set (1)
        3:  # bytes in this track (2)
        5:  # bytes in file (2)
      % 7:  file type (2): HP-75 filetype (HP-75 subformat)
                           LIF filetype (HP-71 subformat)
        9:  creation date (4): hex seconds since start of
                               century.
       13:  file name (8)
     % 21:  password (4): blanks for LIF1 filetype (HP-75)
            implementation (4): (HP-71 subformat)
       25:  marker (2): checksum of entire file, including
```

file header.
27: partial statement status (1)
28: s1 (2)
30: s2 (2)
32: data checksum (2): 2-byte checksum of data field.
34: header checksum (1): 2-byte sum of header field, folded to one byte without wraparound carry.
35: (reserved) (1)
padding added by HP-75 (2)

File headers for the two subformats differ only in bytes 0, 7-8 and 21-24.

**** Data: 650 bytes

padding added by HP-75 (3)

All files except LIF1 will use LIF filetype in the filetype. For the curious among you, HP-75 filetypes consist of two bytes:

```
high order byte: 00=HP-75 system file
                 ??=HP-75 text file
                 ??=HP-75 basic file
                 ??=HP-75 appointment file
                 ??=HP-75 lex file
                 ??=HP-75 keds file
                 "I"=LIF1
```

low order byte: HP-75 attribute byte. Identifies file capabilities. bit masks as defined by HP-75 are:

```
80=in rom
40=file runnable
20=file editable
10=file listable
08=file purgable
04=file copyable
02=standard lined file
01=token file
```

two important bit masks are:

```
34=private file
7E=data file for print#/read#
```

HP-75 documentation identifies some basic file types:

```
0062=calculator file
0000=system file
```

013E=text file
027E=basic file
030C=appointment file
050C=alpd fild
0008=diagnostic file

----------

Function of scratch registers in card write:
R0=scratch
R1[A]=file pointer
R1[9-5]=amount of zero-padding at end of file in bytes (used to bring LIF1 to sector boundary).
R2=pointer to I/Obuffer containing header

Function of status bits:
S1: Used by Verify to suppress DATA ERROR in read from FIFO.
S2: Indicate we are on last track of card set.

Algorithm:
Check for presence of card reader; eDVCNF if absent.
Allocate I/O buffer for building header; eMEM if no room for buffer.
Fetch filetype from file. If not copying to PCRD then goto 2.
Search for filetype in filetype table. If found then goto 1.
If filetype not in standard range then goto 2.
Set privacy bit in filetype. Goto 2.
1: If there are < 3 entries in filetype table for this filetype then goto 2.
Read third entry (private) from filetype table.
2: Store filetype in card header I/Obuffer.
Store passed destfile name in header I/Obuffer.
Compute time (seconds since start of century) and store in header I/Obuffer.
If we are writing out LIF1 filetype then write HP-75 LIF1 filetype to filetype field and 00 to subformat field, else write 01 to subformat field.
Write 01 to track# field in header I/Obuffer.
If copycode=8 then poll for somebody to copy card; eFYPE if not handled.
Compute file length in bytes: (chain length-5)/2 if copycode#1, (chain length-13)/2 if copycode = 1; rounded up to byte. If > FFFF bytes then eF2BIG.
If filetype = LIF1 then pad file length up to sector boundary (256 bytes).
Write file length to header I/Obuffer.
Compute implementation field, write to header

I/Obuffer.
Perform checksum of entire file for "marker" byte--byte which uniquely identifies card set. Write to header I/Obuffer.
Compute # tracks in card set. Write to header I/Obuffer.
3: Read track# and maxtrack# from header I/Obuffer. Deallocate buffer and return if track# > maxtrack#.
Compute trksize. Write to header I/Obuffer.
Compute checksum of this track. Write to header I/Obuffer.
Write 0's to partial card recovery fields (since recovery is not implemented).
Compute header checksum. Write to header I/Obuffer.
Perform WCRD poll.
4: Prompt "Wrt: Align then ENDLN" and wait for ENDLINE or ATTN or f-ATTN or timeout.
If ATTN or f-ATTN or timeout then abort.
Prompt "Pull xxx of xxx".
{card now starts moving.}
Verify start-of-card (SOC) field. If wrong then eUNKCD and goto 4.
Read write-protect field. If not 0's then ePROTD and goto 4.
Switch to write mode. Write 16 nibbles of 0's.
Write BREAK to card.
Write header I/Obuffer to card.
Write 16 nibbles of 0's to card.
Write BREAK to card.
Write data field to card, padding with 0's as necessary for text files.
Turn off card reader.
5: Prompt "Vfy: Align then ENDLN" and wait for ENDLN.
Prompt "Pull xxx of xxx".
Verify SOC field. If fail, eUNKCD and goto 5.
Skip write-protect field.
Verify header field. If error, eVFYER and goto 4.
Verify data field. If error, eVFYER and goto 4.
Turn off card reader.
Update file pointer and increment track#. Goto 3.

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/12/82 | NM | Added documentation |
| 02/25/83 | NM | Updated "CALLS" section |

## 9.25 CRDFIL - Copy Card Into RAM

Category: FILUTL    File: MN&CD::MS

Name:(S) CRDFIL - Copy Card Into RAM

Purpose:
Copy a file from card into memory.

Entry:
R3 = name of file to look for on card (zeroes if not specified).
R2 = name of file to be used in RAM after it is read in (zeroes if not specified).

Exit:
Returns if successful.
R2[A]=pointer to file header of file just read in.
If read fails, this code performs an error exit and does NOT return.

Calls: ASRW4, CHKSUM, CLRALL, CMPALL, CMPWRT, CR??, CRDFAB, CRDOFF, CREATF, DONIBC, D1+13B, D1+29B, FILEF, FNDCLR, HDRHDR, IOAL36, LAKEYS, LC2TRK, MAKHDR, MEMCKL, MOVED3, NOCOMP, OFFSET, PLLCRD, R1DO37, R1TODO, RALIGN, RDSOC, RDYTRK, READ8S, READCS, READFL, RTODP, RWERR, SETBIT, SWPBYT, TRKDON, WRMSG, aslw5, asrw5, crlfnd, csrw5, fpoll, idiva, mvmem+, noscrl.

EXITS through BUFDAL.

Uses.......
A,B,C,D,P,DO,D1,RO-R3,ST, SCRTCH, SNAPBF, 8 or so levels in RSTKBF.

Stk lvls: 5

Detail:
Creates an I/Obuffer for the card header and then creates the biggest possible file in the available memory. Setting aside as many nibbles at the end as are necessary to maintain a tracks-read bitmap, reads the card into the file and then collapses the file to the proper size after the read.

Register usage in CRDFIL routine:
  R1[4-0]=address of data area in file (past header).
    [9-5]=amount of available memory in data area.
    [14-10]=size of bitmap.
  R2=pointer to header.

Algorithm:
    Perform MEMCHK on (size of file header) + (size of card
      header I/Obuffer) + (leeway); eMEM if failure.
    Compute remaining space (B=C-B).
    Add headersize for full file size (C=B+HDRSIZ).
    Create file {this creates the biggest allowable file,
      with R1 pointing at start of file header}.
    Write filename passed in R2 to file header.
    Zero out filetype field in file header.
    Hold address-of-file-data-area and size of file-
      without-header in R0[9-5] and R0[A], respectively.
    Determine size of bitmap needed { (#nibs in file data
      area)/(#nibs in four tracks) + 1 }.
    R1[A]=address of file data area {past header},
      R1[9-5]=(size of data area) - (size of bitmap)
        {eMEM if subtraction generates carry},
      R1[14-10]=size of bitmap {located at end of data
        area}.
    Clear all bits in bitmap.
    Create card header I/Obuffer.
    R2[A]=pointer to buffer area {past header}.
 1: Send READ alignment message to display; CRDFAB, RTNABT
      if abort indicated by RALIGN.
    Send PULL CARD message to display.
    R4[S]=0 {indicate read has not occurred}.
    Read SOC and WPROT (RDSOC); goto 1 if error.
    Set FLGSRV.
    Read card header into card header I/Obuffer (MAKHDR);
      goto 1 if error.
    Read filename passed from file header. If nonzero and
      doesn't match filename on card; eWRGNM and goto 1.
    {We have now determined the card header; hence
      filesize, name, etc. There is no turning back.}
 2: Set FLGSRV.
    Compute offset for this trk based on trk# (OFFSET).
    D[A]=#full {8-nibble} FIFO reads; D[S]=size of partial
      read * 2.
    If track will not fit in available memory, error out
      with eMEM.
 3: If D=0 goto 4.
    Read 8 nibs from FIFO.
    Write at D0.
    Increment D0.
    Goto 3.
 4: If there is no partial read goto 5.

```
         Perform partial read.
         Write at D0.
      5: Turn off card reader.
         Compute checksum of data just read.
         Compare to checksum in header; if not match eRWERR and
           goto 6.
         Set bit corresponding to current trk# in bitmap.
         R4[S]="F" {indicate READ has occurred}.
      6: If filename in file header = 0, copy name from card
           header I/Obuffer.
         Search for filename in file chain.
         If address found # address of this file then error out
           with eFEXST.
         If filetype in file header <> 0 goto 7.
         Compute filetype and security based on filetype in
           card header I/Obuffer (HDRHDR).
         If filetype unrecognized and not standard range then
           error out with eFTYPE.
         If filetype unrecognized and standard range and private
           then error out with eFPROT.
         Write unencoded filetype and flags to file header.
         Read filename from file header.
         If filename <> "keys    " then goto 7.
         If unencoded filetype <> =fKEY then error out with
           eFTYPE.
      7: Check if whole card set fits.  If not
           then error out with eMEM.
         Compute max trk#.
         Write max trk# to card header I/Obuffer.
         If R4[S]<>0 then send "Trk #xxx done" to display.
         Find first unread trk# (FNDCLR in bitmap).
         If next trk# > max trk# then goto 8.
         Send READ alignment message to display.
         If abort, deallocate file and exit through RTNABT.
         Read SOC and WPROT; goto 6 if error.
         { Now we will copy the card header to the card header
           I/Obuffer, selectively comparing nibbles as we go.
           If a read error occurs; goto 6.  If a compare error
           occurs, give eNOTST warning and goto 6.}
         Copy card header to card header I/Obuffer, comparing
           bytes 0, 5-26.5 {lonib of byte 26}.
         Compare header checksum with value on card.  Warn with
           eRWERR if not match and goto 6.
         Goto 2.
      8: { At this point, C[A] contains the length of the data
           area}.
         C[A]=C[A]+5 {compute file chain length}.
         If filetype <> LIF1 then goto 9.
         { File length on card is a multiple of one sector,
           which in general pads the LIF1 file a whole bunch.
           We seek to crunch the file down to its proper size.}
```

```
Chain through LIF1 file looking for last record {FFFF}.
  If found {file not corrupt}, use smaller chain len.
Stash implementation field in R3.]
If copy code = 0, then implementation field[A] is the
  actual file length; add 5 for chain len.
{ We now have the file chain len--either from card
  header, looking at LIF1 chain or imp field}.
Write file chain len to chain len field in file hdr.
Compress file to proper length.
If copy code=1, retrieve implementation field; insert
  into file after chain length; modify chain length.
Send CR-LF to display.
Return.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/14/82 | NM | Added documentation |
| 02/25/83 | NM | Updated "CALLS" section |

## 9.26 WSTRFX - Write a String to a DATA File

Category: FILUTL    File: SC&DAT::MS

Name:(S) WSTRFX - Write a String to a DATA File

Purpose: Write a string to the fixed length data file
If the file is in an external mass memory device, data will be written to its I/O buffer first. When the I/O buffer is full or the file is closed, the content of the I/O buffer will be written back to the file.

```
Entry:  A = string length in bytes
        B = # of bytes left in current record
        R0(A)= Current file pointer
        R0(15,14)=Current byte ptr in file I/O buffer
        R1= record length in bytes
        D1 @ past the string ( String is stored backward)
        STMTD1 Contains FIB entry address
```

```
          S9 = 0 if serial access
             = 1 if random access
          S10 = 0 if internal file
              = 1 if external file(write file I/O buffer)
          It is assumed that there is enough room left in the
            file.
          If the string is too long to fit into the current
           record and it is a serial access, the string will be
           broken down into smaller logical units.

Exit:    Carry set => Random access crossing record boundary
         Carry clear => Done successfully
           R0(15,14) & R0(A) will be maintained

Calls:   D0+2WR, WRBYTC

Uses:    A,B,C,D0,R0, ST[4-0]

Stk lvls:  1 if internal file
           4 if external file (when flush file buffer)
```

9.27 WRTSTR - Write a string to an open TEXT file

Category: FILUTL    File: SC&DAT::MS

```
Name:(S) WRTSTR  -  Write a string to an open TEXT file

Purpose:
     Write a string on stack to an open TEXT file.

     The string will be written out as:
     +--------+--------------------------+----+
     | Length |     String               | Pad|
     +--------+--------------------------+----+
      2 bytes       n bytes               1 bytes

     The pad is not included in the length and it will be
     there only if the string length is an odd number.

Entry:
    D1 @ string length(2 nibs past the string header on
         math stack).
```

R0 = Current file pointer
S6 =1 If length odd
STMTD1 = Entry address in FIB
It is assumed that there is enough room left in the file to store the string.

Exit:
R0(A) will be updated
The string is popped and the AVMEME is update.
Current position in FIB will be updated too.

Calls: DROPST,WRBYTC,UPCPOS,WRTEOF,BACK2B,SETWRT

Uses.......
Inclusive: A,B,C,D0,D1,ST[4-0] ???

Stk lvls:
Internal file: 2
External file: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| | SC | Wrote |
| 11/05/83 | BS | Updated documentation |

## 9.28 WRTNUM - Write a Number to DATA or SDATA file.

Category: FILUTL File: SC&DAT::MS

Name:(S) WRTNUM - Write a Number to DATA or SDATA file.

Purpose: Write a number from math stack to a file of type DATA or SDATA.

A number will always be written out as a real(8 bytes):

```
High                                                    Low
-----------------------------------------------------------
| M0,M1| M2,M3| M4,M5| M6,M7| M8,M9|M10,M11| E0,MS| E1,E2|
```

---

```
Entry:  A= the number (internal form)
        S10 = 0 if the file is in memory.
            = 1 if the file is in an ext. mass memory device.
        D0 @ Current file pointer
          If the file is in memory, D0 is directly pointing
          at the file.
          If the file is in an external mass memory device,
          D0 is pointing at the I/O buffer of the file and
          R0(15,14) = Byte pointer of the file I/O buffer
        D1 @ Top of stack
Exit:   D1 will drop 16(D1=D1-16) and stored to MTHSTK
        D0 Past the number

Used:  A,B,C,D0,D1

Detail: The number will be formatted and written on the math
        stack first, and then it will be written out to the
        file or I/O buffer one byte at a time. If is written
        to an I/O buffer, when the buffer gets full, this
        routine will POLL the HP-IL ROM to dump the buffer to
        the device and read in the next buffer.
```

## 9.29 RDLNAS - Read String Length from a TEXT File.

Category: FILUTL      File: SC&DAT::MS

```
Name:(S) RDLNAS  -  Read String Length from a TEXT File.
Name:    RDLNFX  -  Read String Length from a DATA File.

Purpose:  RDLNAS - Read string length from a LIF1 file
          RDLNFX - Read string length from the fixed length
                   file.

Entry:    D0 @ current file pointer, absolute addr if file
            in RAM/ROM, absolute address in file I/O buffer
            if file is in external device.
          R0(15,14) = current position in file I/O buffer if
            file is in external device.
          STMTD1 contains FIB entry address
```

Exit: A(A) = The two bytes read from the file
DO @ past the two length bytes
The file pointer in the FIB is not updated. However, if the string length is read from the I/O buffer, there is a possibility that the I/O bufffer is overflowed and the next sector is read into the I/O buffer. In this case, if want to back up the DO by two bytes, call the routine BACK2B.

Calls: RDBYTA

Uses: A, C, DO

Stk lvls: +3

## 9.30 RDBYTA - Read Byte From an Opened File Into A

Category: FILUTL File: SC&DAT::MS

Name:(S) RDBYTA - Read Byte From an Opened File Into A
Name:(S) DO+2RD - Move file pointer&check buffer overflow

Purpose: Read a byte from an file into A-reg.
Reading a byte from memory can be easily done by one instruction "A=DAT0 B". But if the byte is read from an I/O buffer, then the possibility of overflowing the I/O buffer should be considered. This routine takes care of this problem automatically.

Entry: DO @ current file pointer(abs.addr. if file in RAM or ROM, absolute addr @ file I/O buffer if file in external device)
R0(15,14) = Current byte position in the file I/O buffer if the file is in external device
STMTD1 contains FIB entry address
S10 = 0 if file is in RAM or ROM.
= 1 if file is in an external mass memory device.

Exit: A(B) = The byte
DO past the byte.
R0[15,14] is updated.
Current position in FIB will be updated if need to

read in next sector from the external file.

Calls: D0+2RD, POLL(pRDNBF)

Uses:
  Internal file: Nothing
  External file: A(14,5), B(15,5), C, D0, R0, ST[4-0]

Stk lvls:
  Internal file: 0
  External file: 3

## 9.31 WRBYTC - Write Byte to an Opened File From C

Category: FILUTL     File: SC&DAT::MS

Name:(S) WRBYTC - Write Byte to an Opened File From C
Name:    WRBYTD - Write a Byte to an Opened File

Purpose: Write a byte to a file in RAM/ROM or to a file I/O buffer if the file is in external device

Entry: D0 @ current file pointer(absolute addr if file in RAM/ROM, absolute @ file I/O buffer if file is in external device.
R0(15,14) = Current byte position in file I/O buffer if the file is in an external device.
S10 = 0 if the file is an internal file
1 if the file is an external file

WRBYTC: C(B) = The byte to write
WRBYTD: D1 @ The byte to write to the byte to be written

Exit: D0 past the source byte.
For an external file:
R0(15,14) will be updated.
If overflow the I/O buffer, current buffer will be written back to the file, next sector will be read into the I/O buffer, current position in FIB will be updated.

WRBYTD: D1 past the byte

Calls: D0+2WR, POLL(pRDNBF)

Uses:
Internal file: D1
External file: A(14,5), B(15,5), C, D0, R0,ST[4-0]

Stk lvls:
Internal file: 0
External file: 3 if have to flush the I/O buffer.

## 9.32 BACK1B - Back up the File Pointer by 1 Byte

Category: FILUTL File: SC&DAT::MS

Name:(S) BACK1B - Back up the File Pointer by 1 Byte
Name:(S) BACK2B - Back up the File Pointer by 2 Bytes
Name:(S) BACK3B - Back up the File Pointer by 3 Bytes

Purpose: Sets the current position field of the file's FIB back the specified number of bytes. If the new position falls in the previous sector, it is read into the file's I/O buffer.

Entry: P= 0
R0(15,14) = Current byte pointer in the buffer
R0(4,0) = Current absolute address in the buffer
S10 = 0 - Internal file
1 - External file
STMTD1 contains file FIB address

Exit: P = 0

Calls: POLL(pRDCBF)

Uses: A,B,C,D0,P

Stk lvls: 0 - internal file
4 - external file (if has to back up)

## 9.33 UPCPOS - Update FIB Current Position

Category: FILUTL File: SC&DAT::MS

Name:(S) UPCPOS - Update FIB Current Position

Purpose: Update current position in FIB

```
Entry:  D0 = Current file pointer or buffer pointer
        R0(15,14)= Byte pointer in buffer if external file
        R1(A) = Record length if fixed length data file
        S9 = 1 for IRAM
        S10 = 0/1 for internal/external file
        S11 = 0/1 for serial/random access
        STMID1 = Entry address in FIB

Exit:  Update current position in FIB
       The D0 on entry is saved in R0(4,0)
       If is DATA file (copy code = 1) :
        Carry set => The file pointer is at the beginning
          of a record and the random access flag is set (S9).
        A(A) = Number of bytes left in current record.
        B(A) = Byte position in current record.
```

Calls: IDIV

Used: A,B,C,D0,R0,P (B is used only for DATA file)

Stk lvls: 1

## 9.34 GTPTRS - Get File Pointers from FIB

Category: FILUTL File: SC&DAT::MS

Name:(S) GTPTRS - Get File Pointers from FIB
Name:(S) GTPTRX - Get File Pointers from FIB

Purpose: Get all the file & FIB pointers into CPU registers

Entry: STMTD1(4-0) = Entry address in FIB
GTPTRX: Should clear S9 & S10 on entry

```
Exit:  D(S) = Copy code of the file
       D(A) = # of bytes to end of file
       B(S) = Device type
       B(A) = # of bytes left in current record
       R0(A)= Current position (absolute address)
       R0(15:14) = Relative position in buffer if external
       R1    = Record length in bytes
       S9   = 0 if serial acces
            = 1 if random access
       S10  = 0 if mainframe RAM/ROM file
            = 1 if is an external file
       S11  = 1 if Independent RAM
            = 0 if not IRAM
```

GTPTRX:
The difference between the two entry points is that in order to determine whether it is a serial or random access.
The GTPTRS entry will go back to the beginning of the statement to check if the record number is specified.
But the GTPTRX entry will not do so, therefore the S9 will not be changed by the GTPTRX entry.

Calls: I/OFND

Stk lvls: 2

Used A,B,C,D,D0, S9-11

## 9.35 FTYPF# - Look Up File Type Given Type Number

Category: FILUTL File: SC&FIL::MS

Name:(S) FTYPF# - Look Up File Type Given Type Number

```
Name:     FTYPFD  -  Look Up File Type Given Type Number

Purpose: Searches the mainframe and LEX File type tables for
           a given file type number.
         A pFTYPE poll is issued to search file type table in
         external LEX file if the mainframe file type table
         does not contain the file type.

Entry:
         FTYPFN:
           A(A) =  File type #  (high nib = 0)
         FTYPFD:
           D1 pts to file type #

Exit:
         D1 preserved
         RO     =  D1 entry state.
         Carry set => C(A) and B(A) point to start of entry
                      B(S)=position of file type# within
                           entry (1 = first filetype, etc.)
                      A(A)  =  File type number
         Carry clear => not found

Calls:   POLL, FTBSCH

Uses:
  Exclusive:  A(A),          C, RO
  Inclusive:  A(A),B(S),B(A),C, RO

Stk Lvls:  2
```

## 9.36 FTBSCH - Search a File Type Table by Type Number

Category: FILUTL     File: SC&FIL::MS

```
Name:(S) FTBSCH  -  Search a File Type Table by Type Number

Purpose:  Searches file type table by file type number.

Category: FILUTL
```

```
Entry:     A(A) = file type to search for (high nib = 0)
           D1 points at start of table

Exit:      A(A) = entry state
           Carry set => B(A) = pointer points to start of
                               entry
                        B(S) = position of filetype # within
                               entry
           Carry clear => not found

Uses:
  Inclusive: B(S),B(A),C(S),C(A),D1

Calls:     None
Stk lvls: 0
```

## 9.37 FASCFD - Look Up File Type Given Type Name

Category: FILUTL    File: SC&FIL::MS

```
Name:(S) FASCFD  -  Look Up File Type Given Type Name

Purpose:   Search the mainframe and LEX file type tables for a
           given file type number. A pFASCH poll is issued to
           search the LEX file type table if the mainframe
           file type table does not contain the file type.

Entry:     D1 points at the beginning of the file type name
            which is up to five characters with trailing
            blanks.

Exit:      D1 past the given file type name.
           P= 0
           Carry set => A(3-0) = file type number
           Carry clear => File type not found.

Calls:     POLL, FILEP!, FASCH,

Uses:      A,B,C,R3, S10

Stk lvls: +3
```

## 9.38 REWIND - Rewind Open File

Category: FILUTL File: SC&FIL::MS

Name:(S) REWIND - Rewind Open File

Purpose: Set the current position in the FIB to start of of data in a file.

Entry:
A = FIB entry address of the file

Exit:
A(B) = FIB # of the file
STMTD1 = FIB entry address of the file
Carry set => successful
Never returns if HP-IL error happens, exit to MFERR.

Calls: STFPTR

Uses: A, B, C, D, D1, D0, S10, STMTD1, S4-0

Stk lvls: 1 - internal file
4 - external file

## 9.39 FIBADR - Find FIB entry address for a channel

Category: FILUTL File: SC&FIL::MS

Name:(S) FIBADR - Find FIB entry address for a channel
Name:(S) FIBAD- - Find FIB entry address for a channel

Purpose: Find the FIB entry address for a given channel #

Entry: A(B) = Channel #

EXit: D1 & A = FIB entry address of the file
STMTD1 = FIB entry address of the file

Calls: FDCH#, FFIB#

Used: A,B,C,D1,R0

Stk lvls: +2

## 9.40 CRFSUB - Create a File in Mainframe

Category: FILUTL File: SC&FIL::MS

Name: CRFSUB - Create a File in Mainframe
Name:(S) CRFSB- - Create a File in Mainframe

Purpose: Create a file in mainframe

ENTRY:
```
        P      = 0
        STMTR0 = FILE NAME
        STMTR1(4) = DEVICE TYPE
        STMTR1(8-7) = PORT #
        STMTR1(15) = FILE COPY CODE FROM FILE TYPE TABLE
        S-R1-0(13-10) = FILE TYPE
        R1  =  ADDRESS OF FILE HEADER ALREADY CREATED BY
               CREATF.  FILE NAME, COPY CODE, AND FILE
               TYPE WILL BE FILLED IN.
```

EXIT: FILE HEADER ALL BEEN PROPERLY FILLED
```
        A      =  FILE CHAIN LENGTH
        C(A)   =  0
        D1     @  PAST THE FILE CHAIN LENGTH FIELD
        D0     @  S-R1-3 (COPY CODE)
        R1     =  ADDRESS OF FILE HEADER
        P      =  0
```

Calls: CREATF, A-MULT

USES:
    Inclusive: A(A),C,D0,D1

Stk lvls: 0

## 9.41 CRTF - Create File in MAIN, PORT, or HPIL

Category: FILUTL    File: SC&FIL::MS

Name:(S) CRTF - Create File in MAIN, PORT, or HPIL

Purpose:
    Create a file of arbitrary type in memory or on an external device.

Entry:
```
A       =  First 8 chars of file name
D(S)    =  FIB device code
D(A)    =  FIB device address:
           D(B)  =  Port# and Extender# for PORT
           D(X)  =  Device address for HPIL device
P       =  0
R0      =  Last two chars of file name if HPIL device
R1(A)   =  File type (high nib = 0)
R2(A)   =  First parameter for create:
```

| Create Code | Format Implied | Meaning of This Parameter |
|---|---|---|
| 0 | Standard | Data length in nibs |
| 1 | DATA | Number of records (can be 0 if not HPIL) |
| 2 | SDATA | Number of records (can be 0 if not HPIL) |
| 4 | Vbl Rec | Number of bytes in file |
| 8 | OEM | Unknown; poll for len |

R2(9:5)= Address of data in RAM/ROM to copy to the newly created file (none if zero)

```
     R3(A)   =  Second parameter for create:
                ----------------------------------------
                Create   Format     Meaning of This
                 Code    Implied       Parameter
                ------   -------    ----------------------
                  0      Standard   (Ignored)
                  1      DATA       Record length in bytes
                                      (256 default)
                  2      SDATA      (Ignored; set to 8)
                  4      Vbl Rec    (Ignored)
                  8      OEM        Unknown; poll for len

Exit:
     P       =  0
     R2(A)   =  File length in nibbles (chain length)
     R3(A)   =  Entry state (updated if default condition)
    Carry set:
     C(A)    =  Error code:
                  "Not Implemented"
    Carry clear:
     D(S)    =  File device code
      (X)    =  Device address
     R1      =  Address of file header if file in memory
     D1      @  Start of data if file is in memory

Calls:       SVFPSC,SVFTYP,POLL,A-MULT,CRETF+,CRFSB-,INITMF

Uses.......
 Exclusive: A,B(S,A),C,D(S),D0,D1,    R1,R2
 Inclusive: A-D,              D0,D1,R0-R4,STMTR0,STMTR1,
            SCRTCH,S11-S0

Stk lvls:   6 - If file created on plug-in, else 5

NOTE:
     This routine can only create BASIC, TEXT, and 41C data
     in memory at the moment.

Algorithm:
     Save away file spec and file type info
     Compute data length from parameters
     Compute and add on subheader length
     If device is not MAIN then
        Error exit for now (not implemented)
     Create file header in memory
     Fill in name, etc.
     Initialize file according to create code

History:

    Date    Programmer                 Modification
```

```
--------  ----------  ---------------------------------
09/24/82  SC          Add code to create and initialize
                      file in HP-IL device.
                      POLL to create OEM file and the
                      POLL handler has to do it all.
07/21/82  NZ          Modified entry condition locations
07/13/82  NZ          Added field (R2(9:5)) for address
                      of data to copy to the file after
                      creation; modified exit code to
                      use D0 instead of D1 to get stated
                      exit conditions; changed R2 exit
                      conditions
06/01/82  FH          Wrote from looking at code for
                      CREATE execute and CRBAS (create
                      BASIC).  Needed for TRANSFORM.
```

## 9.42 OPENF - Open File

Category: FILUTL     File: SC&FIL::MS

```
Name:(S) OPENF   - Open File
Name:    OPENF-  - Open File
Name:    OPENF*  - Open File
Name:    OPNF+   - Open File

Purpose:  Open a new file in the FIB

Entry:
  All:   P   =  0

  OPENF:   D0 points at file spec. in the BASIC statement

  OPENF*:  A, D(S), D(A), R0 set up as on exit from FSPECx
     R2     =  0 if R2/R3 device assignment info not
               present
            =  Device assignment info from FSPECx.
     R3     =  Device assignment info from FSPECx unless
               R2 = 0.

  OPNF+:   D1 points at start of file header in memory
```

```
   OPENF-:  STMTR0 & STMTR1 has the information as the entry
            condition specified by the WRTFIB routin.
     R2      = 0 if R2/R3 device assignment info not
               present
             = Device assignment info from FSPECx.
     R3      = Device assignment info from FSPECx unless
               R2 = 0.

Exit:
            P  =  0
            Carry set => Done successfully
              A(B) = FIB# of file
              R1 = the new entry address in FIB
              S10  =  Set if file has I/O buffer
              STMTD1 = FIB address of file
              STMTR0, STMTR1 set to exit conditions of WRTFIB
              The FIB entry filled with proper information
            Carry clear => Error
              C(3-0) = Error code
              File already opened
              FIB full
              Insufficient memory
              Unrecognized file type

Calls: FSPECx, POLL, FINDF, DATSTR, I/OFND

Uses:
 Inclusive: A,B,C,D,D0,D1,R0,R1,STMTR0,STMTR1,S10

Stk lvls: 6 at least (FSPECx takes 5, pFINDf requires 6)

Note: This routine falls into WRTFIB to write the file
      information in the FIB.
```

## 9.43 WRTFIB - Write File Information to FIB

Category: FILUTL     File: SC&FIL::MS

Name:(S) WRTFIB - Write File Information to FIB

Purpose: Write file information into File Information Buffer

```
Entry:
     D1     @  Entry address of the file in FIB.  FIB# has
                 already been written to the entry
     R2     =  0 if R2/R3 device assignment data are not
               present (relevant only for exernal files)
            =  OTHERWISE, device assignment data from FSPECx
     R3     =  Device assignment data from FSPECx if R2 # 0
     STMTR0(0-10)= File data start address
       If file in RAM/ROM:
           STMTR0(0-4) = Absolute data start address
           STMTR0(5-6) = 0F
           STMTR0(7-10)= Don't care
       If file in port:
           STMTR0(7-8) = PORT #
       If file in HP-IL device:
           STMTR0(0-3) = Record #
           STMTR0(4-10)= HP-IL address
     STMTR0(11-14) = File type
     STMTR0(15) = Device type
       0 - Mainframe
       1 - Independent RAM
       2 - ROM
       8 - HPIL
     STMTR1(0-5) = File start address
       If file in RAM/ROM, this is the absolute
           address of the file headrer.
       If file in HP-IL device, this is the record
           number and byte number of the LIF directory
           entry address of the file.
     STMTR1(6-10) = File length in nibbles if the file
           copy code = 0.
     STMTR1(6-9) = File length in # of records if the
           file copy code = 1.
     STMTR1(10-13) = Record length in bytes if the file
           copy code = 1.

EXIT:
     Never returns if unrecorgnized file type
     R1     =  FIB entry address
     Carry  =  Set if no error

Calls:

Uses:
  Inclusive:  A,B,C,D,D0,D1,R0,R1,R2,R3  S10

Stk lvls:  +5
```

## 9.44 CLOSE# - Close File

Category: FILUTL File: SC&FIL::MS

Name: CLOSE# - Close File
Name:(S) CLOSEF - Close File

Purpose: Close file in File Information Buffer

Entry:
CLOSE#: B(B) = Channel # of the file
CLOSEF: A(B) = FIB # of the file

Exit:
No error condition if the file not found

Calls: FFIB#, POLL

Uses: A,B,C,D0,D1, STMTD1

Stk lvls: 5

NOTE: This program FALLS INTO routine DELFIB

## 9.45 CLOSEA - Close All Open Files

Category: FILUTL File: SC&FIL::MS

Name:(S) CLOSEA - Close All Open Files

Purpose: Close all opening files and delete their entries

Entry: P = 0

Exit: P = 0

Calls: I/OFND, DELFIB, POLL(pWRCBF)

Uses: A-D, D0, D1, R0 ,STMTD1

Stk lvls: 5

## 9.46 FIBON - Reset Devices, Buffers at Power On/Off

Category: FILUTL File: SC&FIL::MS

Name: FIBON - Reset Devices, Buffers at Power On/Off
Name:(S) FIBOFF - Reset Devices, Buffers at Power On/Off

Purpose: When HP-71 powers off, reset all external devices.
When HP-71 powers on, reclaim all the I/O buffers.

Entry: None

Exit: P=0.
Hex mode.

Calls: I/OFND, I/ORES

Uses: A,C, D0,D1, S0

Stk lvls: 2

## 9.47 PUGFIB - Purge the FIB Entries of Purged Files

Category: FILUTL File: SC&FIL::MS

Name:(S) PUGFIB - Purge the FIB Entries of Purged Files

Purpose: Purge the FIB entry of a purged file.
Delete an FIB entry whose "File Begin" address is zero.

Entry: The "file Begin" of the purged file in FIB should be already zeroed.

Exit: The first FIB entries matching the condition is deleted

Calls: FDFILM, DELFIB, I/ODAL

Uses: A-D,D0,D1,R0

Stk lvls: +4

## 9.48 RENSUB - Renumber Subroutine

Category: FILUTL File: SC&REN::MS

Name:(S) RENSUB - Renumber Subroutine

```
Purpose : 1. Compile all line number references
          2. Clear all compiled offsets
          3. Renumber all line number references

Entry : CURRST & CURREN pts current file
        S1 = 0 - Only clear compiled offset
           = 1 - Compile offset or renumber line number
        If S1=1 :
           S2 = 1 - Compile reference offset
           S2 = 0 - Renumber line number

Exit : Carry set => No error
       Carry clear=> Line number not found
                     R2= ptr to stmt len of stmt in error

Calls : PFNDL*, EXPSKP, FINDA, ISRAM?, LINE#1, POLL

Uses : A,B(A),C,D(A),D0,D1,R2, S3

Stk lvls : +2
```

Detail :

The line number is expected to be found in the following mainframe statements:

1. GOTO/GOSUB/RESTORE <LINE#/LABEL>
2. ON ERROR GOTO/GOSUB <LINE#/LABEL>
3. ON TIMER [#<exp>,] <exp> GOTO/GOSUB <LINE#/LABEL>
4. ON <exp> GOTO/GOSUB/RESTORE <LINE#/LABEL>,...
5. IF <exp> THEN LINE#/LABEL/EXT [ELSE LINE#/LABEL/EXT IF]
6. PRINT USING LINE#/LABEL
7. DISP USING LINE#/LABEL
8. ON INTR GOTO/GOSUB LINE#/LABEL
9. POLL for non-mainframe XWORD

For XWORD (External) statements, the line number is handled as follows:

. If RENSUB is just called for zeroing the compiled offset (S1=0), the line # in XWORD statement will be ignored. This means the execution of an XWORD statement has to assume the compiled offset is incorrect and has to zero it everytime.
. If RENSUB is called for renumbering (S1=1), the poll pREN will be issued so that each LEX file that contains XWORD statements that may have line numbers will be allowed to supply the correct renumbering. See the pREN poll interface for details.

## 9.49 EXPSKP - Skip Over Tokenized Expression

Category: FILUTL File: SC&REN::MS

Name:(S) EXPSKP - Skip Over Tokenized Expression

Purpose: Skip over tokenized expression

Entry: D1 = Start of expression

Exit: A = NEXT TOKEN after expression
D1= Points to next token after expression
Carry set

Calls : FINDA

Uses: A(A) ,C(5:0), D1, S10

Stk lvls : 1

## 9.50 FNDFCN - Find User-Defined Function

Category: FILUTL File: SC&SUB::MS

Name:(S) FNDFCN - Find User-Defined Function

Purpose: Find a user-defined function

Entry: R1(X) = Function name(output from ADRSUB)

Exit:
Carry set => Found
D0 past the function name in the DEF FN statement
F-R1-0 = Address past the tDEF of the DEF FN
Carry clear => Not found

Calls: PRSCOP, GETNAM

Uses: A,B,C,D,D1,D0

Stk lvls: 4

## 9.51 KEYMRG - Key Merge

Category: FILUTL File: SG&EXC::MS

Name:(S) KEYMRG - Key Merge

```
Name:    KYMRG+  -  Key Merge

Purpose:    Creates space for new entry in keys file

Entry:   P=     0
         B(A) = HEX Keycode
         2 ENTRY POINTS:
           1) KEYMRG - A(A) =Length of assignment string
           2) KYMRG+ - C(B) =Keycode
                       C(6-2)= Length of assignment string

Exit:    D1 points to start of new entry
         R2(B) = Keycode; R2(3-2) = Entry length
         R2(S) = B(S) on entry
         B(A)  = offset to memory
         R3    = Pointer to keys file header
         Carry clear
         via RSTD1

Calls:   KMEMCK, CREATF, MOVEDM, RFADJ+, KYD30,
         KEYFND, KYPRCK, LAKEYS, UPDFCL

Uses:    A-D, D1, D0, R0-R3, F-R0-1, S6, S8

Stack lvls: 5

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/01/82    S.W.         Added documentation
 11/02/82    S.W.         Added call to UPDFCL
 12/29/82    S.W.         Eliminated call to RFAD85
```

## 9.52 FILXQ^ - Filename Execute

Category: FILUTL    File: SG&FXQ::MS

```
Name:(S) FILXQ^  -  Filename Execute
Name:(S) FILXQ$  -  Filename Execute For a String Expression
```

Purpose:

Executes a tokenized file specifier. Solitary device Specifiers of the form ':CARD', ':PORT', ':MAIN', etc, are accepted. There are two entry points.

FILXQ^:

Assumes that D0 points to a file specifier in program memory. The file specifier may be a literal or a string expression.

FILXQ$:

Assumes that the alleged string has been evaluated and is on the Math Stack. This entry is used by ADDR$ and CAT$.

Entry:

FILXQ^:
D0 at start of file specifier
FILXQ$:
D1 points to string expression on top of Match Stack

Exit:

```
CARRY SET: (both entry points)
  Mainframe-recognizable file specifier found.
  A(W)   =  Blank-padded file name if name present.
         =  0 if only device specifier present, as in
            ':MAIN', ':PORT', ':CARD'
  D(S)   =  F if no device specified
         =  0 if device is :MAIN
         =  1 if device in :PORT, in which case:
              D(0) = PORT extender number 0-F
              D(1) = PORT number          0-4
              D(B) = FF if no PORT number specified
         =  7 if device is card, in which case:
              D(B) = 0 if :CARD
              D(B) = Nonzero if :PCRD
  D0     =  Past file specifier  (FILXQ^ only)
  P      =  0
  If file specifier was a string expression:
   D1 points past the string on the stack
  If file specifier was a literal containing a port#:
   D1 points past the 16 nibble number on the stack
   (AVMEME)=D1

CARRY CLEAR:
 FILXQ^:
  Executed illegal mainframe file name. Either string
  expression or literal name with over 8 characters.
  S7=1 =>      Specifier was string expression, in which
               case the expression is still on the stack.
                AVMEME points to the string header
```

```
                 D0    points past the tokenized expression.

      S7=0 =>     Specifier was a literal;  D0 may be restored
                  to the start of the literal by using (STMTD0).
      P      = 0

     FILXQ$:
      String expression on stack contained an illegal specifier.
      S7     = 1
      AVMEME = Value it contained on entry.  May be used to
               preserve the pointer to the string header prior
               to calling FILXQ$.
      P      = 0

    ERROR EXIT (both entry points):
      Exit to MFERR (eFSPEC) if and only if :PORT is found,
      followed by an illegal port specifier.


Calls:      EXPEXC, FILEP, PDEV, CATCHR, DVCTYP, POLL,
            REVPOP, BLKOK, PRTNP, FINDA, RSTST, SAVED0,
            CNVWUC, AVE=D1
Uses........
  Exclusive:  A-D, D1, D0, STMTD0, R0, R1, S1,S2,S7
  Inclusive:  STMTR1 (all of it) -- port spec. as num expr
              R0-R3, all of function scratch -- EXPEXC

       NOTE:  FILXQ$ entry doesn't use any statement scratch.


Detail:     D0 on entry to FILXQ^ is a pointer to the start
            of the compiled file specification.  FILXQ^ must
            save D0 in STMTD0, since EXPEXC can use all CPU
            registers and all function scratch RAM.  STMTD0
            will be updated if memory moves.

            SYNTAX FOR PORT# IS <d[.d[d]]>

            ASSUMES THAT ALL NON-MAINFRAME DEVICE REFERENCES
            HAVE BEEN TOKENIZED WITH tCOLON.

            Nibs 2,3,4 of D are zeroed out for TRSFMu

Stack lvls: FILXQ$ entry pt - 3
            Otherwise       - 5

History:

 Date       Programmer   Modification
 -------    ----------   ------------
 06/29/82   S.W.         Added documentation.
```

```
07/05/82   S.W.        Modified code to eliminate
                        call to POP1S - lets REV$
                        take care of that.
07/27/82   S.W.        Added code to check for tCOLON
                        before assuming string expr.
10/21/82   S.W.        Save PC in STMTDO, instead
                        of S-R1-0
01/31/83   J.P.        Clear S7 on entry
06/28/83   S.W.        Save rtn stack level in R0
                        prior to calling FILEP!.
```

## 9.53 PDEV - Evaluate Num Expression as Port Device

**Category:** FILUTL **File:** SG&FXQ::MS

```
Name:(S) PDEV   - Evaluate Num Expression as Port Device
Name:    PDEV+  - Evaluate Num Expression as Port Device
Name:    PDEV1  - Evaluate Num Expression as Port Device

Purpose:  Evaluates numeric expression for port address

          PDEV+ and PDEV entries evaluate an expression
          in memory and ensure it is a valid numeric
          expression.

          PDEV1 assumes that the evaluated expression is
          already on the stack.  It is useful for functions.

Entry:    3 entry points:
          1) PDEV+ - D0 2 nibs prior to alleged numeric
                     expression.
          2) PDEV  - D0 at alleged numeric expression.
          3) PDEV1 - D1 points to evaluated expression on
                     math stack.

Exit:     D(0)=Port extender#; D(1)=port#
          D1 points to numeric expression on stack
          D0 past evaluated numeric expression
           (if entered at PDEV1, D0 unchanged from entry)
          Statuses intact
           (except if entered at PDEV1)
```

ERROR EXITS IF EVALUATED EXPRESSION IS INAPPROPRIATE
FOR A PORT ADDRESS.

Calls: EXPEXC, TST12A, FRAC15, FLTDH, ARGSTA
CLRFRC, GTPRTN, RSTST

Uses: A-D, D1,D0, R0-R3, all of function scratch -- EXPEXC

Detail: Allows numeric expressions which evaluate to x.yy
where:
0<= x <= 5 and 0<= yy <= 15

Stack lvls: 5

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/29/82 | S.W. | Added documentation |
| 08/05/82 | S.W. | Added PDEV1 entry point |

## 9.54 FSPECx - File Specification Execute

Category: FILUTL File: SG&FXQ::MS

Name:(S) FSPECx - File Specification Execute

Purpose: Evaluates a file specification

Entry: D0 @ File specification start

Exit : D0 past file specification
Carry Clear: Legal file specification

```
A = filename  (blank filled)
A = 0 if no filename
R0 = last two chars of file name (if any)
   = two blanks by default
D(S) = F  NO DEVICE SPECIFIED
       0  MAIN
       1  PORT    D(B) = PORT number
```

```
                            D(B) = FF if :PORT
                    7  CARD     D(B) = 0 if CARD
                       PCRD     D(B) # 0 if PCRD
                  = 8  HP-IL device  (D(X) = device addr)
                  > 8   other

              P=0
              P reset before POLL
              If file specifier was a string expression:
               (AVMEME) points past the string on the stack


            Carry set:
              Unrecognized File Specification
              C(3-0) = Error#

  Calls:    FILXQ^, POLL

            FILXQ^ returns:
            Carry Clear ---> Illegal File Spec
                             S-R1-0 holds original D0
            Carry Set   ---> Legal File Spec
                             S8=0  Simple Filename
                             S8=1  D(S)=F No Device specified
                                        0 MAIN
                                        1 PORT
                                          D(B) = Port#
                                               = FF if :PORT
                                        7 CARD
                                          PCRD
                                          D(B) = 0 if CARD
                                               # 0 if PCRD

  Uses:     A-D
            D = End of Expression stack  (from FILXQ^)
            STMTD0, STMTR1 (all of it), S1,S2,S7 -- FILXQ
            D1,D0, R0-R3, all of function scratch -- EXPEXC

  Detail: Try Mainframe File Execute                  (FILXQ^)
          Blank-fill lower 2 bytes of R0
          If acceptable file specification  (Carry set)
             If simple filename
                Set Device = 0   (D(S))
             RTNCC
          else
             POLL for File Specification Execute
             Return if Carry Set
             If handled (XM=0)
                Return with Carry Clear
             else
                C <-- eFSPEC
```

Return with Carry Set

Stack lvls: 6

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/29/82 | S.W. | Added documentation |

## 9.55 FINDF - Find a file

Category: FILUTL File: SG&FXQ::MS

Name:(S) FINDF - Find a file
Name:(S) FINDF+ - Find a file
Name: FILEMF - Find a file
Name:(S) FILEF - Find a file
Name: FINDWF - Find a file

Purpose: Searches for a Specified File in file chain(s) specified by the caller.

The entry points which allow the file chains to be specified require as entry conditions some of the exit conditions from FILXQ^/FSPECx.

FILEF and FILEMF entries search the MAIN file chain only.

FINDF and FINDF+ entries look at D(S) to determine which file chains to search. The only difference between the two entry points is that FINDF assumes the integrity of D(S) and A(W), whereas FINDF+ checks their integrity to ensure that A(W) is nonzero and D(S)<=6.

FINDWF searches the MAIN file chain for <workfile>

Entry: P=0
5 entry points:

```
        1) FINDF -  file name in A(W)
                    D(S) determines search pattern:
                       =F => Search MAIN, plug-ins
                       =0 => Search MAIN only
                    other => Search Plug-ins only
                            D(B) indicates port desig.
                               =FF  =>   all PORTS (:PORT)
                            OR D(1)=     PORT #
                               D(0)=     Extender #

        2) FILEF -  File name in A(W) - Mainframe search only

        3) FILEMF   same as above, except file name in B(W)

        4) FINDF+ - Same as FINDF.

        5) FINDWF - Searches for workfile

  Exit:     P=0
            Carry Clear - File found
              D1   @  File Start
              A(W)=B(W) contain file name
              D(S) =  Device Type

                      0 = Mainframe RAM
                      1 = IRAM
                      2 = ROM
                      3 = EEPROM

                 It cannot be assumed that Device Type is
                 limited to these numbers.

                 Routines using FINDF should probably POLL when
                 Device Type is not 0-2.

              D(B) =  Extender#, Port#   (if applicable)

            Carry Set => File not found
              S6=1 =>
                B=A  = Filename
                C(3-0) contains err# for eFnFND or eDVCNF
              S6=0 (FINDF+ entry only) =>
                Illegal file spec for file chain search
                 either A(W)=0 or D(S)>=7
                C(3-0)=eFSPEC
                C(S) = 2^(D(S)+1)

  Calls:    ROMCHK, ROMFND, ROMF-1, FILSKP, C=MAIN, WRKFIL

  Uses:     A-D, D1, S6,S8,R1,R2 (if outside of Main search)
                                 R3 (if single PORT search)
```

```
          S6 = Not Initial PORT search
          S8 = Single/ Special file chain search

          ROMCHK
          ROMFND  uses A-D,D1,R0,R1
          ROMF-1  uses A-D,D1,R0,R1,R3

  Stk lvls: 2

  Detail:

   FINDF+:  If D(S) >= 7 or A(W)=0
              Return with carry set; C(3-0)=eFSPEC
   FINDF:   Clear Single Filechain Search flag  (S8)
            Move filename to B
            If Standard search     D(S)=F
               goto 1;
            If MAINframe only      D(S)=0
               goto FILFMF;
            else                   (PORT)
               Save filename       (R2)
               If all Ports     (D(B)=FF)
                  go Search ALL Ports  (goto 3);
               else
                  Set single file chain flag (S8)
                  Find Start of file chain in Port  (ROMF-1)
                  Restore filename to A
                  Put filename in B
                  Set S6 for error  (file not found)
                  If not found
                     Return Carry   C(3-0)=eDVCNF
                  else
                     Continue search (goto 2)
    FILEF:   B <-- Filename
   FILFMF:   Set Single Filechain flag       (S8)
       1:    Set pointer @ Main memory start
             Clear Initial Port Search flag (S6)
       2:    Read filename
             If not at end of file chain     (A(B)#0)
                If filename match --> RTNCC
                else
                   Skip to next file
                   goto 2;
             else  (End of file chain)
                Restore file name to A
                If single search only          (S8)
                   RTNC   C(3-0)=eFnFND
                else
                   If initial PORT search       (S6=0)
                      Save filename             (R2)
                      Set Not Initial PORT search  (S6)
```

```
3:              If no ROMs exist          (ROMCHK)
                   Restore filename
                   RTNC   C(3-0)=eFnFND
                else
                   Set B = filename
                   go search ROM for file   (goto 2)
             else
                Find next ROM               (ROMFND)
                Restore filename
                If no more ROMs  ---> RTNC C(3-0)=eFnFND
                Set B = filename
                go Search ROM for file      (goto 2);
```

Note:
Device ID's 2-6 are NOT available for use.
Dedicated devices are restricted to ID's 9-E

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/29/82 | S.W. | Added Documentation |
| 10/29/82 | S.W. | Modified entry conditions for new device codes |
| 12/20/82 | S.W. | Calls FILSKP instead of RDHDR so FINDF doesn't use S9 |

## 9.56 PRGFMF - Purge File in Memory

Category: FILUTL    File: SG&FXQ::MS

Name:(S) PRGFMF - Purge File in Memory

Purpose:
Purges specified file

Entry: 2 entry points:
1) PRGFMF - D(S) as it is after FINDF call
   D1 pointing to start of file header
2) PRGF - File in MAIN; S11=0.
   D1 at file type in file header.

```
Exit:
     Carry set => error# loaded in C(3-0)
                  Caller should exit using BSERR
     Carry clr => File purged successfully
                  S7=1 => Purged current running file

Calls:      POLL, RAMROM, GETPRO, EOFLCH, CREATF, FINDWF
            LEXBF+, ZERPGM, MEMCKL, PUGFIB, FILSKP,
            RFA-I, D1=CRS, RSTOFS, MOVEUM, EDIT81

Uses.......
 Exclusive: A-D, D0,D1,R0,R1,S-R0-0,S-R0-1, S7,S9-S11
   If purging current file, also uses R2 & R3, S6,S8, S-R0-0
   If purging a LEX file, also use R2,R3
   If purging current file AND there's no workfile, uses S0-S7

Stk lvls:   5

   Date      Programmer   Modifications
 --------    ----------   -------------
 08/04/82    S.W.         Added documentation
 12/16/82    S.W.         Replaced calls to RSTK=R and
                           R=RSTK with R<RSTK and RSTK<R
                           C(S) now used
 06/06/83    S.W.         Replaced call to CLSUSP with
                           a call to ZERPGM. (Poll must
                           go out when curr file purged)
```

## 9.57 EDIT - Moves EDIT Pointers to Specified File

Category: FILUTL     File: SG&FXQ::MS

```
Name:      EDIT    - Moves EDIT Pointers to Specified File
Name:(S)   EDITWF  - Designates workfile as Current File
Name:(S)   EDIT80  - Designates Specified File as Current
Name:      EDIT20  - Collapses Stks; Spec. File Becomes Curr.

Purpose:
          EDIT executes the EDIT statement.

          EDITWF designates the workfile as the current file.
```

```
          If it doesn't exist, it is created.  EDITWF is
          called when current file is purged and during
          configuration.

          EDIT80 designates the specified file as current.
          If file isn't BASIC, a POLL goes out, resulting
          in an error if no one responds.  This entry
          point is used by CAT when [f][EDIT] is hit
          during a multiple file catalog.

          EDIT20 collapses all the execution stacks before
          designating the specified file as current.  This
          is the entry point used by RUN.  An assumption is
          made that this file is of legal type to be made
          current.

Entry:    4 entry points:
          P=0
            1) EDIT   - D0 past tEDIT.

            2) EDITWF - S10=1 => No collapse of stacks and
                                 no CAtalog.
                        S10=0 => No collapse of stacks
                                 CAtalog iff S11=0

            3) EDIT80 - S10 and S11 as with EDITWF.

            4) EDIT20 - D1 points at new current file.

Exit:
          CURRL UPDATED; Stacks, etc collapsed via CLPSTK
            Error Exits if:
              1) file must be created and not enough memory
              2) specified file is not BASIC
              3) port# specified that doesn't exist
              4) non-mainframe device specified
          If no CAtalog is done:
            B(A)=CURRST; C(A)=D(A)=CURREN;
            D0 points to CURREN RAM location

Calls:    CRETF, CLPSTK, FINDF, SAVEL, WRKFIL
          EOLXCK, FSPECx, POLL, NULLP, BASKEY

Uses:     A-D, R0-R3, S6, S8, S9, S10, S11, D1,D0
          + If FSPECx is called: S1,S2,S7, STMTD0,
                STMTR1 (All of it), All of function scratch

Detail:   EDIT is a system command (non-programmable).  The
          reason for this limitation is that EDIT changes
          CURRST & CURREN; this would be nonsensical during a
          running program, since the same pointers are used
```

to indicate current EDIT file as current RUN file.

EDIT [filename]

Stack lvls: 7

History:

| Date | Programmer | Modifications |
|---|---|---|
| 06/30/82 | S.W. | Added documentation |
| 07/20/82 | S.W. | No longer saves 2 stack levels (burden put on PRGFMF) |
| 09/17/82 | J.P. | Set S9 before NULLP call |
| 11/11/82 | S.W. | Deleted poll on external file |
| 12/17/82 | S.W. | Eliminated call to CHAIN - caused problems when old EDIT file is in non-RAM medium; ptr to new CURRST no longer in R3 on exit. |
| 01/11/83 | J.P. | Change S9=1 to P=1 before NULLP call. |
| 03/02/83 | J.P. | Added pEDIT poll |

## 9.58 RAMROM - Classify Memory Device

Category: FILUTL    File: SG&FXQ::MS

Name:(S) RAMROM - Classify Memory Device

Purpose: Returns info on whether file in RAM,IRAM,other

Entry: D(S) preserved from FINDF call:
=0 => Mainframe RAM
=1 => IRAM
=2 => ROM
=3 => EEPROM

Exit: CARRY SET => RAM
S8=1 => IN MAIN
0 => IRAM

```
                    CLR => non-RAM memory device
                           S8=0
```

Calls: none

Uses: S8, C(S)

Stack lvls: 0

History:

| Date | Programmer | Modifications |
|---|---|---|
| 06/30/82 | S.W. | Added documentation |
| 12/17/82 | S.W. | Eliminated distinction between ROM & other non-RAM memory devices |

## 9.59 LOCADR - Locate, Classify Address's Memory Device

Category: FILUTL File: SG&FXQ::MS

Name:(S) LOCADR - Locate, Classify Address's Memory Device
Name:(S) CURDVC - Classify Current File's Device

Purpose: Given a file address, returns information regarding the medium (MAIN,IRAM,ROM, etc.)

CURDVC entry assumes the file address is (CURRST).

Entry: 2 entry points:
1) CURDVC - No additional requirements.
2) LOCADR - C(A) = some address in the file

```
Exit:   Specified address in R2
        Carry clr => Legitimate address
                     D(S)=0 => MAIN
                         NO => PORT
                               D(S) reflects memory type
                                   =1 => RAM
                                   =2 => ROM
```

```
                                      =3 => EEPROM
                                 D(0)= Extender#
                                 D(1)= Port#
                                 D(7-2)=Rest of Config. entry
                        A(A)=D1=R2(A)

        Carry set (LOCADR entry only) =>
                        Not a legitimate address
```

Calls: ROMCHK, ROMFND, EOFLCH, D1=CRS

Stk lvls: 2

Uses: A-D, D1, R1 & R2

Detail: THE ADDRESS MUST BE WITHIN A FILE CHAIN, OR CARRY WILL AUTOMATICALLY COME BACK SET.

History:

| Date | Programmer | Modifications |
|---|---|---|
| 06/30/82 | S.W. | Added documentation |

## 9.60 GETPRO - Get File Protection of Current File

Category: FILUTL File: SG&SYS::MS

Name:(S) GETPRO - Get File Protection of Current File
Name: GETPR+ - Get File Protection of Specified File
Name: GETPR - Get File Protection of Specified File
Name:(S) GETPR1 - Get File Protection of Specified File

Purpose: Returns file protection information

GETPRO reads file protection of the current file.

All other entry points read the file proection nibble of the file specified by the caller.

```
Entry:      4 ENTRY POINTS:
            P= 0
            1) GETPRO - (CURRST) is accurate
            2) GETPR+ - (D1) = pointer to file header.
            3) GETPR  - A(A) = pointer to file header.
            4) GETPR1 - D1   = pointer to file header.

Exit:       SB=1 . . . IFF SECURE
            CARRY SET  IFF PRIVATE
            D1 POINTS AT FILE TYPE FIELD
            P=0
            C(3-0)= eFPROT

Calls:      none

Uses:
     exclusive...  C, D1, SB
     inclusive...  A(A), C, D1, SB  (GETPRO, GETPR+ only)

Stack lvls: 0

History:

  Date      Programmer   Modification
 --------   ----------   ------------
 06/28/82   S.W.         Added Documentation
 10/13/82   S.W.         C(B)=eFPROT on exit
 11/23/82   S.W.         C(3-0) as above
```

## 9.61 FILSKP - File Skip

Category: FILUTL     File: SG&SYS::MS

```
Name:     FILSKP  -  File Skip
Name:     FLSKPB  -  File Skip
Name:(S)  FILSK+  -  File Skip

Purpose:
     Skips over specified file

Entry:
```

P=0
3 entry points:
1) FLSKPB - B(A) at file header start
2) FILSKP - C(A) at file header start
3) FILSK+ - A(A) at file header start

Exit:
P=0
C(A)= Points to next file in chain (OR to 00 BYTE)
A(A)= Length in file's file length field
D1 = Points to file length field
Carry clear

Calls: none

Uses A(A), C(A), D1

Stk lvls: 0

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/05/82 | S.W. | Added documentation |
| 10/21/82 | S.W. | Changed entry conditions |

## 9.62 FILFIL - Fill in Missing File Name

Category: FILUTL File: TI&UTL::MS

Name:(S) FILFIL - Fill in Missing File Name

Purpose:
Adjusts file spec info on Save Stack to fill in missing file name if necessary. If the destination file name is null, it always receives the source file name. If source file name is null, it receives desination file name unless source device is CARD or PCRD, or if high bit of the device info is set. Status is returned indicating if one file spec (or both) is external, and if both file names are undefined.

```
Entry:
     P        =  0
     File specs on Save Stack as per SVINFO.
     Upper nib of device info on stack has upper bit set if
     source file name fill is NOT to be done for this file
     spec.

Exit:
     P        =  0
     Updated file specs on Save stack as per SVINFO, with
       the no-fill flag cleared for each file spec
     S(sEXTDV) =  Set if either or both file specs are
                  on HPIL device.
     S(sUNDEF) =  Set if both file names are zero
                  (that is, undefined).
     S(sCARD)  =  1 if Source or Dest Device = CARD|PCRD
     S(sDEST)  =  0 ("Source")
     A         =  First 8 chars of source file name
     R0(3-0)   =  Last 2 chars of source file name
     D(A)      =  Source device info from RDINFO
     R2(A)     =  Dest device info from RDINFO
     Carry     =  Clear

Calls:       RDINFS, RDINFD, SVINFO, MFDEVC, MFDVC-

Uses.......
 Inclusive:  A,B,C,D(A),D1,R0,R1,R2,S4-S0

Stk lvls:    2

Detail:      Module Flow:
             ------------
             Clear Status
             Read Source info, check device type and save away
             Read Dest info
             If Source file is undefined and device not card
                Source file name <-- Dest file name
             Check Dest device type
             If Dest file name is undefined
                and neither device is CARD | PCRD
                  Dest file name <-- Source file name
             Write back Dest file info
             Recall Source file info
             Check Source device type
             Write back Source file info

History:

  Date      Programmer              Modification
--------    ----------    -----------------------------------
05/15/82       FH         Designed and coded.
```

```
02/15/83      FH       Added check for "No fill" bit of
                         device code
```

## 9.63 FLADDR - Find First/Last Address of Mem Device

Category: FILUTL     File: TI&UTL::MS

```
Name:(S) FLADDR  -  Find First/Last Address of Mem Device

Purpose:
     Find the first and last address of available memory on
     the specified memory device (PORT or MAIN).

Entry:
     D(S)    =  Device type code of memory device (MAIN = 0,
                  IRAM = 1, ROM = 2, etc)
     D(0)    =  Port number if PORT device
     D(1)    =  Extender number if PORT device
     D(7-2)  =  Nibs 8-3 on configuration table entry for
                port device (contains size, address)
     P       =  0

Exit:
     A(A)    =  Address of first nib available memory on
                device
     C(A)    =  Address of last nib available memory on
                device
     D       =  Entry state
     D1      @  AVMEMS for MAIN device
             =  Size of module if PORT device
     P       =  0
     Carry clear

Calls:       EOFLC+, LSTADR

Uses.......
 Exclusive: A(A),C,D1
 Inclusive: A,   C,D1

Stk lvls:   2
```

Algorithm:
  If PORT then
    Start of module plus offset to file chain
    Skip to end of file chain
    Space beyond chain to av mem start
    Find last address (call LSTADR)
  Else (it's MAIN)
    Fetch AVMEMS, AVMEME

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/11/82 | FH | Designed and coded |

## 9.64 RPLLIN - Replace Line in Memory File

Category: FILUTL    File: TI&UTL::MS

Name:(S) RPLLIN - Replace Line in Memory File

Purpose:
  Replace a line in a memory file with the contents of the output buffer. May be used to insert, delete, or replace a line in the file.

Entry:
  OUTBS @ Start of replacement line
  AVMEMS @ End of replacement line (address of last nib + 1)
  A(A) = Address of last nib + 1 of old line
  C(A) = Address of file header of file
  R3(A) = Length of OLD line in nibs (zero for insertion)
  P = 0

Exit:
  R3(A) = Offset of move (DEST END - SOURCE END)
  P = 0
  Carry clear: [Successful replacement]
  Output buffer collapsed

```
      A(A)    =  End + 1 of replaced line in file
      B(A)    =  Length of replacement line in nibs
      C(A)    =  (OUTBS)
     Carry set:
      C(3-0) =  Error code:
                  eMEM   - Insufficient memory
                  eILACS - Illegal access (if ROM or PROM)
```

Calls: DBLCMP, MOVE*M, MVMEM+, INITPT

```
Uses.......
 Exclusive: A,B(A),C,                 D1,R0,R1,   R3
 Inclusive: A,B   ,C,D(S),D(7-0),D0,D1,R0,R1,R2,R3
```

Stk lvls: 3

NOTE:
Security and privacy are not checked. ROM or EPROM access returns eFACCS error.

Algorithm:

History:

| Date | Programmer | Modification |
|---|---|---|
| | FH | Adapted from a TRANSFORM utility |
| 02/15/83 | FH | Packed and updated documentation |

```
+-------------------------------------------+-------------------+
|                                           |                   |
|   FNEXEC  -  Function Execute             |   CHAPTER  10     |
|                                           |                   |
+-------------------------------------------+-------------------+
```

## 10.1 TRMNTR - Process Terminator In Expr Execute

Category: FNEXEC    File: AB&EXP::MS

Name:(S) TRMNTR - Process Terminator In Expr Execute

Purpose:
Process terminator in expression execute. Collapse expression execution environment and return to whomever called EXPEXC.

Entry:
D1 = mathstack pointer.

Exit:
D1 = mathstack pointer.
A[W] = 16 nibbles at top of stack.

Calls: None.

Uses.......
A,C[A].

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |

## 10.2 GDISP$ - GDISP$ function execution

Category: FNEXEC File: SB&GPM::MS

Name:(S) GDISP$ - GDISP$ function execution

Purpose:
Implements GDISP$ function

Entry:
P = 0
D0 is program counter
D1 is stack pointer

Exit:
Exits through EXPR

Calls: CPYDD-

Algorithm:
Save D0 on stack
Calculate where stack item will start
If not enough memory then
  Exit with "Insufficient Memory" error
Write out header for 132 character string
Copy rightmost display driver (DD) to string
Copy middle DD to string
Copy leftmost DD to string
Point stack pointer to new string
Restore D0 from stack
Exit through EXPR

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/26/83 | B.S. | Added documentation |

## 10.3 KEY$ - KEY$ function

Category: FNEXEC File: SG&KEY::MS

Name:(S) KEY$ - KEY$ function

Purpose:
Evaluates KEY$ function

Entry:
P = 0

Exit:
P = 0
via ADHEAD

Calls: D=AVMS, POPBUF, KEYNAM, STKCHR

Uses: A-C,D(A), R0-R2, S0-S2, D1,D0

Stk lvls: 3

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/29/83 | S.W. | Added documentation header |

## 10.4 CAT$ - CATalog Function

Category: FNEXEC File: SG&SYS::MS

Name: CAT$ - CATalog Function
Name:(S) CAT$20 - Build CATalog Information Buffer

Purpose: CAT$ function returns CATalog information on a specific file.

The CAT$20 entry point is used to build a buffer of CATalog information. It is used by CAT and CAT$ for the card reader, and the mainframe.

Entry: 2 ENTRY POINTS:
1) CAT$ - Entry for execution of CAT$
2) CAT$20 - Entry for CAT. S0 must be clear to flag that the buffer shouldn't be pushed on the stack. D1 at file header start.

Exit: BUFFER POINTED TO BY CONTENTS OF 'OUTBS'

Calls: OUTNBS, FLTDH, GETRG+, LOCADR, SAVDO, RSTDO, FILXQ$, POLL, FTYPDC, PRTNDC, LDCSET, ROMF-1, CAT$70, CAT$80, BLNKC+, AVS=DO, OBCOLL, GETPRO, FILSKP, BF2STK, DOOUTB, D1=AVE, D1=CRS, C=MAIN, AVE=D1

Uses: A-D, D1,DO, S0, R1,R2 -- CAT$20 entry point
Inclusive: All the above + F-RO-O, AVMEME, R3, S7-S11

Detail: FILE LENGTH < 1,048,576 NIBS (DECIMAL)

IF numer expr <=0 AND no 2nd parm, then defaults to current file.
REGARDLESS OF ANY SPECIFIED STRING EXPRESSION.

If called by CAT, then after return AVMEMS should be set to OUTBS via OBCOLL

Stack lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/28/82 | S.W. | Increased documentation |
| 08/05/82 | S.W. | Added code to swap date & time, and to add port# |
| 10/21/82 | S.W. | Calls to AVS=DO & OBCOLL |
| 06/10/83 | S.W. | Replaced calls to LDCSET & BLANKC with call to BLNKC+ |
| 06/28/83 | S.W. | Port# saved in R3 (not on RSTK) before calling GETRG+ |

| | |
|---|---|
| GENUTL - General Purpose Utilities | CHAPTER 11 |

## 11.1 STKCMD - Pushes Statement On Command STACK

Category: GENUTL File: AB&CLC::MS

Name:(S) STKCMD - Pushes Statement On Command STACK

Purpose:
Pushes statement on command stack.

Entry:
P = 0

Exit:
P = 0

Calls: ORGN10, STREQL, MOVEU3

Uses....... A, B, C, D, P, D0, D1

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/09/83 | SA | Added documentation |

## 11.2 D=WORD - Read 8 Bytes And Convert To Uppercase

Category: GENUTL File: AB&LEX::MS

Name:(S) D=WORD - Read 8 Bytes And Convert To Uppercase

Purpose:
Read 8 bytes from memory and convert to uppercase.

Entry:
D0 pointing at text to be read.

Exit:
P=0.
D[W] contains uppercase version of text.

Calls: None.

Uses.......
C,D,P.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | MM | Attempted to document |

## 11.3 RANGE - Verify A Byte Is In Certain Range

Category: GENUTL File: AB&UTL::MS

Name:(S) RANGE - Verify A Byte Is In Certain Range
Name:(S) DRANGE - Verify A Byte Is In Range "0"-"9"

Purpose:
    Determine if a byte is in a specified range.
    Caller supplies range for RANGE.
    This code supplies range of "0" to "9" for DRANGE.

Entry:
    P=0.
    A[B] = byte to be checked.
  RANGE: C[B] = lower bound of range to check,
         C[3-2] = upper bound of range to check.

Exit:
    P=0.
    Carry clear if byte in range.

Calls: None.

Uses.......
    C[A].

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 11.4 MEMBER - Check If Byte Is A Member Of A Set

Category: GENUTL    File: AB&UTL::MS

Name:(S) MEMBER - Check If Byte Is A Member Of A Set

Purpose:
    Determine if a byte is a member of a set of bytes.

Entry:
    C=set of bytes (C[1-0], C[3-2], etc.).
    P points to hinibble of upper byte of set.

A[B] = byte to be checked.

Exit:
P=0.
Carry clear if byte in set.

Calls: None.

Uses.......
C[WP] (whatever P was on entry), P.

Stk lvls: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 11.5 STUFF - Fill Memory With Stuff Or 0's

Category: GENUTL File: AB&UTL::MS

Name:(S) STUFF - Fill Memory With Stuff Or 0's
Name:(S) WIPOUT - Fill Memory With Stuff Or 0's

Purpose:
Fill up memory will a pre-determined 16-nibble pattern (STUFF) or with zeroes (WIPOUT).

Entry:
HEX mode.
D1 @ start of area to be stuffed.
C[A] = length of area to be stuffed (in nibs).
STUFF: A[W] = pattern to be stuffed into memory.
(WIPOUT presets A[W] to 0).

Exit:
P=0.
Carry clear.

D1 pointing past last nibble stuffed.

Calls: None.

Uses.......
P,C,D1. WIPOUT: A.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 11.6 MOVEDM - Blk Move To Higher Addr

Category: GENUTL File: AB&UTL::MS

Name:(S) MOVEDM - Blk Move To Higher Addr
Name:(S) MOVEDO - Blk Move To Higher Addr
Name:(S) MOVEDA - Blk Move To Higher Addr
Name:(S) MOVED1 - Blk Move To Higher Addr
Name:(S) MOVED2 - Blk Move To Higher Addr
Name:(S) MOVED3 - Blk Move To Higher Addr
Name:(S) MOVEDD - Blk Move To Higher Addr

Purpose:
Block move of memory to higher address.

Entry:
```
MOVEDM: A[A] @ end of destination
        B[A]   = block length
        C[A] @ end of source

MOVEDO: DO @ end of source
        D1 @ end of destination
        B[A]   = block length

MOVEDA: =AVMEME @ start of source
```

```
            D1 @ end of destination
            A[A] @ end of source

   MOVED1:  D0 @ pointer to start of source
            D1 @ end of destination
            A[A] @ end of source

   MOVED2:  D1 @ end of destination
            A[A] @ end of source
            C[A] @ start of source

   MOVEDD:  A[A] @ end of source
            D1 @ end of destination
            C[A] = block length

   MOVED3:  D0 @ end of source
            D1 @ end of destination
            C[A]   = block length
```

Exit:
P=0.
D0 @ start of source.
D1 @ start of destination.

Calls: None.

Uses.......
A,C[A],D0,D1,P.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 11.7 MOVEUM - Blk Move To Lower Addr

Category: GENUTL File: AB&UTL::MS

Name:(S) MOVEUM - Blk Move To Lower Addr
Name:(S) MOVEU0 - Blk Move To Lower Addr
Name:(S) MOVEUA - Blk Move To Lower Addr
Name:(S) MOVEU1 - Blk Move To Lower Addr
Name:(S) MOVEU2 - Blk Move To Lower Addr
Name:(S) MOVEU3 - Blk Move To Lower Addr
Name:(S) MOVEU4 - Blk Move To Lower Addr

Purpose:
Move a block of memory to a lower address.

Entry:

```
MOVEUM: A[A] @ start of destination
        B[A]   = block length
        C[A] @ start of source

MOVEU0: D0 @ start of source
        D1 @ start of destination
        B[A]   = block length

MOVEUA: =AVMEMS @ end of source
        D1 @ start of destination
        A[A] @ start of source

MOVEU1: D0 @ pointer to end of source
        D1 @ start of destination
        A[A] @ start of source

MOVEU2: D1 @ start of destination
        A[A] @ start of source
        C[A] @ end of source

MOVEU3: D0 @ start of source
        D1 @ start of destination
        C[A]   = block length

MOVEU4: A[A] @ start of source
        D1 @ start of destination
        C[A] = block length
```

Exit:
P=0.
D0 @ end of source.
D1 @ end of destination.

Calls: None.

Uses.......
A,C[A],D0,D1,P.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 11.8 STRTST - Test Strings For Equality

Category: GENUTL File: AB&UTL::MS

Name:(S) STRTST - Test Strings For Equality
Name:(S) STREQL - Test Strings For Equality

Purpose:
Test two strings for equality.

Entry:
STRTST:
D0 and D1 at high-memory end of the two strings to be compared.
C[A] = block comparison length (in nibbles).
STREQL:
D0 and D1 at high-memory end of the two strings to be compared.
B[A] = (block comparison length - 1)/16.
P = (block comparison length - 1) mod 16.

Exit:
If comparison length = 0, carry clear and XM=1.
If strings equal, carry clear and XM=0.
If strings not equal, carry set and XM=0.
P can be anything.
B[A] contains remnant of length/16.
A, C contains first words not equal.
D0 and D1 point at first words not equal.

Calls: None.

Uses.......
A,B[A],C,P,D0,D1.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 11.9 CSRC1 - Perform 1 CSRC

Category: GENUTL File: AB&UTL::MS

Name:(S) CSRC1 - Perform 1 CSRC
Name:(S) CSRC2 - Perform 2 CSRCs
Name:(S) CSRC3 - Perform 3 CSRCs
Name:(S) CSRC4 - Perform 4 CSRCs
Name:(S) CSRC5 - Perform 5 CSRCs
Name:(S) CSRC6 - Perform 6 CSRCs
Name:(S) CSRC7 - Perform 7 CSRCs
Name:(S) CSRC8 - Perform 8 CSRCs
Name:(S) CSRC9 - Perform 9 CSRCs
Name:(S) CSRC10 - Perform 10 CSRCs
Name:(S) CSRC11 - Perform 11 CSRCs
Name:(S) CSRC12 - Perform 12 CSRCs
Name:(S) CSRC13 - Perform 13 CSRCs
Name:(S) CSRC14 - Perform 14 CSRCs
Name:(S) CSRC15 - Perform 15 CSRCs
Name:(S) CSLC1 - Perform 1 CSLC
Name:(S) CSLC2 - Perform 2 CSLCs
Name:(S) CSLC3 - Perform 3 CSLCs
Name:(S) CSLC4 - Perform 4 CSLCs
Name:(S) CSLC5 - Perform 5 CSLCs
Name:(S) CSLC6 - Perform 6 CSLCs
Name:(S) CSLC7 - Perform 7 CSLCs
Name:(S) CSLC8 - Perform 8 CSLCs
Name:(S) CSLC9 - Perform 9 CSLCs
Name:(S) CSLC10 - Perform 10 CSLCs
Name:(S) CSLC11 - Perform 11 CSLCs
Name:(S) CSLC12 - Perform 12 CSLCs
Name:(S) CSLC13 - Perform 13 CSLCs

Name:(S) CSLC14 - Perform 14 CSLCs
Name:(S) CSLC15 - Perform 15 CSLCs

Purpose:
Perform 1 to 15 circular left or right shifts to C.

Entry:
None.

Exit:
C-register shifted.

Calls: None.

Uses.......
C.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 11.10 OUT1TK - Output 1 byte from A(B)

Category: GENUTL File: JP&PR2::MS

Name:(S) OUT1TK - Output 1 byte from A(B)
Name:(S) OUT1T+ - Increment D1, Output 1 byte from A(B)
Name:(S) OUTBYT - Output 1 byte from C(B)
Name:(S) OUTBY+ - Increment D1, Output 1 byte from C(B)
Name:(S) OUT2TK - Output 2 bytes from A(3-0)
Name:(S) OUT2TC - Output 2 bytes from C(3-0)
Name:(S) OUT3TK - Output 3 bytes from A(5-0)
Name:(S) OUT3TC - Output 3 bytes from C(5-0)
Name:(S) OUTNIB - Output 1 nibble from C(0)

Purpose:

Output specified number of nibbles to address pointed to by D0; a check is made so that D0 does not write past available memory end.

Entry:

```
D(A) = (AVMEME) - Available Memory End
D0   = address at which output to go

OUTNIB:  Nibble to output in C(0)
OUT1TK:  Byte to be output in A(B)
OUT1T+:  Byte to be output in A(B)
OUTBYT:  Byte to be output in C(B)
OUTBY+:  Byte to be output in C(B)
OUT2TK:  2 Bytes to be output in A(3-0)
OUT2TC:  2 Bytes to be output in C(3-0)
OUT3TK:  3 bytes to be output in A(5-0)
OUT3TC:  3 bytes to be output in C(5-0)
```

Exit:

```
No memory error =>
  Carry clear on exit
  D0 incremented past the tokens that were output
  D1 incremented by 2 (OUT1T+, OUTBY+ entries only)
  A(B) & C(B) are swapped (OUTBYT, OUTBY+ entry)
  A(A) & C(A) are swapped (OUT2TC entry only)
  A(W) & C(W) are swapped (OUT3TC entry only)

Else
  golong MEMERR
```

```
Calls:     OVFLCK

Uses:      D0                  (OUTNIB,OUT1TK,OUT2TK,OUT3TK)
           D1,D0                      (OUT1T+)
           A(B),C(B), D0              (OUTBYT)
           A(B),C(B), D1,D0           (OUTBY+)
           A(A),C(A), D0              (OUT2TC)
           A,C, D0                    (OUT3TC)

Stk lvls:  1
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/07/82 | JP | Modified documentation |
| 11/02/83 | S.W. | Modified documentation header. |

## 11.11 D1C=R3 - Restore C(A),D1 from R3

Category: GENUTL    File: JP&PR2::MS

Name:(S) D1C=R3 - Restore C(A),D1 from R3

Purpose:
    Restores D1 from R3(5-9)
    Reverse effect of R3=D1C

Entry:
    None

Exit:
    C(A) = R3(A)
    A(A) = R3(5-9)
    D1 = R3(5-9)
    Carry preserved from entry

Calls: None

Uses.......
 Exclusive: A,C(A),D1
 Inclusive: A,C(A),D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/07/82 | JP | Modified documentation |

## 11.12 R3=D10 - Save D0 and D1 in R3

Category: GENUTL    File: JP&PR3::MS

```
Name:(S) R3=D10  -  Save D0 and D1 in R3
Name:    R3=D1C  -  Save C(A) & D1 in R3
Name:    R3=D1+  -  Save C(A) & A(A) in R3

Purpose:
     R3=D10 entry saves D0 in R3(A) and D1 in R3(9-5).
     R3=D1C entry saves C(A) in R3(A) and D1 in R3(9-5).
     R3=D1+ entry saves C(A) in R3(A) and A(A) in R3(9-5).

Entry:
     R3=D10:  D0 and D1 contain values to save in
              R3(A) and R3(9-5), respectively.
     R3=D1C:  C(A) and D1 contain values to save in
              R3(A) and R3(9-5), respectively.
     R3=D1+:  C(A) and A(A) contain values to save in
              R3(A) and R3(9-5), respectively.

Exit:
     Carry preserved from entry
     A(A)=C(A)
     R3=D10: R3(A)=D0 on entry;   R3(9-5)=D1 on entry
             C(A)=A(A)=D0
     R3=D1C: R3(A)=C(A) on entry; R3(9-5)=D1 on entry
     R3=D1+: R3(A)=C(A) on entry; R3(9-5)=A(A) on entry

Calls:      None

Uses.......
     R3=D10: A, C(A), R3
     R3=D1C: A, R3
     R3=D1+: A, R3

Stk lvls:   0

History:

  Date     Programmer              Modification
 --------  ----------   -----------------------------------
 07/06/82  JP           Modified documentation
```

## 11.13 CSL9RO - Copy D1 to RO(9-5)

Category: GENUTL File: MB&IMG::MS

Name:(S) CSL9RO - Copy D1 to RO(9-5)

Purpose:
Copy D1 to RO(9-5) without disturbing the rest of RO.

Entry:
No necessary conditions.

Exit:
P = 0
Carry clear

Calls: CSLWP9

Uses.......
Exclusive: A,C(A)
Inclusive: A,C(A),P

Stk lvls: 1

Detail:
```
=CSL9RO A=RO
        CD1EX
        D1=C
        GOSBVL =CSLWP9
        C=A    A
        RO=C
        RTN
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 11.14 IMDO+2 - Add 2 to R1(A), copy value to DO

Category: GENUTL File: MB&IMG::MS

Name:(S) IMDO+2 - Add 2 to R1(A), copy value to DO
Name:(S) IMDO-2 - Subtract 2 from R1(A)

Purpose:
IMDO+2: Take DO storage in R1, increment by 2 and copy to DO.
IMDO-2: Subtract 2 from R2(A).

Entry:
No necessary conditions.

Exit:
Carry clear.
IMDO+2: R1(A) incremented by 2.
DO=C(A)=R1(A)
IMDO-2: R1(A) decremented by 2.

Calls: none

Uses.......
Exclusive:
IMDO+2: C(W), DO
IMDO-2: nothing

Stk lvls: 0

Detail:
```
=IMDO-2 CR1EX
        C=C-1  A
        C=C-1  A
        CR1EX
        RTNCC
=IMDO+2 C=R2
        C=C+1  A
        C=C+1  A
        R1=C
        DO=C
        RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|

| -------- | ---------- | ---------------------------------- |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 11.15 D12ROA - Copy D1 to RO(A)

Category: GENUTL     File: MB&IMG::MS

Name:(S) D12ROA - Copy D1 to RO(A)

Purpose:
To copy D1 to RO(A) without disturbing the rest of RO.

Entry:
No necessary conditions.

Exit:
Carry clear.

Calls: none

Uses.......
Exclusive: RO(A)

Stk lvls: 0

Detail:

```
=D12ROA CROEX
        CD1EX
        D1=C
        CROEX
        RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 11.16 NwOFFS - Recover old offset, store new one in RAM

Category: GENUTL     File: MB&USG::MS

```
Name:(S) NwOFFS  -  Recover old offset, store new one in RAM

Purpose:
     Recover old offset from AvMemEnd, store a new one
     in the same location.  (Utility for IMAGE execution,
     but can be used anywhere.)

Entry:
     D1=address+5 for which new offset will be computed
     Old offset resides at AvMemEnd

Exit:
     Carry clear
     New offset stored in AvMemEnd
     C(A)=recovered offset from AvMemEnd (recovered means
       that the addition has been performed on the offset
       to recover the address)
     D1=A(A)=AvMemEnd+5

Calls:       StAVE+ (SetAVE), CA2D1+

Uses.......
 Exclusive: A(A),C(A),D1
 Inclusive: A(A),C(A),D1

Stk lvls:    1

Detail:
    =NwOFFS D1=D1- 5
            AD1EX
            GOSBVL =SetAVE       Set D1=C=AvMemEnd
            A=A-C  A             Compute new offset.
            C=DAT1 A             Fetch old offset.
            DAT1=A A             Store new offset.
            GOTO   CA2D1+        Recover compute address.

History:

  Date     Programmer               Modification
 --------  ----------  -----------------------------------
 12/08/82  MB          Documentation
```

## 11.17 RCVOFS - Recover offset from RAM storage

Category: GENUTIL File: MB&USG::MS

```
Name:(S) RCVOFS  -  Recover offset from RAM storage
Name:(S) C+A2D1  -  Recover offset from RAM storage
Name:    CA2D1+  -  Recompute offset from RAM storage

Purpose:
     To recover a 5-nibble offset from RAM (recover means
     to fetch the offset, perform addition to recompute
     the original address).

Entry:
     RCVOFS: offset to recover resides at D1-5
     C+A2D1: offset to recover resides at D1

Exit:
     Carry clear
     D1=A(A)=address+5 where offset was found
     C(A)=recovered offset (offset was added to D1 to
       recompute old address)

Calls:       none

Uses.......
 Exclusive:  A(A),C(A)
             C+A2D1 also uses D1 (does a D1+5)

Stk lvls:    0

Detail:
    =RCVOFS D1=D1- 5
    =C+A2D1 C=DAT1 A
     CA2D1+ D1=D1+ 5
            AD1EX
            D1=A
            C=A+C  A
            RTNCC

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 11.18 BP - Machine-level Beep

Category: GENUTL File: MN&BP::MS

```
Name:    BP     - Machine-level Beep
Name:    BP+    - Machine-level Beep
Name:(S) BP+C   - Machine-level Beep
Name:(S) TONE   - Machine-level Beep

Purpose:
     Perform BEEP.

Entry:
     BP: A = frequency in hz (floating point dec).
         C = duration in secs (floating point dec).

     BP+: A[A] = duration in msec (hex).
          D[A] = frequency in hz (hex).
          HEX mode.

     BP+C: C[A] = duration in msec (hex).
           D[A] = frequency in hz (hex).
           HEX mode.

     TONE: C[X] = inner loop countdown constant.
           B[W] = outer loop countdown constant (# cycles).
           HEX mode.
           (Bypasses check of beep flag, computation of
             constants based on freq, duration and
             clockspeed.)


Exit:
     HEX mode.

Calls:      BP: RJUST, DCHXW, all BP+ calls.
            BP+: CSLW5, CSRW5, IDIV, MPY, SFLAG?.
```

Uses.......
A,B,C,D,D0,P.

Stk lvls: 2

Detail:
Maximum duration is 1048.575 seconds (FFFFF msec).
Maximum frequency is determined by clockspeed. At 500 khz clockspeed, maximum frequency is 6757 hz.

Algorithm:
```
Define: f = frequency
        t = duration in msec
        k1 = inner loop countdown constant
        k2 = outer loop countdown constant
One beep cycle (one cycle of square wave) takes
32*k1+74 machine cycles.  The routine beeps for k2 beep
cycles.
  k1=(clkspd/f-74)/32
  if k1<0 then k1=0
  if k1>FFF then k1=FFF
  f'=clkspd/(32*k1+74)   {compute actual frequency}
  k2=f*t/1000    {compute cycle count}
  Execute tone loop, using k1 to time square waves
    and k2 to count tone cycles.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/20/82 | NM | Added documentation |

## 11.19 CHIRP - Do An Annoying Little Beep

Category: GENUTL    File: MN&BP::MS

Name:(S) CHIRP - Do An Annoying Little Beep

Purpose:
Quick, high-pitched beep for errors and whatever.

Entry:
    HEX mode.

Exit:
    HEX mode.

Calls: BP+C (falls through).

Uses.......
    A,B,C,D,P,D0.

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/02/83 | MM | Added documentation |

## 11.20 ROMCHK - Find ROM / File Chain Start

Category: GENUTL File: MN&CNF::MS

Name: ROMCHK - Find ROM / File Chain Start
Name:(S) ROMFND - Find ROM / File Chain Start

Purpose:
    Check if ROMs exist
    Find file chain start within ROM/IRAM
    Return Device Information about ROM

Entry:
    ROMCHK: First time entry point
      Finds ROM Configuration Table
      If non-empty, save pointers required for entry to
        ROMFND.

    ROMFND: Repeated entry point
      R1(X) = Length to end of Configuration Table
      R1(3-7)=Position within Configuration Table

```
Exit:
     P=0
     ROMCHK:
       Carry set:
         Empty Configuration Table
       Carry Clear:
         D1,C(A) @ First file on plug-in
         D(S) = Device type
                1 = IRAM
                2 = ROM
                3 = HP EEPROM
                4 = Intel EEPROM
                Device type is incremented by 1
                to distinquish from RAM
         D(0) = Port Extender #   (Device #)
         D(1) = Port #
         D(2-7) = Nibs (3-8) of config table entry
         R1(X) = Length to end of Configuration Table
         R1(3-7)=Position within Configuration Table

         R1 must be preserved between calls to ROMFND

     ROMFND:
       Carry set:
         No more ROMs
       Carry Clear
         Same Exit Conditions as ROMCHK

Calls:      CNFFND

Uses.......
 Exclusive: A-D,D1,R1
 Inclusive: A-D,D1,R1

Stk lvls:   1

NOTE:
     R1 must be preserved between calls to ROMFND

Algorithm:

ROMCHK:  Find ROM Configuration Table (CNFFND)
         If no table entries ---> RTNC
         Move to Device # field in table
         Move Table length to B
     1:
         Read Device#, Port# and Size infor into C,D
         Read 3 High nib address & Device type
         Adjust pointer (D1) to next entry in table
         Increment & Move Device type to D(S)
         Calculate & Read first file address
```

```
             Save Len of Config table & Next entry pos'n
             in R1.
             D1 <-- Start of file
             RTNCC
  ROMFND:  Restore Len of Config Table  (Low 3 nibs R1 -> B)
           Restore Position in Config Table  (R1 --> D1)
       2:  If entries left (B>0)
              goto 1;
           else
              RTNSC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/09/82 | JP | Modified documentation |

## 11.21 ASRW3 - Shift A Right 3 Nibbles

Category: GENUTL     File: MN&UTL::MS

```
Name:(S) ASRW3   -  Shift A Right 3 Nibbles
Name:(S) ASRW4   -  Shift A Right 4 Nibbles
Name:(S) ASRW5   -  Shift A Right 5 Nibbles
Name:(S) ASLW3   -  Shift A Left  3 Nibbles
Name:(S) ASLW4   -  Shift A Left  4 Nibbles
Name:(S) ASLW5   -  Shift A Left  5 Nibbles
Name:(S) CSRW3   -  Shift C Right 3 Nibbles
Name:(S) CSRW4   -  Shift C Right 4 Nibbles
Name:(S) CSRW5   -  Shift C Right 5 Nibbles
Name:(S) CSLW3   -  Shift C Left  3 Nibbles
Name:(S) CSLW4   -  Shift C Left  4 Nibbles
Name:(S) CSLW5   -  Shift C Left  5 Nibbles
```

Purpose:
(SL or SR) (A or C) (3, 4 or 5) times.

Entry:
Yes.

Exit:

xSdWn: Register x shifted direction d n times.
Carry and pointer unaffected.

Calls: None.

Uses.......
Register x (above, Exit conditions).

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/23/82 | NM | Added documentation |

## 11.22 SFLAGS - Sets system flag

Category: GENUTL File: PM&FLG::MS

Name:(S) SFLAGS - Sets system flag

Purpose:
Sets a system flag and updates annunciators

Entry:
C(B) -- hex flag number (e.g. load FF for -1)
HEXMODE
P=0

Exit:
specified flag set
any corresponding annunciator turned on
Carry=Clear
D(A) - Set to D0
HEXMODE
P=0

Calls: GTFLAG,UPDANX

Uses.......

```
 Inclusive: CPU: A(A),B(A),C(15,5-0),D(A),P
            RAM: ANNAD1-4,SYSFLG

Stk lvls:   2

History:

  Date     Programmer            Modification
 --------  ----------  -----------------------------------
 06/11/82      PM       Documented routine
 04/11/83      PM       Revised documentation
```

## 11.23 SFLAGC - Clears system flag

Category: GENUTL    File: PM&FLG::MS

```
Name:(S) SFLAGC  -  Clears system flag

Purpose:
     Cleares a system flag and updates annunciators

Entry:
     C(B) -- hex flag number (e.g. load FF for -1)
     HEXMODE
     P=0

Exit:
     specified flag cleared
     any corresponding annunciator turned on
     Carry=Clear
     D(A) - Set to DO
     HEXMODE
     P=0

Calls:      GTFLAG,UPDANX

Uses.......
 Inclusive: CPU: A(A),B(A),C(15,5-0),D(A),P
            RAM: ANNAD1-4,SYSFLG

Stk lvls:   2
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/11/82 | PM | Documented routine |
| 04/11/83 | PM | Revised documentation |

## 11.24 SFLAGT - Toggles system flag

Category: GENUTL File: PM&FLG::MS

```
Name:(S) SFLAGT  -  Toggles system flag

Purpose:
     Toggles a system flag and updates annunciators

Entry:
     C(B) -- hex flag number (e.g. load FF for -1)
     HEXMODE
     P=0

Exit:
     specified flag toggled
     any corresponding annunciator turned on
     Carry=Set if flag previously set
     Carry=Clear if flag previously cleared
     D(A) - Set to D0
     HEXMODE
     P=0

Calls:       GTFLAG,SYSFLC,UPDANN

Uses.......
 Inclusive: CPU: A(A),B(A),C(15,5-0),D(A),P
            RAM: ANNAD1-4

Stk lvls:    3

History:

   Date     Programmer              Modification
```

| -------- | ---------- | ---------------------------------- |
|---|---|---|
| 06/11/82 | PM | Documented routine |
| 04/11/83 | PM | Revised documentation |

## 11.25 SFLAG? - Tests system flag

Category: GENUTL File: PM&FLG::MS

Name:(S) SFLAG? - Tests system flag

Purpose:
Tests a system flag

Entry:
C(B) -- hex flag number (e.g. load FF for -1)
HEXMODE
P=0

Exit:
Carry=Set if flag set
Carry=Clear if flag clear
D(A) - Set to DO
HEXMODE
P=0

Calls: GTFLAG

Uses.......
Inclusive: A(A),C(15,5-0),D(A)

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/11/82 | PM | Documented routine |
| 04/11/83 | PM | Revised documentation |

## 11.26 GTFLAG - Gets RAM nib and flag mask

Category: GENUTL File: PM&FLG::MS

Name:(S) GTFLAG - Gets RAM nib and flag mask

Purpose:
Gets nibble and mask for SYSTEM flag specified
by hex flag #

Entry:
C(B) -- hex flag number
HEXMODE
P=0

Exit:
A(XS) - appropriate nibble from flag register
C(XS) - mask: 1 bit on at position of flag
D(A) -- previous content of D0
D0 ---- points at appropriate nibble in flag register
carry=clear
P=0
HEXMODE

Calls: nothing

Uses.......
Inclusive: A(A),C(15,5-0),D(A),D0

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/14/82 | PM | Documented routine |
| 12/17/82 | PM | Removed conversion ovfl. tests |
| 04/11/83 | PM | Revised documentation |

## 11.27 FINDA - Look For A(B) In A Table And Jump

Category: GENUTL File: SB&DSP::MS

Name:(S) FINDA - Look For A(B) In A Table And Jump
Name:(S) FINDD0 - Look For (D0) In A Table And Jump

Purpose:
Searches a table following GOSUB for a byte matching A[B] and jumps to address specified for that value.

Entry:
FINDA:
A(B)=byte to be found
FINDD0:
(D0)=byte to be found
Table of bytes and address offsets must follow GOSUB
The call should look as follows:

```
GOSBVL =FINDA       <---GOSUB is followed by table
CON(2) \Q\          <---Byte to be matched
REL(3) ESCQ         <---Where to jump if matched
CON(2) \R\
REL(3) ESCR
CON(2) \C\
REL(3) ESCC
.
.
.
CON(2) 0            <---Null byte terminates table
                    <---Followed by code to execute
                        if no match is found
```

Entry points:
1) FNDD0+ - Increments D0 1 byte, then reads in A(B)
2) FINDD0 - Reads in A(B) from D0
3) FINDA - Assumes byte to compare already in A(B)

Exit:
P = 0

Calls: None

Uses.......
Inclusive: C(A)

Stk lvls: 0

Detail:
This routine uses 3 nibble self-relative offsets

Algorithm:
Pops address off return stack and uses that address as the start of a table of alternating byte to be compared and 3-nibble relative offsets of where to jump if that byte matchs what is in A(B). The last entry in the table should be a 0 byte followed by the code to execute if no match is found.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/13/82 | B.S. | Wrote routine to replace BYTSCN |
| 09/14/82 | B.S. | Changed to fall thru to otherwise code |

## 11.28 TBLJMP - Indexed table jump

Category: GENUTL File: SB&DSP::MS

Name:(S) TBLJMP - Indexed table jump
Name:(S) TBLJMC - Indexed table jump

Purpose:
Performs an indexed table jump into a table of 3-nibble relative offets following GOSUB.

Entry:
Table of relative offsets must follow GOSUB
TBLJMP: P = index of table to jump to
TBLJMC: C(0) = index of table to jump to

Exit:
P = 0

Calls: None

Uses.......

Inclusive: C(A)

Stk lvls: 0

Detail:
Pops address off stack and adds 3 times the index to it. It then uses REL3DO to jump to the address specified by that table entry.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/14/82 | B.S. | Created routine to replace CASE. |

## 11.29 INTRPT - Interrupt Handler

Category: GENUTL File: SB&DVR::MS

Name: INTRPT - Interrupt Handler
Name:(S) INTR50 - Reentry point for ext. interrupt handler

Purpose:
INTRPT:
Processes interrupts whenever they happen
INTR50:
Reentry point for external interrupt handlers
Restores CPU registers for interrupt RAM then returns from interrupt.

Entry:
None

Exit:
R4(A)=DO at time of call. No other registers changed.

Calls: KEYSCN

Uses.......
Exclusive: R4(A),RAM(INTR4,INTA,INTB,INTM)

Stk lvls: 0

Detail:
Uses 56 nibbles of reserved RAM to save state of machine. Assumes that the subroutine stack has at least one (out of 8) levels available to save the return address.
This routine is not permitted to alter any hardware status bits or the D register since they are not saved or restored.

```
  R4(A) saves C register
  INTR4 saves R4(15-5) and D0
  INTA  saves A register
  INTB  saves B register
  INTM  saves Mode,P,Carry,RSTK[N+1]
```

Algorithm:

```
     Save C(W) in R4
     Save R4(5-15) and D0 in INTR4
     Save A(W) in INTA
     Save B(W) in INTB
     Save 1 stack level, Pointer, Carry, and Mode in INTM
     If this is not a module pulled interrupt then
            goto INTR20
     MP=0
     If MP still active then
            goto MPI
     Set flMPI
   INTR20:
     If Interrupt Ignore Flag is set
            then clear it and goto RESTORE
     If CMOS test word is invalid
            then  Call WARMST and goto RESTORE if it returns
     If VECTOR is non-zero
            then jump to that address
     Wait 8/512ths second to debounce keyboard
     Call KEYSCN
   RESTORE:
     Restore Mode, Carry, Pointer and 1 Stack level
     Restore B(W)
     Restore A(W)
     Restore D0
     Restore C and R4
     Return from interrupt
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | B.S. | Updated documentation |

## 11.30 ATNCLR - Clear Attention Flags

Category: GENUTL File: SB&DVR::MS

Name:(S) ATNCLR - Clear Attention Flags

Purpose:
Clears ATNFLG to inhibit effects of ATTN key. Also returns old state of ATTN flag.

Entry:

Exit:
Carry clear iff ATNFLG was set.

Calls: None

Uses.......
Inclusive: A[A],D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/10/82 | NM | Added documentation |
| 07/25/83 | B.S. | No longer clears Except status bit |

## 11.31 DSLEEP - Deep sleep

Category: GENUTL File: SB&DVR::MS

Name:(S) DSLEEP - Deep sleep

Purpose:
Put TITAN into a power-off state.

Entry:
None.

Exit:
P=0.
Carry clear.

Calls: ALMSRV, ATNCL1, BF2DSP, FIBOFF, I/ODAL, LOCKD?, OUT=1, PWCONF, RCLSTA, FPOLL, NOKEYS, SFLAG?, SFLAGC, SFLAGS, ACBAT?

Uses.......
All CPU registers. SCRTCH in RAM.

Stk lvls: 5

NOTE:
This is how you put the machine to sleep. If memory configuration changes while the machine is asleep, the soft-configured module which called DSLEEP may have moved. Thus when DSLEEP tries to return, the machine will go out to lunch. It is RECOMMENDED that you call DSLEEP through the MGOSUB utility:

```
  GOSBVL =MGOSUB
  CON(5) =DSLEEP
```

Then if configuration changes, the GOSUB stack will be collapsed and the attempt to return from DSLEEP will give a SYSTEM ERROR. This beats going out to lunch.

Secondary local entry point DPS010 is used by PWROFF.

Detail:
Performs power-down poll on entry and one or two power-up polls on wakeup. Control is returned to the calling routine in the following circumstances:

If ATTN key was not hit:
An on-timer alarm is pending with program running
or
A poll handler cleared =flTNOF on =pDSWNK poll.

If ATTN key was hit:
A poll handler cleared =flTNOF on =pDSWKY poll.
or
Password is null
or
User supplies correct password.

LOCK is implemented with the aid of the =flTNOF and =flMKOF flags. Proper manipulation thereof will keep the user from breaking into a locked machine. Guidelines for their use are found in the poll interface descriptions below.

Some special things happen for the benefit of the PWROFF routine, since PWROFF returns control to the main loop upon wakeup. See PWROFF documentation for more detail, including explanation of =bECOMD.

```
Algorithm:
 DSLEEP: Clear =flPWDN flag (indicate that we were not
           called from PWROFF).
 DPS010: (Entry point for PWROFF).
         If ON key down
            Set ATTN flag and goto DSP040
         If display-clear flag clear then goto DPS030
            Send <cursor on>/CR/LF.
 DPS030  Send <cursor off>
 DPS035: Perform power-down poll.
         Set TURNOFF (flTNOF) flag.
         Clear MAKEOFF (flMKOF) flag.
         Turn off display.
         Clear f-g shift status bits.
         Clear ATNFLG and ATNDIS.
         Turn off timer _#3 (Low battery check).
         Activate KB row with ATTN key.
         SHUTDN.

 DPS040: Configure.
         Deallocate external command buffer (to give poll
           handlers a chance to create one if we were
           called by PWROFF).
         Check clock system
         If ATTN key woke us up, goto DPS200.
         If program running and ON TIMER pending
            Clear =flTNOF; goto DPS200.
         Perform pDSWNK poll (who woke us up?!?).
         If turnoff flag set and ATNFLG clear then
            goto DSP035
 DPS200: Flush key buffer.
         Clear flALRM flag.
         =pDSWKY poll
         Password processing (does not require password if
           password=null or =flTNOF is clear).
         If failed to unlock machine (password required but
           not correctly given), goto DPS035.
         AC/BAT check
         RETURN
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | NM | Added name to documentation |
| 09/07/82 | NM | Added calls to AC/BAT at end |
| 09/09/82 | NM | Moved puroff poll after DSP020 |
| 09/13/82 | NM | Made CR/LF conditional on clear flg |
| 09/20/82 | NM | Check ON key at DPS010 |
| 09/23/82 | NM | Clear flALRM before pDSWKY poll |
| 10/25/83 | B.S. | Updated documentation |

## 11.32 SLEEP - Scan KB, do LSLEEP if key buffer empty

Category: GENUTL    File: SB&DVR::MS

Name:(S) SLEEP - Scan KB, do LSLEEP if key buffer empty
Name:(S) LSLEEP - Light Sleep

Purpose:
SLEEP:
Debounces keyboard and shuts CPU down unless keys are in buffer or down.
LSLEEP:
Shuts CPU down (enters low power state) until some activity on the bus or the keyboard wakes up CPU.

Entry:

Exit:
P = 0
Carry clear if keys in buffer
Carry set if no keys were in buffer

Calls: DEBNCE,KEY?

Uses.......
Exclusive: C(A)
Inclusive: A(W),B(W),C(W),D0

Stk lvls: 1

Algorithm:
```
    Debounce for 8/512ths second then scan keyboard
    If key buffer not empty
        then return with carry clear
    If any keys are down
        then return with carry set
    Shut down CPU
    If MP=1 or flMPI set then
      Go to MPI
    Return with carry set
```

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/15/82 | B.S. | Updated documentation |

## 11.33 CKSREQ - Handle service requests

Category: GENUTL File: SB&DVR::MS

Name:(S) CKSREQ - Handle service requests

Purpose:

Handle service requests. This routine recognizes several possible sources of service requests:
1) Timer 1--Display code needs service.
2) Timer 2--Clock system needs service.
3) Timer 3--Battery check code needs service.

After examining above, CKSREQ performs a poll which allows:
1) Handling of SREQs we don't recognize.
2) Handling related to recognized SREQs (e.g., scheduling a new external alarm through clock system).

This code is typically called when:
1) We wake up from a sleep state (delay, etc.).
2) We recognize that an SREQ is exerted at certain points in the mainframe (e.g., interpreter loop).

```
Entry:
     Hex Mode

Exit:
     Hex Mode

Calls:      ACBTSR,CKTMOU,DSPUPD,ALMSRV,PUTPND,FPOLL

Uses.......
            A,B,C,D,P,D0,D1,32 nibs at SCRTCH

Stk lvls:   4

NOTE:
     This code saves the status bits in the user-status
     save area used by the display code.

Algorithm:
     Set BAT annunciator if low battery
     Save caller's status bits in display status area
     If display timer has timed out
       then update display (blink cursor, etc.)
     Check alarm clock system
     Clear external alarm bit in clock system status
     If Except bit set or service request still pending then
       Poll (pSREQ)
     Restore caller's status
     Return

History:

  Date      Programmer             Modification
--------    ----------    ----------------------------------
02/25/83    NM            Added documentation
10/25/83    B.S.          Updated documentation
```

## 11.34 QUOTCK - Quote and Apostrophe Check

Category: GENUTL     File: SB&EXD::MS

Name:(S) QUOTCK - Quote and Apostrophe Check

Purpose:
    Checks if A(B) is a quote or an apostrophe

Entry:
    P     = 0
    A(B)  = Byte to be checked

Exit:
    P     = 0
    Carry set iff A(B) is a quote or an apostrophe

Calls:    None

Uses.......
 Inclusive: C(B)

Stk lvls:  0

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/19/82 | B.S. | Added documentation |

## 11.35 MFLG=0 - Clear MLFFLG nibble

Category: GENUTL    File: SC&DAT::MS

Name:(S) MFLG=0 - Clear MLFFLG nibble
Name:    MFLG=X - Set MLFFLG nibble

Purpose:
    MFLG=0: Clear MLFFLG nibble
    MFLG=X: Set MLFFLG nibble

Entry:
    MFLG=X: C(P) is value to be stored at MLFFLG

Exit:
    MFLG=0: C(A)=0
    (MLFFLG) = Specified value

```
Calls:      None

Uses.......
 Inclusive: D1,(MLFFLG),  and MFLG=0 entry also uses C(A)

Stk lvls:   0

History:

  Date      Programmer              Modification
 --------   ----------   -----------------------------------
 11/06/83   BS           Added documentation
```

## 11.36 PSHSTK - Push Stack

Category: GENUTL    File: SG&EXC::MS

```
Name:(S) PSHSTK  -  Push Stack
Name:(S) PSHSTL  -  Push Stack

Purpose:    Moves high memory to lower memory to allow 'push'
            onto GOSUB, VARIABLE, or some other stack.

            Push address on stack with NO LEEWAY check

Entry:
           D0     pointer to top of stack pointer
           B(A)=  Amt memory needs to 'open up'.
           PSHSTK:
             P=n-1  where n=# pointers to be adjusted
             LEEWAY will ALWAYS be checked
           PSHSTL:
             C(0) = # pointers to be adjusted
             P= non-zero if LEEWAY not to be checked

Exit:     Carry Clear:
             B(A) is preserved
             P=0
             D1 points to new top of stack
             RAM pointers are adjusted
          Error Exit
```

Insufficient Memory to open stack

Calls: MOVEU1, PTRAD1, MEMCL+

Uses: A, C(A), D(A), D0, D1

Detail: Usefulness of this routine could be extended to variable creation, CALL/SUB. etc

GOSUB required C(S) not be altered.

Preserves math stack.

Stack lvls: 1

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/04/82 | S.W. | Added documentation |
| 08/10/82 | S.W. | Modified to preserve math stk |
| 09/30/82 | J.P. | Added MEMCL+ call, removed R1 |
| 10/12/82 | S.W. | Changed D=C B to A field. Replaced MEMCL+ with KMEMCK. |
| 10/29/82 | S.W. | Took out KMEMCK call, due to subroutine levels - PSHSTK to be used by GOSUB/GOSUB |
| 02/15/83 | J.P. | Added PSHSTL entry for no LEEWAY check |

## 11.37 PSHGSB - Push address on GOSUB Stk

Category: GENUTL File: SG&EXC::MS

Name:(S) PSHGSB - Push address on GOSUB Stk
Name:(S) PSHUPD - Push address on GOSUB Stk
Name:(S) PSHMCR - Push address on GOSUB Stk

Purpose:
Push address and return type nibble on GOSUB stack
Allows address to be updated when memory moves

Allows microcode GOSUB/RETURN to work

Entry:
    A(A) = Address to push on stack
    PSHMCR: Sets return type for Microcode return
    PSHUPD: Sets up as Update address. P must be 0.
    PSHGSB: C(S) = Return type (see GOSUB)

Exit:
    Carry Clear:
      P = 0 (not necessary for PSHGSB)
      D1 @ Return type nibble on stack
      C[0] = Return type
      C[5-1]=Address just pushed on stack
    Error Exit:
      Insufficient Memory to open stack

Calls: PSHSTK

Uses.......
 Exclusive: C(W),D(S),P,D
 Inclusive: A-D,D0,D1

Stk lvls: 3

Algorithm:

PSHMCR: C(S) <-- Microcode Return type
PSHUPD: C(S) <-- Update Address Return type
PSHGSB: Save Return Type D(S) <-- C(S)
        Save Return address on stack
        Open up GOSUB stack by 6 nibbles (PSHSTK)
        Restore address and return type
        Write return type and address to stack
        RTNCC

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/30/82 | J.P. | Added code |

## 11.38 POPSTK - Pop Stack

Category: GENUTL File: SG&EXC::MS

```
Name:(S) POPSTK  -  Pop Stack
Name:    POPGSB  -  Pop Stack
Name:(S) POPUPD  -  Pop Stack

Purpose:  Deletes stack entry(ies) and adjusts pointers
        --pertains to FOR/NEXT, GOSUB, etc.
          POPGSB/POPUPD:
          Pop return address/update address off GOSUB stack
        --Reads Return Address and Return type, then deletes

Entry:
 POPGSB:  Sets C(A) and A(A) to top entry of GOSUB stack
          Reads Return type and Return address into D
          Sets P for PTRADJ
 POPSTK:  C(A) points to start of entry to delete (pop)
          A(A) points to end of entry to delete
          P set for PTRADJ

Exit:
 POPSTK: CARRY CLEAR, P=0.
 POPGSB/POPUPD: If Carry set
                   Stack was empty, P unchanged
                Else carry clear, P=0
                  D(A) = Return address
                  D(S) = Return type (see RETURN)

                  If the address on the stack points into a
                  file and that file is purged before the
                  address is popped off, the return address
                  will be ZERO.

                  This can happen if Expression Execute is
                  called, and a multi-line user defined
                  issues a PURGE.

                  Calling routines may need to check for this.

          via PTRAD1

Calls:    MOVED3, RTNSTK

Uses:     A, B(0-5), C, D1, D0
```

If POPGSB/POPUPD uses D(W)
B(S) must be preserved for POLL

**Detail:** Could also be useful to variable elimination (e.g. DESTROY) or to eliminating SUB environments
Must immediately precede PTRAD1

If the return address on the stack points into a file and that file is purged before the address is is popped off, this address will be ZERO.

This can happen if Expression Execute is called and a user defined function issues a PURGE.
A calling routine may have to check this is EXPEXC can be called in the interim.

**Stack Lvls:** 1

**History:**

| Date | Programmer | Modifications |
|---|---|---|
| 07/04/82 | S.W. | Added documentation |
| 08/10/82 | S.W. | Modified to preserve math stk |
| 10/06/82 | J.P. | Added POPGSB/POPUPD entries |
| 10/07/82 | NM | Added stack-empty check |
| 02/10/83 | J.P. | Use only B(0-5) to pres B(S) |

## 11.39 RELJMP - Relative Jump From (D1)

Category: GENUTL    File: SG&LDC::MS

**Name:(S) RELJMP - Relative Jump From (D1)**

**Purpose:**
RELJMP reads the address pointed to by D1, adds it to D1, then does a direct jump to the resulting address.

The mainframe uses RELJMP to jump to a decompile routine.

Entry:
D1 points to relative address

Exit:
D1 = R1 on entry
A(5-0) = 6 nibbles pointed to by D1
C=A
PC is at resulting address

Calls: none

Uses: A,C,D1

Stk lvls: 0

Detail:
When the mainframe uses RELJMP to decompile a statement, on entry D1 points to the decompile address and R1 contains the pointer into the token stream, ie R1 points past the begin BASIC token. So on exit from RELJMP (upon entry to the decompile routine), D1 points past the begin BASIC token and A contains the first six tokenized nibbles that follow.

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/08/83 | S.W. | Added documentation header |

## 11.40 EOLXCK - End of Stmt check

Category: GENUTIL File: SG&LDC::MS

Name:(S) EOLXCK - End of Stmt check
Name:(S) EOLDC - End of Stmt check

Purpose: Checks for statement terminator in the form of t! or t@ or tEOL

Entry: P=0

```
            2 entry points:
              1) EOLDC  -  D1 at token in question
              2) EOLXCK -  A(B) contains token

Exit:     CARRY CLR=> No end of statement token found

Calls:      none

Stack lvls: 0

Uses:     C(B)

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 07/07/82    S.W.         Improved documentation
 07/28/82    S.W.         Eliminated ELSE check
```

## 11.41 OUTNBS - Output nibbles

Category: GENUTIL    File: SG&LDC::MS

```
Name:(S) OUTNBS  -  Output nibbles
Name:(S) OUTNBC  -  Output nibbles
Name:(S) OUTC15  -  Output nibbles

Purpose:    Outputs specified number of nibbles from A or
            C to RAM pointed to by D0

Entry:      D(A) points to AVMEME
            D0 positioned properly
            3 entry points:
             1) OUTNBS - P set for WP write
                         Source in A
             2) OUTNBC - same as above except source in C
             3) OUTC15 - Outputs entire word from C

Exit:       P=0, Carry clear, D0 updated, D(A) preserved

Calls:      none
```

Stack lvls: 0

Uses: C, A (all entry points except OUTNBS), P, D0

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/08/82 | S.W. | Improved documentation |
| 10/18/82 | S.W. | Deleted OUTNC+, OUTNB+ entry points; added OUTC15 |

## 11.42 MFWRN - Warning/message driver

Category: GENUTL File: TI&ERD::MS

Name:(S) MFWRN - Warning/message driver
Name:(S) MFWRNQ - Warning/message driver
Name:(S) MFWRQ8 - Warning/message driver

Purpose:
Display warnings and messages from standard message tables.

Entry:

```
(1)-----------------
 |
 |  P= 1xxx   Sound Beep
 |
 |  P= x1xx   Do not store ERRN
 |
 |  P= xx1x   Display message only (Else display
 |              "WRN:" or "WRN L:" prefix, too)
 |
 |  P= xxx1   Display message without setting delay.
 |


(2)-----------------
 |
```

```
| C(3-2)= LEX ID# (hex)  (mainframe ID# = 00)
| C(B)= message ID number (hex)
|

(3)-----------------
| If desired message has text insertion points:
|  R2 register: source of text insertion.
|  C(14):  type of insertion.
|  C(13):  how many characters in insertion.
|
|   R2
|   -----
|     = actual output characters if C(14)= 1xxx
|     = address of output characters if C(14)= 0xxx
|     = additionally, if C(14)= 0000, upper byte
|          of R2 contains control nibbles.
|
|   C(14)
|   -----
|    1xxx   use contents of R2 register as output
|    0xxx   use address in R2 register to find output
|
|    x000   Output is already in ASCII form
|
|        Digit output (digits can be Hex or Dec):
|    x001   Digit output-- replace leading 0's w/blanks
|    x010   Digit output-- don't suppress leading 0's
|    x011   Digit output-- suppress leading 0's
|
|
|        Hex-to-Dec conversions always generate
|          decimal numbers with 7 digits:
|    x100   Hex-to-Dec: suppress up to 3 leading 0's
|    x101   Hex-to-Dec: suppress up to 4 leading 0's
|    x110   Hex-to-Dec: suppress up to 5 leading 0's
|    x111   Hex-to-Dec: suppress up to 6 leading 0's
|
|
|   C(13)
|   -----
|     For C(14)= 1000 ("ASCII output is in R2")
|         C(13)= #nibbles-1 to be output. Hence the
|                #nibs MUST be even!!; C(13) odd. E.g.,
|                if 5 chars for output, C(13)=9.
|
|     For C(14)= x0xx (hex or dec digit output)
|         C(13)= #digits-1 to be output, hence
|                no more than 16.
|
|     For C(14)= x1xx (hex-to-dec conversion)
```

```
|           C(13)= #digits-1 in number to be converted
|                  Max hex value for conversion is FFFFF
|                  (1048575 dec), hence C(13) must be 4
|                  or less.
|
|       For C(14)= 0000 ("ASCII output from DAT1")
|           C(13)= 0: no output
|                  1: Send out specified number of
|                     character; R2(15-14)= #chars-1.
|                  2: Send out chars until ASCII termin-
|                     ator is found.  ASCII terminator
|                     is passed in R2(15-14) (usually
|                     an FF terminator, but any byte
|                     value can be used).
|
```

Entry for MFWRQ8:
    Same as for MFWRNQ, except that P will be set explicitly to 8. Processing then falls into MFWRNQ.

Exit:
    P = 0
    Carry set

Calls: POLL, SFlag?, KILLKY, fCALC?, CRLFND, UPDCRL, SFlagC, TBMSID, DOASCI, TBMSTX, A=CUR, AVS=C, AVS2DS, CHIRP, XDELAY, CRLFSD, BLDDSP, MFLG=X, R<RST2, RST2<R

Uses.......
 Exclusive: A(W), B(W), C(W), D(W), P, D0, D1, R0
            R2 (only if text insertion; otherwise not used)
 Inclusive: Same

Stk lvls: 2

NOTE:
    If the message constant is eMEM (18 hex), the message routines will automatically invoke MEMERR, and issue an Insufficient Memory error.

Detail:
  Example of text insertion:
    Message #88 in the mainframe is TFM WRN L{5}:{6}, where {5} indicates an insertion point for a line number, and {6} indicates an indirect reference to another message. If we wanted to display
        TFM WRN L145:Syntax      (Syntax is msg #4Bhex)
    we could pass the line number in R2 with the appropriate control codes in C (x=don't care):

```
        R2= xxxxxxx004Bx0145
                        0145= dec digits for output
                   004B= indirect message number

        C=  xB3xxxxxxxxx0088
                        0088= desired warning message
             3=#digits-1 to be output
            B=1xxx: use contents of R2
              x011: digit output, suppress leading O's

    Or, alternatively,
        R2= xxxxxxx004Baaaaa
                       aaaaa= address to find digits
                   004B= indirect message number

        C=  x33xxxxxxxxx0088
                        0088= desired warning message
             3=#digits-1 to be output
            3=0xxx: use address in R2
              x011: digit output, suppress leading O's

    Or,...
        R2= xxxxxxx004Bxxx91
                          91hex=145 decimal
                   004B= indirect message number

        C=  xF1xxxxxxxxx0088
                        0088= desired warning message
             1=#digits-1 to be converted to decimal
            F=1xxx: use contents of B register
              x111: suppress up to 6 leading O's

    Or,...
        R2= 03xxxxx004Baaaaa
                       aaaaa= address to find ASCII output
                   004B=indirect message number
           03=#characters-1 to be output

        C=  x01xxxxxxxxx0088
                        0088= desired warning message
             1= output number of chars found in R2(15-14)
            0=output is in ASCII form already, resides at
                  address found in R2.
```

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/29/82 | MB | documentation |
| 01/27/83 | MB | Poll error handle, XM=0 suppress |
| 03/04/83 | MB | Saved 3 RSTK levels |

04/11/83 MB Added KILLKY call.

## 11.43 MFERR* - Error message driver

Category: GENUTL File: TI&ERD::MS

```
Name:(S) MFERR*  -  Error message driver
Name:    MFERR-  -  Error message driver

Purpose:
    Display error messages from standard message tables.

Entry:

 (1)----------------
  |
  | P=  1xxx    This is a Parse error (i.e., re-
  |               display input line w/cursor backup)
  |     x1xx    Do not store ERRN
  |               (Else store ERRN and ERRL)
  |     xx1x    Display msg only (Else display
  |               "ERR:" or "ERR L:", too)
  |     bit0 not used at present (**)
  |

 (2)----------------
  |
  | C(B)= message ID number in Hex.
  | C(3-2)= LEX ID# in Hex  (=00 for mainframe tbl)
  |

 (3)----------------
  |
  | If P=1xxx  (parse error):
  |    INBS points to first char of INput Buffer, with
  |       a 3 nibble length field preceding it.
  |
  |    D1 points to char in input buffer w/error
  |
```

```
|    A(A)= Address of prompt string for input
|          re-display (prompt must be enclosed in
|          delimiters, both sides. Delimiters
|          can be any byte value. E.g., prompt
|          string for an editor might look
|          like  xCmd:x  , where x's are any
|          matching byte value.)
|    or =0 For "use BASIC prompt string" (defaults
|          to the prompt string  3>3, where the
|          3's are the matching delimiters).
|
```

(**) Bit0 of the P register is reserved for future applications, as a way for the LEX file which generated the error to communicate with other LEX files; this bit can be detected during the pERROR poll in R0(S). The meaning of this bit is not yet decided. In the meantime, bit0 must=0.

```
Entry for MFERR- :
     D0 as C(3-0) above.


Exit:
     P      = 0

Calls:       POLL, fCALC?, CRLFND, UPDCRL,
             SFlagC, TBMSID, DOASCI, TBMSTX, A=CUR, AVS=C,
             AVS2DS, CHIRP, XDELAY, CRLFSD, BLDDSP, MFLG=X,
             R<RST2, RST2<R.  Might jump to ONERR.
           Parse errors also call:
             CKINF-, DSPBUF, DSPCNA, DSPCHA, CURSFL, CURSRR,
             ESCSEQ

Uses.......
 Exclusive:  A(W), B(W), C(W), D(W), P, D0, D1, R0
             R2 (only for MFERsp entry with text insertion;
                 otherwise not used)
             S13 is tested for: "Running program?"
                 If you're calling this routine just for
                 message display, watch out for S13!!!
             Available Memory (starting at AvMemSt) is
                 also used as a building buffer for msg.
           PARSE ERRORS also use:
             R3 (stores prompt address and #cursor-rights)
             R1, R2 (used in SENDWD)
             STMTR0 (in CKINFO and SENDWD)
 Inclusive:  Same

Stk lvls:    4 (parse errors only)
             2 (all other errors)
```

NOTE:

Parse errors re-prompt and rebuild the input line. The prompt is built in the display observing WIDTH. This is not a problem with the BASIC prompt (">"), since it is only one character; but an external system using a multi-character prompt should be aware that the prompt, after a parse error, may be split between two lines. (This feature was incorporated to accomodate INPUT prompts.)

Messages are built in Available Memory, which is used as a temporary buffer. This can cause a MEMERR; see the MEMERR routine for details.

If the error message number at entry is the eMEM constant (18hex), the message routines will automatically invoke the MEMERR routine, and an Insufficient Memory error will result.

Any error entering through MFERR* (includes MFERR and BSERR) disallows text insertion. Some applications may construct error messages which allow text insertion; if you want to issue these messages as errors you have three choices:

1) Issue them without any text insertion (use MFERR*, MFERR or BSERR)
2) Issue them as warnings, made to look like errors (use MFWRN) (see IDS volume I, chapter "Message Handling").
3) Call MFERsp entry point (see MFERsp heading).

Detail:

```
    R0 usage:
     F E D C B A 9 8 7 6 5 4 3 2 1 0
    ---------------------------------
    | |   | |                 |     |
    ---------------------------------
      |   |  +- wrng or error    +- msg number
      |   +- insert codes
      +- option flags
```

Algorithm:

```
   (1) Put option flags in C(S).
       Save options and LEX#, msg# in R0.
       Call POLL
       If Parse error, calculate #backups and store
        with A(A) in R3.
       If eMEM constant, branch to MEMERR.
 MFER.6 If "don't store error#" option go to (2)
        Else, store error# in ERR#.
        If running program (S13=1), store Line#.
```

```
       (2) If running program (S13=1), and not a warning,
            and ON ERROR in effect, branch to ONERR.
           If "message text only" option, go to (4)
           Build LEX ID prefix for message.
           Build "ERR" or "ERR L".
           If running program (S13=1), build line#.
           Build ":"
       (4) Build message text.
           Display entire message.
           Beep.
           Send CR, LF.
           If warning, return.
           If not parse error and S13=1, position D0
            to line# or @, return.
           (Parse error:)
             Set up CKINFO for SENDWD, send out prompt.
             Redisplay input line.
             Move cursor far left.
             Send out required # of cursor-rights.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/29/82 | MB | documentation |
| 06/18/83 | MB | deleted P=xxx1 entry flag |

## 11.44 MFERsp - Error Message With Text Insertion

Category: GENUTL File: TI&ERD::MS

Name:(S) MFERsp - Error Message With Text Insertion

Purpose:
Special entry point into error message handler, allowing text insertion (only in those known messages which have insertion points).

Entry:

(1)-----------------

```
    |
    | R0(S) = entry options as specified for P in MFERR*
    |


  (2)-----------------
    |
    | R0(B)= message ID number in Hex.
    | R0(3-2)= LEX ID# in Hex  (=00 for mainframe tbl)
    |


  (3)-----------------
    |
    | Parse errors: Same as condition (3) for MFERR*.
    |


  (4)-----------------
    |
    | Text insertion: Same as condition (3) for MFWRN.
    | (See "Details" under MFWRN for examples.)
    |
```

All other details as specified in MFERR* .

See "NOTE", "Details" and "Algorithm" entries under MFERR*.

Detail:
MFERsp should be called (as a subroutine) as follows:

```
        <set R2 according to text insertion options>
        <set C(14-13) according to text insert options>
        <set C(S) bits according to MFERR* options>
        <set C(3-0)=message number>
        R0=C                    Store options, msg_# in R0
        SETHEX
        GOSBVL =POLL            pERROR poll.
        CON(2) =pERROR
        CPEX   15               In case poll error, options.
        P=     12               P value for "error".
        LCHEX  00F              In case poll error...
        GOC    LABEL1           CRY=poll error.
        ?XM=0                   Poll handled?
        GOYES  LABEL3           Yes! Abort message.
        C=R0
        LCHEX  F                C(12)=F for "error" flag.
LABEL1  GOSBVL =MFERsp
LABEL3  P=     0                (if necessary from ?XM=0
```

```
     .....                       jump, above....)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/22/83 | MB | documentation |

## 11.45 DOASCI - Send ASCII bytes to DAT0

Category: GENUTL File: TI&ERD::MS

Name:(S) DOASCI - Send ASCII bytes to DAT0
Name:(S) DOASC+ - Send ASCII bytes to DAT0

Purpose:
Build a buffer of ASCII characters starting at D0;
the ASCII characters can originate from four types:
1) BCD digits
2) HEX digits
3) numeric conversion from Hex-to-Dec
4) existing ASCII bytes (or tokens)
Output can reside in one of two places:
1) in B register
2) in DAT1

Entry:
D0= output address (must be less than AVMEME pointer)

B register or D1: source of text insertion.
C(1): type of insertion.
C(0): how many characters in insertion.

```
    B
    -----
     * actual output characters or digits
         if C(1)= 1xxx
     * additionally, if C(1)= 0000, upper byte
         of B contains control nibbles.
```

```
D1
-----
  = address of output characters if C(1)= 0xxx

C(1)
-----
 1xxx   use contents of B register as output
 0xxx   use address in D1 to find output

 x000   Output is already in ASCII form

     Digit output (digits can be Hex or Dec):
 x001   Digit output-- replace leading 0's with blanks
 x010   Digit output-- don't suppress leading 0's
 x011   Digit output-- suppress leading 0's


     Hex-to-Dec conversions always generate
       decimal numbers with 7 digits:
 x100   Hex-to-Dec: suppress up to 3 leading 0's
 x101   Hex-to-Dec: suppress up to 4 leading 0's
 x110   Hex-to-Dec: suppress up to 5 leading 0's
 x111   Hex-to-Dec: suppress up to 6 leading 0's


C(0)
-----
  For C(1)= 1000 ("ASCII output is in B")
      C(0)= #nibbles-1 to be output. Hence the
            #nibs MUST be even!!; C(0) odd. E.g.,
            if 5 chars for output, C(0)=9.

  For C(1)= x0xx (hex or dec digit output)
      C(0)= #digits-1 to be output, hence
            no more than 16.

  For C(1)= x1xx (hex-to-dec conversion)
      C(0)= #digits-1 in number to be converted
            Max hex value for conversion is FFFFF
            (1048575 dec), hence C(0) must be 4
            or less.

  For C(1)= 0000 ("ASCII output from DAT1")
      C(0)= 0: no output
            1: Send out specified number of
               character; B(15-14)= #chars-1.
            2: Send out chars until ASCII termin-
               ator is found.  ASCII terminator
               is passed in B(15-14) (usually
               an FF terminator, but any byte
               value can be used).
```

Entry for DOASC+:
    This entry point is for "ASCII output from DAT1"
    only:
        D1 points to output already in ASCII form
        C(15-14)= #bytes to output
    DOASC+ then sets C(B)=01 for appropriate codes.

Exit: (May exit through MEMERR if not enough memory)
    Carry clear
    P = 0
    B(A) = # bytes left in available memory past buffer.
    D0 points to FF terminator, ready for another call.

Calls: HEXDEC (only for hex-to-dec conversion;
        i.e., only if C(1)=x1xx)
    MOVEU3 (only for ASCII output from DAT1;
        i.e., only if C(1)=0000)

Uses.......
    P, A(W), B(W), C(W), D(15-13)
    D0
    Uses D1 only if C(1)=0 (i.e., only if ASCII output
        from DAT1; otherwise D1 not changed). And then,
        D1 is only moved past source ASCII.

Stk lvls: 1

Detail:
Fills DAT0 with characters from B register or from DAT1 (as specified by calling routine). An FF terminator is placed at the end of the buffer, ready for a call to BF2DSP or BF2STK.
AvMemEnd is checked for sufficient memory. This is why D0 at entry must be less than AvMemEnd.

If ASCII output from DAT1, maximum #characters is 255.
If digit output, maximum number of digits is 16. If ASCII from B, maximum number of characters is 8.

If source is HEX or BCD digits, converts to ASCII equivalents first, for output to DAT0.

For numeric Hex-to-Dec output, conversion to BCD is performed, then converted to ASCII for output to DAT0.

Algorithm:
    Copy control nibs from C to D, calculate
      #bytes in AvMEM.
    Do:
      If ASCII output, copy bytes to DAT0.

```
        If Hex-to-Dec, call to HEXDEC, then digit output.
        If Digit output, convert digits to ASCII and output.
      As chars are output, decrement #bytes in AvMEM.
      Terminate buffer with FF.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/25/82 | MB | Documentation |

## 11.46 MOVE*M - Move Memory Up or Down Without Ref Adj

Category: GENUTL File: TI&UTL::MS

Name:(S) MOVE*M - Move Memory Up or Down Without Ref Adj

Purpose:
Move memory up or down with no reference adjust.

Entry:

```
A(A)  =  Source address
B(A)  =  Length of block to move in nibs
C(A)  =  Dest address
```

Exit:

```
All entry conditions
P     =  0
```

Calls: MOVEDM, MOVEUM

Uses.......
 Exclusive: A, C(A), D0, D1
 Inclusive: A, C(A), D0, D1, P

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|

```
    06/14/82      FH        Designed and coded.
```

## 11.47 MVMEM - Move File Memory W/Ref Adjust

Category: GENUTL    File: TI&UTL::MS

```
Name:    MVMEM   - Move File Memory W/Ref Adjust
Name:(S) MVMEM+  - Move File Memory W/Ref Adjust

Purpose:
     Move memory in a file chain up or down with reference
     adjust.  Works for either MAIN or Independent RAM.
     RFADJ is called, and pointers MAINEN -> AVMEMS and
     CURRST -> CURREN are updated if they fall into the
     block that moved.  Note that if the pointer value falls
     outside the block that moved but inside the area into
     which it moved, no action is taken.  If the source of
     the move is NOT EQUAL to the corresponding file header
     address passed in C(A), then that file header's chain
     length is also adjusted.

Entry:
     A(A)    =  Starting address to move up or down.  Equal
                to C(A) if adding or deleting file to/from
                file chain.
     B(A)    =  Offset (dest address - source address)
     C(A)    =  Address of header of file containing address
                to be moved.  File chain length field of the
                header will be updated to new length if and
                only if C(A) # A(A).  If adding or deleting a
                file to or from the chain, this address should
                point to the following file header in the file
                chain or to the end of the chain.
     P       =  0
    MVMEM:
     D(S)    =  Device code for memory device
     D(B)    =  Port number if port device
     D(7-2)  =  Nibs 8-3 of port's configuration table entry
    MVMEM+:
     D entry state will be computed from C(A)
```

```
Exit:
     R0      =  A(A) entry: starting address of move
     R2      =  C(A) entry: start of file header
     B(A)    =  Entry state
     P       =  0
    Carry clear:
     Memory moved and references adjusted
    Carry set:
     C(3-0)  =  Error code if error occurred:
                   eMEM   - Insufficient Memory
                   eILACS - Illegal Access (if ROM or EPROM)


Calls:       LOCADR, FLADDR, RMEMCH, MOVE*M, ADJREF

Uses.......
 Exclusive: A,B,C,D,D0,D1,R0,R1,R2
 Inclusive: A,B,C,D,D0,D1,R0,R1,R2,SCRTCH(4-0)

Stk lvls:   3

NOTE:
     NO CHECK IS MADE to verify that the starting address
     actually falls within a file chain or whether the port
     specified corresponds to the specified address.

Algorithm:
  MVMEM+ :
     Compute memory device info
  MVMEM  :
     If move is memory expansion then
        Check memory (return if error)
     If source # file header start then
        Update chain length
     Move memory
     Adjust references

History:

    Date       Programmer               Modification
  --------     ----------     ----------------------------------
  06/09/82         FH         Designed and coded.
                   SW         Check for ROM file
  02/15/83         FH         Packed, updated documentation
```

```
+--------------------------------------------------+-------------------+
|                                                  |                   |
|  KEYUTL  -  Keyboard Utilities                   |  CHAPTER  12      |
|                                                  |                   |
+--------------------------------------------------+-------------------+
```

## 12.1 CHEDIT - Character Editor

Category: KEYUTL     File: MN&ED::MS

Name:(S) CHEDIT - Character Editor

Purpose:
Accepts keyboard input and edits line in display.

Entry:
P=0, Hexmode

Exit:
P=0
If carry set then A(A)=Function code.
If carry clear then CHEDIT was terminated by an immediate execute key. R3(A)=Definition length. D1 points to first char of definition.

Calls: CHEDEX, CHROUT, DSPCHA, DSPCHR, DSPCL?, DSPSPC, KEYRD, TBLJMC, WRIT05, WRITE, bf2dsp.

Uses:
A,B,C,D,P,D0,D1,R0,R3,ST,DEFADR,USRSTA,32 nibs at SCRTCH.

Stk lvls: 6

Detail:
This subroutine implements a character editor which accepts keyboard input and edits display as needed until a key is entered which is not meaningful in character edit mode. The keycode of the terminator is returned in the A register. The following keys are terminators:

A(A) Key# Function

```
13 -- 38 -- EndLine
14 -- 43 -- Attention
15 -- 46 -- RUN key
16 --112 -- CONTinue key
17 --102 -- SST key
18 -- 50 -- Cursor up
19 -- 51 -- Cursor down
20 --162 -- Cursor to top
21 --163 -- Cursor to bottom
22 --155 -- g Attention
23 --111 -- CALC mode key
24 -- 99 -- OFF key
25 --164 -- g EndLine (Cmd Stack)
```

Although these keycodes map to the same values as certain control keys (ctrl-M through ctrl-Y), hitting the CTRL sequence followed by a key will NOT be interpreted as one of these terminators with the exception of CTRL-M. They will simply be put into the display as funny-looking characters.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/23/82 | BS | Updated documentation |
| 11/05/82 | NM | Rewrote |

## 12.2 KEYRD - Read A Key

Category: KEYUTL File: MN&ED::MS

Name:(S) KEYRD - Read A Key

Purpose:
Read a key and return a pointer to its expanded value.

Entry:
HEX mode.
f1RPTD and last position in keybuffer contain information necessary for repeating keys to work.

**Exit:**

P=0.

DEFADR contains pointer to expanded value:

DEFADR: Length of string in bytes.

DEFADR+2: Key type:

0 = Single ASCII character. Includes control characters 0-31, which should usually cause some action in the editor calling KEYRD.

1 = ASCII control character. Must subtract #40 from the 1-byte definition we are pointing to. These characters should be interpreted as text, and should not cause any special action in the editor.

2 = User-defined key; Terminating.

4 = User-defined key; Non-terminating.

6 = User-defined key; Non-displaying.

8-F = LEX entry with lower 3 bits as follows:
- bit 0: Parenthesis needed.
- bit 1: Trailing space needed.
- bit 2: Leading space needed.

(spaces & paren not included in string length field)

DEFADR+3: Address of text.

**Calls:** ALMSRV, ASLW5, BLDDSP, CSLW3, FINDAj, FLIPO, FLIPCS, FPOLL, GETDEF, KEYTYP, MTADDR, POPBUF, RPTKY, SETTMO, SFlagC, SLEEP, Sflag?, VWFC-2, WIPOUT, cksreq, range, sflagt, usrsta.

**Uses.......**
A,B,C,D,P,D0,D1,R3,USRSTA (for holding ST), DEFADR (for definition), 32 nibs at SCRTCH.

**Stk lvls:** 5

**Algorithm:**

KEYRD: Build display.

KEYR50: Perform WTKY fastpoll.
If handled then goto KEYR69.
Check for repeating keys (RPTKY).
If we have a repeating key then goto KEYR72.
Build display.

```
         Set 10-minute timeout (SETTMO).
KEYR60:  Go to light sleep (SLEEP).
         If key in buffer then goto KEYR70.
         If 10-minute timeout not expired then goto KEYR60
           else return OFF-key definition.
KEYR69:  Set up registers after poll.
         Goto KEYR72.
KEYR70:  Pop key# from buffer.
KEYR72:  Put key# and logical keycode in R0.
         Perform KYDF fastpoll.
         If handled then if S0=0 (not-returning-definition)
           then goto KEYR50 else return.
         If VIEW flag is clear then goto KEYR75.
         Clear VIEW flag.
         Get key definition; if none then goto VIEWUN.
         Write definition to LCD.
         Goto VIEW30.
VIEWUN:  Write "Unassigned" to LCD.
         Loop until keys up (VWFC-2).
         Goto KEYR50.
KEYR75:  If CTRL flag clear then goto KEYR80.
         If keycode not in CTRL'able range then goto
           KEYR80.
         Return CTRL key definition.
KEYR80:  If USRX flag clear then goto KEYR90.
         Clear USRX flag.
         Toggle USER flag (carry reflects old state).
         Goto KEY100.
KEYR90:  Carry := USER flag.
KEY100:  If carry clear (not USER) then goto KEY110.
         Fetch key redefinition.  If non-existent then
           goto KEY110.
         Return redefined key definition.
KEY110:  RSTK=KEY120 (return address in case we do
           internal processing).
         {start of internal processing jump table}
         If keycode = LC key then goto LOWERC.
         If keycode = USER key then goto USERK.
         If keycode = CTRL key then goto CTRL.
         If keycode = VIEW key then goto VIEWK.
         If keycode = temp-user key then goto USERX.
         If keycode = Last-err key then goto lerrm.
         {end of internal processing jump table}.
         Pop RSTK.
         If keycode in range of typing aids then goto
           NEWTOK.
         If LC flag set then flip case if appropriate
           (FLIPCS).
         {we have a simple 1-char definition}
         Look up key definition in KEYCOD table and return
           definition.
```

```
KEY120: {we have finished internal processing}
        If keybuffer empty then zero out last entry in
          keybuffer to disable repeating key.
        Goto KEYR50.
NEWTOK: Find typing aid definition (MTADDR).
        Return definition.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/02/82 | NM | Began to write. |

## 12.3 -LINE - Delete Through End Of Line

Category: KEYUTL File: MM&ED::MS

Name:(S) -LINE - Delete Through End Of Line

Purpose:
Send an ESC K to display to delete through end of line

Entry:
P = 0

Exit:
P = 0

Calls: ESCSEQ

Uses.......
Exclusive: C(B)
Inclusive: A(W),B(W),C(W),D(W),D0,D1

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/16/82 | BS | Added documentation |

## 12.4 RPTKY - Check For Repeating Keys

Category: KEYUTL     File: MN&ED::MS

Name:(S) RPTKY - Check For Repeating Keys

Purpose:
Check for repeating keys.

Entry:
P=0.
HEX mode.
The last position of the keybuffer contains the key#
to look for.
System flag flRPTD indicates whether the key has begun
repeating yet.
User status bits have been saved into DSPSTA.

Exit:
Carry clear if: Key comes up before repeat interval.
Keybuffer non-empty.
No key in last position of keybuffer.
Carry set indicates that a repeat should be done.
Key# is in B[A].
Flag flRPTD = 1 iff carry set.
P=0.
TIMER1 has been reset to .5 sec.
User status bits have NOT been restored to ST.

Calls: CKSREQ, DEBNCE, IDIVA, TMRRST, WRTTM1, Sflag?,
SFlagC, SFlagS, usrsta.

Uses.......
A,B,C,D,P,DO,D1,ST

Stk lvls: 3

History:

| Date | Programmer | Modification |
|---|---|---|

| | | |
|---|---|---|
| 11/04/82 | NM | Wrote. |

## 12.5 CMD1ST - Set command stack pointer to 1st cmd

Category: KEYUTL     File: SB&CMD::MS

Name:(S) CMD1ST - Set command stack pointer to 1st cmd

Entry:
    None

Exit:
    D1 points to CMDPTR
    C(A)=0

Calls: None

Uses.......
 Exclusive: C(A)

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/28/83 | B.S. | Added documentation |

## 12.6 CMDS00 - Display Cmd Stack Entry

Category: KEYUTL     File: SB&CMD::MS

```
Name:    CMDS00  -  Display Cmd Stack Entry
Name:    CMDS10  -  Display Cmd Stack Entry
Name:(S) CMDS20  -  Display Cmd Stack Entry

Purpose:
     CMDS00 - Initializes to first command stack entry then
     CMDS10 - Puts up command stack prompt then
     CMDS20 - Puts up command stack entry and moves cursor
              to far left.
Entry:
     P      = 0
     CMDS10 and CMDS20 require that CMDPTR be set to specify
          which command should be displayed.

Exit:
     P      = 0

Calls:     BF2DSP,CMDFND,DSPCNA,CURSFL,CMD1ST

Uses.......
 Exclusive: D1,C(A),A(W)
 Inclusive: D0,D1,A,B,C,D

Stk lvls:  5

History:

   Date     Programmer           Modification
 --------   ----------   ----------------------------------
 07/28/83   B.S.         Added documentation
```

## 12.7 CMDFND - Find Nth Command Stack Entry

Category: KEYUTL     File: SB&CMD::MS

```
Name:(S) CMDFND  -  Find Nth Command Stack Entry

Purpose:
     Finds the command stack entry indicated by CMDPTR

Entry:
```

CMDPTR is number of entry to find (0-->first,F-->15th)

Exit:
D1 points to start of cmd stack entry (at length field)

Calls: None

Uses.......
Inclusive: D1,A(W),C(A)

Stk lvls: 0

Detail:
This routine starts with the newest command (pointed to by RAWBFR) and chains up stack toward the oldest entry until the specified entry is reached.

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/28/83 | B.S. | Added documentation |

## 12.8 CMDINI - Recalls CMDPTR and MAXCMD

Category: KEYUTL File: SB&CMD::MS

Name:(S) CMDINI - Recalls CMDPTR and MAXCMD

Purpose:
Recall CMDPTR and MAXCMD to A(0) and C(0)

Entry:
None

Exit:
A(0) = (CMDPTR)
C(0) = (MAXCMD)

Calls: None

Uses.......
 Inclusive: D1,C(0),A(0)

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/28/83 | B.S. | Added documentation |

## 12.9 SCRLLR - Scroll Left and Right

Category: KEYUTL File: SB&DSP::MS

Name:(S) SCRLLR - Scroll Left and Right

Purpose:
    Watch for scroll keys and perform display scroll

Entry:
    P = 0

Exit:
    P = 0
    A(B) contains keycode that is first in key buffer

Calls: ALMSRV, BLDDSP, BLDLCD, CKSREQ, D1=FC, FINDDO,
    GETSTA, POPBUF, RPTKY, SCRL60, SETFC, SETTMO,
    SLEEP, USRSTA.

Uses.......
 Exclusive:
 Inclusive: A(W),B(W),C(W),D(W),DO,D1

Stk lvls: 5

Detail:
    Sleeps and watches for scrolling key in the key
    buffer and causes the display to respond
    appropriately. Routine exits when a key is found

in buffer that isn't a scrolling key or when
display timer times out.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/19/82 | B.S. | Updated documentation |
| 07/18/83 | B.S. | Will not time out if a program is running |

## 12.10 FGTBL - State table for F & G shifted keys

Category: KEYUTL File: SB&FGT::MS

Name:(S) FGTBL - State table for F & G shifted keys

Purpose:
This table defines a state machine used to determine
how to process f and g shifted keys

Entry:
Do not enter

Detail:
The state machine has 7 input bits and 4 output bits.
The seven input bits are as follows
Bit 6 F key currently down
Bit 5 G key currently down
Bit 4 Some non-FG key newly down
Bit 3 g annunciator on
Bit 2 f annunciator on
Bit 1 Ghost bit
Bit 0 F or G key was down during last key scan
The ghost bit is used to indicate that an f or g
shift has been performed but the annunciator was
left on because the corresponding key was still
down.
The lower 4 bits are stored between key scans
in the display RAM nibble that contains the f and
g annunciators. The lower two bits do not affect

the display since there are no annunciators in the LCD to correspond to these bits.
These 7 bits form an offset into the table which gives the new "state" of the state machine and is stored back into display memory. If bit 4 is set but bits 5 and 6 are clear then all bits should be cleared following putting the f or g modified key codes in the buffer.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/18/83 | B.S. | Updated documentation |

## 12.11 KEYCOD - Keycode Map

Category: KEYUTL File: SB&KCM::MS

Name:(S) KEYCOD - Keycode Map

Purpose: System keycode map. Maps keys to their definition

Entry:
Do not enter

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | B.S. | Added documentation |

## 12.12 DEBNCE - Debounce and scan keyboard

Category: KEYUTL File: SB&KEY::MS

Name:(S) DEBNCE - Debounce and scan keyboard
Name:(S) KEYSCN - Scan keyboard

Purpose:
Scans keyboard and puts all new keys in key buffer

Entry:

Exit:
P = 0
D0=(5) =DISINT (except for WARMST exit)

Calls: None

Uses.......
Inclusive: A(W),B(W),C(W),D0

Stk lvls: 0

Detail:
The keyboard is scanned and a bit map of all keys down is made. If the number of keys down (not counting the ON key is greater than 3 then no change is made to the bit map or key buffer and KEYSCN returns immediately. The map is compared to the map that was made the last time the routine was called. The new bit map is saved for the next call. All keys that have gone down since the last call (up to 7 new keys) are added to the key buffer (space permitting). The logical keycodes for unshifted keys that are generated and stored in the buffer are as follows:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Q 01 | W 02 | E 03 | R 04 | T 05 | Y 06 | U 07 | I 08 | O 09 | P 0A | 7 0B | 8 0C | 9 0D | / 0E |
| A 0F | S 10 | D 11 | F 12 | G 13 | H 14 | J 15 | K 16 | L 17 | " 18 | 4 19 | 5 1A | 6 1B | * 1C |
| Z 1D | X 1E | C 1F | V 20 | B 21 | N 22 | M 23 | ( 24 | ) 25 | eol 26 | 1 27 | 2 28 | 3 29 | - 2A |

```
| ON| f |  g|RUN| Lf| Rt|SPC| Up| Dn|   | 0 | . | = | + |
| 2B|   |   | 2E| 2F| 30| 31| 32| 33|   | 35| 36| 37| 38|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

F shifted keys have 56 added to these values.
G shifted keys have 112 added to these values.

The f and g keys themselves are never put in the buffer.

A state machine is used to control turning on and off of the f and g annunciators. See documentation on FGTBL for further details.

The key buffer looks like this:

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-----------+
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-----------+
^ ^                                         ^
| +---- KEYBUF (points to first of 15       |
|              bytes of key buffer)         |
+---- KEYPTR (points to nibble that         |
              tells how many keycodes       |
              buffer contains)              |
          KEYSAV (points to 14 nibbles that |
                 hold previous key bit map) --+
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/16/82 | B.S. | Updated documentation |
| 11/16/82 | M.B. | Updated exit conditions |

## 12.13 POPBUF - Pop Key Buffer

Category: KEYUTL File: SB&KEY::MS

Name:(S) POPBUF - Pop Key Buffer

Purpose:
Pops a key from keyboard buffer into B(A)

Entry:

Exit:
Carry set ==> Key buffer was empty
clear ==> B(A) contains keycode
Key# just popped has been copied to last position
in keybuffer.

Calls: None

Uses.......
Inclusive: C(W),B(A),D0

Stk lvls: 0

Detail:
Disables interrupts and pops a key from buffer.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/16/82 | B.S. | Updated documentation |
| 11/04/82 | NM | Add copy of last key to key14 slot |

```
+------------------------------------------+-------------------+
|                                          |                   |
|   MATH    -  System Math Functions       |  CHAPTER  13      |
|                                          |                   |
+------------------------------------------+-------------------+
```

## 13.1 ADDONE - Add One

Category: MATH      File: JT&MTH::MS

Name:(S) ADDONE - Add One
Name:(S) SUBONE - Subtract One

Purpose:
To compute X+1 & X-1 for X an internal number.

Entry:
Standard floating point math input with (A,B)=X.
sINFRD(s10)&sNEGRD(s11),rounding modes, are consulted
only if X+1=0 (or X-1=0) in which case the result may
be +0 or -0 depending on the mode.(see AD15s)

Exit:
Standard floating point math output.

Calls: Goes to AD15s .

Uses.......
Inclusive: P; A,B,C,D;
HD.ST.[SB];

Stk lvls: 0

NOTE:
Can raise no XM=1 xception . (clrs SB but not XM)

## 13.2 1/X15 - 1/X

Category: MATH File: JT&MTH::MS

Name:(S) 1/X15 - 1/X

Purpose:
To compute 1/x

Entry:
Standard floating point math input.

Exit:
Standard floating point math output.

Uses.......
Inclusive: P; A,B,C,D;
HD.ST.[SB,XM];

Stk lvls: 0

NOTE:
Goes to DV15S (divides 1 by x )

## 13.3 AD2-15 - Add two 15 digit forms

Category: MATH File: JT&MTH::MS

Name:(S) AD2-15 - Add two 15 digit forms
Name:(S) AD2-12 - Add two 12 digit forms
Name:(S) ADDF - Add for finite args only
Name:(S) AD15M - Add according to modes
Name:(S) AD15s - Add with XM sticky

Purpose:
To compute the sum x+y .

Entry:
```
     Standard floating point math input.
     AD2-15 assumes NOT(round to -Inf)  (i.e. x+(-x)=+0)
     AD15s has rounding mode inputs (SB cleared inside)
     s10 & s11 set ==> rnd to -inf (i.e. x+(-x) = -0 )
      else x+(-x) = +0.

     CODE:  =AD2-12  GOSUB  SPLTAC
     ----   =AD2-15  ST=0   sNEGRD   (s11) NO round to NEG.
            =AD15M   XM=0
            =AD15s   SB=0            (add uses SB for result!)
```

Exit:
```
     Standard floating point math output.
     XM=1 implies Inf+(-Inf)
```

Calls:      (none)

```
Uses.......
 Inclusive: P; A,B,C,D;  ST.[s11 for AD2-15 only];
            HD.ST.[SB,XM];
```

Stk lvls:   0

NOTE:
```
     The main entry AD2-15 forces rnd to nearest
      (same result except for rnd to -inf).
     Results are truncated. (e.g. 1 - 1E-100 -->
      .999999999999999 with SB=1 ! )
```

## 13.4 MP2-15 - Multiply

Category: MATH      File: JT&MTH::MS

```
Name:(S) MP2-15  -  Multiply
Name:(S) MP15S   -  Multiply without clearing SB
Name:(S) MULTF   -  Multiply for finite args only
Name:(S) MP1-12  -  Multiply for one 12-form
Name:(S) MP2-12  -  Multiply for two 12-forms
```

Purpose:
    To compute x*y

Entry:
    Standard floating point math input.
    MULTF & MP15S: SB & XM are not cleared on entry.

```
CODE:  =MP2-12  GOSUB  SPLITA
----   =MP1-12  GOSUB  SPLITC
       =MP2-15  SB=0
                XM=0
       =MP15S
```

Exit:
    Standard floating point math output.
    XM=1 implies 0*Inf

Calls:    (none)

Uses.......
 Inclusive: P; A,B,C,D;
            HD.ST.[SB,XM];

Stk lvls:   0

NOTE:
    Reg. D has the 16 digit mant. of x*y if D(S)#0,
      (mant of Inf & NaN is not put into D,but D(S)=0 here)
    Results are truncated to 15 digits.
    Unfortunately SB=1 when XM=1 on exit. (This is true for
     most math routines.)

## 13.5 DV2-15 - Divide

Category: MATH        File: JT&MTH::MS

Name:(S) DV2-15 - Divide
Name:(S) DIVF   - Divide for finite args only
Name:(S) DV15S  - Divide without clearing SB

```
Name:(S) DV15M   -  Divide (same as DV2-15)
Name:(S) DV2-12  -  Divide for two 12-forms

Purpose:
     To compute  y/x

Entry:
     Standard floating point math input.

     CODE:   =DV2-12   GOSUB  SPLTAC
             =DV2-15
             =DV15M    XM=0
                       SB=0
             =DV15S


Exit:
     Standard floating point math output.
     XM=1 & P=3 implies c/0 where 0<|c|<Inf
          & P=4    "    0/0 or Inf/Inf

Calls:      (none)

Uses.......
 Inclusive: P; A,B,C,D;
            HD.ST.[SB,XM];

Stk lvls:   0

NOTE:
     Divides (A,B) by (C,D) .
     Results are truncated to 15 digits.
```

## 13.6 SQR15 - Square Root

```
      Category:  MATH       File:  JT&MTH::MS


Name:(S) SQR15   -  Square Root
Name:(S) SQR17   -  SQRT for finite arguments only

Purpose:
```

To compute SQRT(x)

Entry:
Standard floating point math input.

Exit:
Standard floating point math output.
XM=1 implies SQR(neg)

Calls: (none)

Uses.......
Inclusive: P; A,B,C;
MD.ST.[SB,XM];

Stk lvls: 0

NOTE:
Certain 15-form inputs can exit with SB=0, even though the result is inexact! e.g. SQR(1E14+1)-->1E7 & SB=0.
This occurs from BSR instr. before SQR30.

## 13.7 INVNaN - Create IVL NaN

Category: MATH File: JT&MTH::MS

Name:(S) INVNaN - Create IVL NaN

Purpose:
To create an internal NaN and set XM for IVL.

Entry:
C(B)=two nib. mainframe error msg code.

Exit:
(A,B):=NaN with B(14..11):= 4nib msg code
C(A):= 4nib msg code for input to MESSAGE ROUTINE.
XM:=1 (indicates xcpt'n) & P:=IVP (IV xcpt'n)
B(XS)=9 (if in DEC MODE !). This indicates a 15-form INVNaN (i.e. created in math routine -- input NaNs from SPLITA will have F instead of 9 in B(XS)).

```
        This causes INVNaNs (and their encoded message)
        to be more significant than input NaNs and thus
        will be preserved when two NaNs enter a function.

      i.e.  A = 000000000000F01
            B = 000mm0000000900
            C = -----------000mm


   Calls:      (none)

   Uses.......
    Inclusive: P; A,B,C(A);
               HD.ST.[XM,SB];

   Stk lvls:   0

   NOTE:
        CAUTION: This routine will set SB (unfortunately).
```

## 13.8 LN1+15 - LN(1+X)

```
      Category:  MATH       File:  JT&MTH::MS


   Name:(S) LN1+15  -  LN(1+X)
   Name:(S) LN1+XF  -  LN(1+X) for finite args only

   Purpose:
        To compute  ln(1+x) from x.

   Entry:
        Standard floating point math input.

   Exit:
        Standard floating point math output.
        XM=1 & P=3  implies  LN(0)
             & P=4     "     LN(negative)


   Calls:      ADDONE, LN15,
```

```
Uses.......
 Inclusive: P; A,B,C,D; R regs [0]; ST.[s10];
               HD.ST.[SB,XM];

Stk lvls:   1
```

## 13.9 LN15 - Natural Logarithm

Category: MATH      File: JT&MTH::MS

```
Name:(S) LN15     - Natural Logarithm
Name:(S) LN12     - LOG for 12-form args.
Name:(S) LN30     - LOG entry for finite args only.

Purpose:
     To compute  LN(x)

Entry:
     Standard floating point math input.

Exit:
     Standard floating point math output.
     XM=1 & P=3 implies  LN(0)
          & P=4    "     LN(negative)

Calls:      SHF10, (GOES TO DV15?)

Uses.......
 Inclusive: P; A,B,C,D; R regs [0]; ST.[10];
            HD.ST.[SB,XM];

Stk lvls:   1
```

## 13.10 EXP15 - EXP(x) (exponential fcn)

Category: MATH File: JT&MTH::MS

```
Name:(S) EXP15   -  EXP(x)     (exponential fcn)
Name:(S) EX-115  -  EXP(x)-1   (EXPM1(x))
Name:(S) DXP100  -  EXP for double precision arg

Purpose:
     To compute  e^x

Entry:
     Standard floating point math input.
     ( Uses s11 to distinguish e^x from [e^x - 1] )
     DXP100: finite args only; s11=0; R0=low order
      digits of double precision argument.

Exit:
     Standard floating point math output for EXP15.
     EX-115 outputs e^x in (A,B)&SB and [e^x - 1] in
      (R1,R0)&SB  (SB is the same for both).

Calls:   DBLSUB,SHFLAC,1/X15S,STAB1,EXAB1,ADDONE,SUBONE
         and other local subroutines.

Uses.......
 Inclusive: P; A,B,C,D; R regs [0]; ST.[10,11];
                 MO.ST.[SB,XM];
            EX-115 also uses R1.


Stk lvls:   1

NOTE:
     When xpon(e^x)>19999 then EXP15(x):=9.99...99E+19999 .
     When      "     < -19999 then EXP15(x):=9.9..99E-19999 .
```

## 13.11 LGT15 - Log base 10

Category: MATH File: JT&MTH::MS

Name:(S) LGT15 - Log base 10

Purpose:
To compute the base 10 logarithm of x.

Entry:
Standard floating point math input.

Exit:
Standard floating point math output.

Calls: NRMLAB, EX15, LN15, LNC10+, DV15S, MAKE1

Uses.......
Inclusive: P; A,B,C,D; R regs [0]; ST.[10];
HD.ST.[SB,XM];

Stk lvls: 2

NOTE:
LGT(10^n) returns n exactly.

## 13.12 YX2-15 - Y to the X power

Category: MATH File: JT&MTH::MS

Name:(S) YX2-15 - Y to the X power
Name:(S) YX2-12 - Y^X for 12-form arguments

Purpose:
To compute y^x

Entry:
    Standard floating point math input.
    s11 can be used to compute [Y^X - 1] by entering
        later with s11=1.

Exit:
    Standard floating point math output.

Calls:      LN   , EXP

Uses.......
 Inclusive: P; A,B,C,D; R regs [0,2,3]; ST.[sY=INF(s8),10,11];
            HD.ST.[SB,XM];
            [y^x - 1] uses R1 also.

Stk lvls:   3

NOTE:
    If |y^x| > 1E20000 or <1E-20000 then y^x-->1E(+/-)20000,
      these are the internal ovf/unf thresholds.

## 13.13 FAC15S - Internal Factorial

Category: MATH      File: PM&STA::MS

Name:    FAC15S  -  Internal Factorial
Name:    FACTF   -  Internal Factorial
Name:(S) FCSTRT  -  Internal Factorial

Purpose:
    Computes the factorial of the 15-digit quantity in
    registers A/B.

Entry:
    A/B -- normalized 15-digit quantity
    user modes set
    DECMODE

Exit:
    A/B -- factorial in 15-digit form
    SB set if result is inexact

XM set if NaN created
Carry=Set
DECMODE

Calls: FNPWDS,INFR15,SHFMLT
may exit through aornan

Uses.......
Inclusive: A,B,C,D,P,SB,XM

Stk lvls: 2

NOTE:
The result is accurate to 12 digits for all integer arguments i, where 0<=i<=253. A noninteger finite or -Inf argument causes a NaN to be created and XM set.

Algorithm:
A fast integer multiply method is used with adjustments for i=137 and 167 to insure full 12-digit accuracy.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/28/82 | PM | Documented routine |
| 06/25/82 | " | Fatal errors for noninteger args |
| 01/06/83 | " | Reviewed documentation |
| 01/13/83 | " | NaN created for invalid args |

## 13.14 uTEST - Perform comparisons

Category: MATH File: SM&MTH::MS

Name:(S) uTEST - Perform comparisons

Purpose: User Real Comparisons - <, >=, etc.

Entry: P encodes predicate (see Predicate table).
A:a C:c (Arg's are 12-dig forms a&c).

Exit: Carry=Result (Set=TRUE), P=CellN for pair, raises Invalid if Unordered and predicate contains one of ">" or "<" but not "?". If invalid, subsequent action based on user traps.

Calls: TST12A, (Also uRESXT - if INVALID raised)

Alters (INC): A,B,C,D,P,XM,SB,sIX

Stk lvls: MAX(3,MESSG)

Algorithm: See =TST15

| Date | Programmer | Modification |
|---|---|---|
| 07/09/82 | SB | Bugfix: HTRAP now works off of sIX |
| 02/07/83 | SB | Update header. |

## 13.15 EX12 - Return exponent of 12-dig arg

Category: MATH File: SM&MTH::MS

Name:(S) EX12 - Return exponent of 12-dig arg
Name:(S) EX15M - Return exponent of 15-dig arg (XM=SB=0)
Name:(S) EX15S - Return exponent of 15-dig arg
Name:(S) EXF - Return exponent of finite 15-dig arg

Purpose: Returns the exponent of given argument.

Entry:

EX12: 12-digit arg in A.
EX15M: 15-digit arg in A&B.
EX15S: 15-digit arg in A&B.
EXF: 15-dig finite arg in A&B.

Exit: A&B: y=EXPONENT(x) 15-digit form

Calls: SPLTA,XMOSBO,AFIN,=DZ10

Alters (INC): A,B,SB,XM,P,CARRY

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 6/01/82 | SB | Documented. |
| 9/22/82 | SB | EXPONENT(0) raises DVZ |
| 10/08/82 | SB | Code Pack: Tighter loop |
| 12/09/82 | SB | Improve Comments |
| 02/07/83 | SB | Update Header. |
| 03/31/83 | SB | Dedicated err msg: "EXPONENT(0)" |

## 13.16 SQRSAV - SQR for Chain calculations.

Category: MATH File: SM&MTH::MS

```
Name:(S) SQRSAV   -  SQR for Chain calculations.
Name:(S) ORXM     -  Set XM if sXM=1 and Set SB if sIX=1
Name:(S) ORSB     -  Set SB if sIX=1
Name:(S) SETSB    -  Set SB

Purpose:  SQRSAV-Puts XM & SB into status bits sXM & sIX,
                 calls SQR15M, and falls into ORXM which
                 establishes XM<--XM OR sXM, SB<--SB OR sIX.
                 This preserves exactness in SB and
                 exceptions in XM thru a call to SQR15M.

Entry:    SQRSQV:15-Digit arg in A and B.
                 DEC Mode
                 XM Set if previous exception.
                 SB Set if previous inexact calculation

Exit:     SQR(Arg) in 15-digit form in A and B
          DEC Mode
          XM Set if previous exception or SQR exception.
          SB Set if previous inexact or SQR inexact.

Alters:   A,B,C,P,SB,XM,CARRY, and status bits sIX,sXM

Calls:    SAVEXM,SQR15M,SETXM

Stack Levels: 1
```

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/02/83 | SB | Documented |

## 13.17 SAVGSB - Put SB into sINX

Category: MATH File: SM&MTH::MS

Name:(S) SAVGSB - Put SB into sINX
Name:(S) ORGSB - Set SB if sINX=1
Name:(S) SAVEXM - Put XM into sXM & SB into sIX
Name:(S) SAVESB - Put SB into sIX

Purpose: Routines save and restore SB and XM from status

Entry: See description above

Exit: See description above

Alters: SAVGSB - CARRY, status sINX
ORGSB - C[S], SB, CARRY
SAVEXM - CARRY, status sIX,sXM
SAVESB - CARRY, status sIX

Calls: Nothing

Stack Levels: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/02/83 | SB | Documented |

## 13.18 ARG12 - Return Arg of X+iY (12-dig args)

Category: MATH File: SM&MTH::MS

Name:(S) ARG12 - Return Arg of X+iY (12-dig args)
Name:(S) ARG15 - Return Arg of X+iY (15-dig args)
Name:(S) ARGF - Return Arg of X+iY (15-dig finite args)

Purpose: Argument of X+iY. Used by ANGLE.

Entry: ARG12: 12-Dig args- A:X, C:Y, =sRAD
ARG15: 15-Dig args- AB:X, CD:Y, =sRAD
ARGF : 15-Dig finite args- AB:Y, CD:X, =sRAD

Exit: A&B: ARG(X,Y)

Calls: SPLTB,MSN15,AFIN,SWAPXY,=DV2-15,SAVGSB,
ATAN15,ORGSB,PI/2D,=ADDF,XMOSBO.

Alters (INC): A,B,C,D,RO,R1,P,sIX,=sINX,sCOMP,sATAN,sSGN,
=sRAD,s+PI/2,SB,XM

Stk lvls: 2

Algorithm: Weed special cases, call ATAN15(Y/X)

History:

| Date | Programmer | Modification |
|---|---|---|
| 6/30/82 | SB | sAFFIN used in place of P |
| 10/06/82 | SB | Code Pack: Eliminate Proj Mode, Also bugfix (X,Y)=(FINITE,INF). |
| 11/15/82 | SB | Bugfix: ANGLE(0,0)=0 is EXACT. |
| 02/07/83 | SB | Update header. |
| 11/02/83 | SB | Additional Documentation |

## 13.19 SIN12 - Trig: Sine of 12-dig arg

Category: MATH File: SM&MTH::MS

```
Name:(S) SIN12   - Trig: Sine of 12-dig arg
Name:(S) COS12   - Trig: Cosine of 12-dig arg
Name:(S) TAN12   - Trig: Tangent of 12-dig arg
Name:(S) SIN15   - Trig: Sine of 15-dig arg
Name:(S) COS15   - Trig: Cosine of 15-dig arg
Name:(S) TAN15   - Trig: Tangent of 15-dig arg

Purpose:    SINE, COSINE, & TANGENT

Entry:      SIN12,COS12,TAN12 - Standard Math, 12-dig arg'ts
            SIN15,COS15,TAN15 - Standard Math, 15-dig arg'ts

                 All entries assume Status bit =sRAD encodes
                 the desired angle mode (SET=RAD MODE)

Exit:       A&B: 15-digit result.  COS & SIN entries also
                 produce TAN (or COT) magnitude in R0&R1.

Calls:      SPLTA,AFIN,SHFRAC,SHFRBD,PI/4,TWO*,DBLSUB,
            SHFLAC,FLIP8,FLIP10,FLIP11,GETCON,=MULTF,=1/X15,
            =DIVFCD,STAB1X,RCCD1X,=MP2-15,=ADDONE,=SQR15,
            FUDGE.

Alters (INC) : A,B,C,D,R0,R1,P,SB,XM,CARRY, and
            Status bits -  sIX, sINVRT, sTAN, sSGN, sSGNT.
            Current Value:  7      8      6    10     11

Stk lvls:   2

Algorithm:
    The absolute value of the argument is dbl word reduced
    by 2*Pi (or 360), then by Pi/2, and Pi/4 to obtain
    0<=Phi<=Pi/4.  A pseudo divide produces (X,Y) with
    0<=Y<=X and TAN(Phi) = Y/X .  Formulas;
            TAN(Phi) = Y/X
            SIN(Phi) = 1/SQRT(1+(X/Y)^2)
            COS(Phi) = 1/SQRT(1+(Y/X)^2)
    Related back to the argument via STATUS bits.

sIX    (7) : Local exactness.  Not set=Exact. (INEXACT flag)
sINVRT (8) : If set, use X/Y instead of Y/X   (INVERT flag)
sTAN   (6) : If not set, TAN is desired       (TAN flag)
```

```
sSGN  (10): Sign of result                  (SGN  flag)
sSGNT (11): Sign of TAN                     (SGNT flag)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 7/15/82 | SB | Fix to sign of 0 for COS(90), etc. |
| 8/12/82 | SB | Bugfix: Neg Exp in Radian Mode. |
| 10/29/82 | SB | Pack: INIT rearrangement |
| 12/09/82 | SB | Improve Comments, Label Change TRG150->REDUCE, Code Pack |
| 12/14/82 | SB | Label changes, code Pack in area where exactness established. |
| 02/10/83 | SB | Code Pack: Put XTENDE in line. |
| 03/31/83 | SB | Error msg change: TAN=INF replaces previous TAN or SEC=INF. |

## 13.20 TRC90 - Table of numeric constants

Category: MATH File: SM&MTH::MS

Name:(S) TRC90 - Table of numeric constants

Purpose: Constants used by the trig routines.

Entry: Values are accessed by a call to GETCON with a select code in P (See GETCON, GETVAL).

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/02/83 | SB | Documented |

## 13.21 ASIN12 - ArcSin Inv Trig (12-dig argument)

Category: MATH File: SM&MTH::MS

```
Name:(S) ASIN12  -  ArcSin Inv Trig (12-dig argument)
Name:(S) ACOS12  -  ArcCos Inv Trig (12-dig argument)
Name:    ATAN12  -  ArcTan Inv Trig (12-dig argument)
Name:(S) ASIN15  -  ArcSin Inv Trig (15-dig argument)
Name:(S) ACOS15  -  ArcCos Inv Trig (15-dig argument)
Name:(S) ATAN15  -  ArcTan Inv Trig (15-dig argument)
Name:(S) BRT30   -  Inv Trig, defined by status
Name:(S) BRIF    -  Inv Trig, finite arg, defined by status

Purpose:  ARCSINE, ARCCOSINE, ARCTANGENT

Entry:    ASIN12,ACOS12,ATAN12 - Stnd. Math, 12-dig arg'ts
          ASIN15,ACOS15,ATAN15 - Stnd. Math, 15-dig arg'ts

               All entries assume angle mode encoded in
               status bit =sRAD (set=RAD Mode).

Exit:     Standard math (15-digit result in A&B)

Calls:    SPLTA,AFIN,=INVNaN,PI/2,SWAPXY,STAB1X,
          =STAB2,=ADDONE,=EXAB1,=SUBONE,RCCD1X,=MULTF,
          =SQR17,=RCCD2,=X/Y15,=1/X15,FLIP8,GETCON,
          =DIV120,=SHF10,=AD15s,=DIV100,FUDGE

Alters (INC): A,B,C,D,R0,R1,R2,R3,P,XM,SB,sIX,sCOMP,
              sATAN,sSGN,s+PI/2
Stk lvls:  2

Algorithm:

sIX    (7) : If set, result may be inexact    (INEXACT flag)
sCOMP  (8) : If set, need complementary angle (COMP flag)
sATAN  (6) : If set, need ATAN                (ATAN flag)
sSGN   (10): If set, negate result            (SGN flag)
s+PI/2 (11): If set, need add PI/2            (Add PI/2)

History:

    Date     Programmer            Modification
  --------   ----------   ----------------------------------
   6/07/82   SB           Documented
  10/06/82   SB           Code Pack:  Eliminate proj mode
```

```
+-------------------------------------------------+------------------+
|                                                 |                  |
|  MTHSTK  -  Math Stack Utilities                |  CHAPTER  14     |
|                                                 |                  |
+-------------------------------------------------+------------------+
```

## 14.1 POP2N - Pop 2 Numbers From Stack.

Category: MTHSTK    File: AB&FCN::MS

```
Name:(S) POP2N   -  Pop 2 Numbers From Stack.

Purpose:
     Pop 2 numbers from math stack.

Entry:
     D1=Stack pointer.

Exit:
     DEC mode.
     D1 16 nibbles before end of entry (D1=D1+16 to get to
       next entry.
     If carry clear:
       C[W] = first number on stack.
       A[W] = second number on stack.
     If carry set {one or both numbers complex}:
       C[W]=Real part of first number.
       R2=Imaginary part of first number.
       A[W]=Real part of second number.
       R0=Imaginary part of second number.
       Imaginary part = 0000000000000900 if arg is real.
     Error exit (eDATTY) if either arg not numeric.

Calls:      None.

Uses.......
            A,B[0],C,P.  If Carry Set: R0, R2.

Stk lvls:   0

History:

    Date      Programmer               Modification
  --------   ----------   ----------------------------------
```

| | | |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 14.2 POP1N - Pop 1 Number Off Of Stack

Category: MTHSTK File: AB&FCN::MS

Name:(S) POP1N - Pop 1 Number Off Of Stack

Purpose:
Pop one numeric value off of math stack.

Entry:
D1 = Stack pointer.

Exit:
Errors out (eDATTY) if non-numeric item.
DEC mode.
P=0.
If carry clear: Result real.
Result in A.
If carry set: Result complex.
Real part in A.
Imaginary part in R0.

Calls: None.

Uses.......
A,B[0]. If carry set, R0.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 14.3 REVPOP - REV$ On String And Then POP1S

Category: MTHSTK File: AB&FCN::MS

Name:(S) REVPOP - REV$ On String And Then POP1S

Purpose:
Reverse a string on the stack and then pop it.

Entry:
D1=Mathstack pointer.
HEX mode.

Exit:
A[A]=string length.
D1 pointing at low-address end of string (last char).
P=0.

Calls: REV$, POP1S (falls through).

Uses.......
A,B,C[A],D[A],P,D1

Stk lvls: 2

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 14.4 POP1S - Pop 1 String Arg Off Stack

Category: MTHSTK File: AB&FCN::MS

Name:(S) POP1S - Pop 1 String Arg Off Stack

Purpose:
Position pointers to pop a string argument off of math stack.

Entry:
HEX mode.
D1 pointing at string header in stack.

Exit:
Errors out (Data type) if item on stack is not string.
P=0.
D1 pointing past string header... pointing at last character of string.
A[A]=length of string in nibbles.

Calls: None.

Uses.......
A[W],D1,P

Stk lvls: 0

NOTE:
Does not return if item on stack is not string.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 09/23/83 | MM | Attempted to document |

## 14.5 MPOP2N - Pop 2 Args W/signan Check

Category: MTHSTK File: AB&FCN::MS

Name:(S) MPOP2N - Pop 2 Args W/signan Check
Name:(S) POP2N+ - Pop 2 Args W/signan Check

Purpose:
Pop two arguments off of the math stack and report signaling NANs.
MPOP2N calls uMODES to fetch modes to ST.
POP2N+ assumes this has already been done.

Entry:
D1 = stack pointer.

Exit:
Carry set: One or both numbers are complex. signaling NaN check not done. Same exit conditions as POP2N.
Carry clear: C[W] = first number on stack.
A[W] = second number on stack.
P=0.
D1 pointing 16 nibbles before next stack entry.

Calls: POP2N, SIGTST, URES12, uMODES.

Uses.......
A,B,C,D,R3,S7-S11.

Stk lvls: 3

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 10/14/83 | NM | Attempted to document |

## 14.6 MPOP1N - Pop 1 Arg & Check For Sig NaN

Category: MTHSTK File: AB&FCN::MS

Name:(S) MPOP1N - Pop 1 Arg & Check For Sig NaN
Name:(S) POP1N+ - Pop 1 Arg & Check For Sig NaN

Purpose:
Pop one numeric argument and give Signaled Op message if appropriate.

```
Entry:
     D1=Mathstack pointer
   POP1N+: S8-S11 already set according to modes (uMODES)
           has already been called.

Exit:
     DEC mode.
     Carry set: Result is complex.  Signaling NaN check not
                performed.  Result in A/R0 as per POP1N.

Calls:      uMODES, POP1N, SIGTST, uRES12.

Uses.......
            A,B,C,D,R3,S8-S11.

Stk lvls:   3

History:

  Date      Programmer              Modification
 --------   ----------   ----------------------------------
            SA           Wrote
 10/14/83   NM           Attempted to document
```

## 14.7 REV$ - Reverse Characters In A String On Stack

```
     Category:  MTHSTK     File:  AB&UTL::MS


Name:(S) REV$    -  Reverse Characters In A String On Stack

Purpose:
     Reverse a string on the mathstack.

Entry:
     HEX mode.
     D1 pointing at string header.

Exit:
     D1 pointing at string header.
     String has been reversed.
     C[A]=D[A]=copy of D0.
```

Error exit (eDATTY) if not pointing at string.

Calls: POP1S.

Uses.......
A,B,C,D,P.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 14.8 POPMTH - Skip Past An Item On Mthstk

Category: MTHSTK File: AB&UTL::MS

Name:(S) POPMTH - Skip Past An Item On Mthstk
Name:(S) POPSTR - Skip Past An Item On Mthstk

Purpose:
Skip past current item on the mathstack. Useful for finding a particular item or for counting items.

Entry:
P=0.
POPMTH: D1 at top of mathstack.
POPSTR: D1 pointing past first 2 nibbles of string header at top of mathstack.

Exit:
P=0.
D1 at new top of mathstack.
Carry clear.

Calls: None.

Uses.......

A,C,D1.

Stk lvls: 0

Detail:
Correctly skips past complex numbers and string items.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SC | Wrote |
| 10/18/83 | NM | Attempted to document |

## 14.9 ARGPR+ - Reads modes, pops and norm. real nbr

Category: MTHSTK File: PM&STA::MS

Name:(S) ARGPR+ - Reads modes, pops and norm. real nbr

Purpose:
Reads user modes, pops numeric argument off math stack, tests for array or complex type or signaling NaN, splits and normalizes argument to 15-digit form, detects non-finiteness

Entry:
Numeric argument on top of math stack
D1 points to top of math stack

Exit:
A/B -- 15-digit form of argument
If signaling NaN: Carry=Set, XM=1
Otherwise: Carry=Clear
DECMODE
Fatal error if complex or array data type

Calls: INVNaN,POP1R,SPLITA,umode+

Uses.......
Inclusive: A,B,C(A),D(A),P,SB,XM,s8-11,

unless fatal error

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/26/82 | PM | Documented routine |
| 12/14/82 | " | Added signaling NaN test |
| 01/06/83 | " | Revised documentation |

## 14.10 ARGPRP - Pops and normalizes real number

Category: MTHSTK File: PM&STA::MS

Name:(S) ARGPRP - Pops and normalizes real number

Purpose:
Same as ARGPR+, except that user modes are not read.

Entry:
Same as ARGPR+

Exit:
Same as ARGPR+, except user modes not read.

Calls: INVNaN,POP1R,SPLITA

Uses.......
Inclusive: A,B,C(A),P,XM, unless fatal error

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/26/82 | PM | Documented routine |
| 01/06/83 | " | Revised documentation |

## 14.11 POP1R - Pops real number from math stack

Category: MTHSTK File: PM&STA::MS

**Name:(S) POP1R - Pops real number from math stack**

**Purpose:**
pops numeric argument off the top of the math stack and tests that it is a real data type.

**Entry:**
Numeric argument on top of math stack
D1 points to top of math stack

**Exit:**
A -- has 12-digit form of argument
Carry=clear
DECMODE
fatal error if array or complex data type

**Calls:** POP1N

**Uses.......**
Inclusive: A,B(X),P, unless fatal error

**Stk lvls:** 1

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 08/12/82 | PM | Documented routine |
| 01/06/83 | " | Revised documentation |

## 14.12 ARGSTA - Pops and tests real number

Category: MTHSTK File: PM&STA::MS

Name:(S) ARGSTA - Pops and tests real number
Name:(S) ARGST- - Pops and tests real number

Purpose:
Reads user modes, pops numeric argument off math stack, tests for array or complex type, detects non-finiteness, and tests for NaN.

Entry:
Numeric argument on top of math stack

Exit:
A ---- 12-digit argument from top of stack
Carry=Clear if real finite
Carry=Set if infinity
Fatal error if array, complex, or NaN
DECMODE

Calls: POP1R,finita,umode+

Uses.......
Inclusive: A,B(X),D(A),P
ARGSTA: also SB,XM,s8-11

Stk lvls: 2

NOTE:

| Input | Fatal error message |
|---|---|
| array | "eDATTY" |
| complex | "eDATTY" |
| NaN | "eIVARG" |

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/16/82 | PM | Documented routine |
| 10/06/82 | " | Removed projective infinity test |
| 01/06/83 | " | Revised documentation |

## 14.13 XXHEAD - Remove String Header (Undo ADHEAD)

Category: MTHSTK    File: SB&EXC::MS

Name:(S) XXHEAD - Remove String Header (Undo ADHEAD)

Purpose:
Removes string header from a string on stack. Leaves registers set up so that STKCHR may be called again.

Entry:
P = 0

Exit:
P = 0
D(A)=Pointer to AVMEMS
R1(A)=Pointer to end of stack item (highest address)
D1 points to start of stack item (lowest address)
Carry clear

Calls: POP1S

Uses.......
Inclusive: C(A),D1,D(A)

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/19/82 | B.S. | Added documentation |

## 14.14 ADHEAD - Add String Header

Category: MTHSTK File: SB&IO::MS

Name:(S) ADHEAD - Add String Header

Purpose:
Adds string header to string on stack

Entry:
R1(A)=Start of stack item(hi mem)
D1=End of stack item(low mem)
S0 set iff RTN desired (jumps to EXPR otherwise
D(A)=(AVMEMS)
P=0

Exit:
D1 points at string header on stack

Calls: STKCH+

Uses.......
Exclusive: A(A),C(W),D1
Inclusive: A(A),C(W),D1

Stk lvls: 0

Detail:
R1 should have been used to store stack pointer before putting string on stack. As the string was added to stack, D1 should have been decremented to keep it pointed at the last char of string. This routine can then be used to tack on the string header (F0lllll000000000) where lllll is the length of the string.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/20/82 | B.S. | Updated documentation |

## 14.15 BF2STK - Buffer To Stack

Category: MTHSTK File: SB&IO::MS

Name:(S) BF2STK - Buffer To Stack
Name: BF2ST+ - Buffer To Stack

Purpose:
Pushes a string buffer onto math stack

Entry:
P = 0
S0 = 0 ---> GOTO EXPR when done (don't return)
S0 = 1 ---> Return when done
BF2ST+ pre-clears S0 causing a GOTO EXPR when done
D1 points to stack
D0 should be PC if S0 clear for proper function rtn
C(A) should point to buffer which is a string of
bytes terminated by a FF byte.

Exit:
P = 0
D1 reflects new stack pointer
D0 unchanged

Calls: STKCHR,ADHEAD,D=AVMS

Uses.......
Inclusive: A(A),B(A),C(A),D(A),R1,D0,D1

Stk lvls: 1

Detail:
Buffer is terminated by an FF byte.
Pushes a buffer onto stack a character at a time
and jumps to MEMERR if memory overflows. The result
is a string item on stack with proper header set up.
If S0 is clear the routine assumes that a function
is ending returns directly to EXPR to continue
expression evaluation.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/19/82 | B.S. | Updated documentation |

## 14.16 COLLAP - Collapse Math Stack

Category: MTHSTK File: SG&EXC::MS

Name:(S) COLLAP - Collapse Math Stack

Purpose:
Collapses math stack

Entry:

Exit:
D1 = MTHSTK
C(A)= new value of MTHSTK pointer
Carry clear

Calls: none

Uses.......
C(A),D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/25/82 | S.W. | Created utility |

## 14.17 ERRM$f - Transfer ASCII from AvMem to stack

Category: MTHSTK File: TI&ERD::MS

```
Name:(S) ERRM$f  -  Transfer ASCII from AvMem to stack

Purpose:
     Transfer an ASCII buffer from AvMemSt to Math Stack.

Entry:
     P      = 0
     R3(A)= PC address (from D0)  (see R3=D10)
     R3(9-5)= stack address (from D1)  (see R3=D10)
     D0 points to ASCII buffer. ASCII string ends
       in FF byte. (D0 must be less than FORSTK pointer.)
     B(A) points to terminator FF byte

Exit:
     P      = 0
     D1 = new stack pointer
     String on stack
     D0 = address passed in R3(A)
     Will jump to MEMERR if insufficient memory.

Calls:  D1C=R3, BF2ST+

Uses.......
 Exclusive: B(A)
 Inclusive: A(W),B(A),C(A),D(A),R1,D1

Stk lvls:   1

NOTE:
     See ERRM$ heading for that entry point.

Algorithm:
     D1 and D0 are restored from R3.
     Before calling BF2ST+, which moves the message from
       AvMem to the math stack, checks whether total
       available memory is at least twice as large as the
       length of the string (since copying it to the
       stack would otherwise overwrite the tail end of
       of the string).  If not, MEMERR.
     Exits through BF2ST+: buffer to math stack.

History:

  Date      Programmer              Modification
 --------   ----------   ----------------------------------
 09/14/82   MB           Documentation
```

```
+---------------------------------------+------------------+
|                                       |                  |
|  MTHUTL  -  System Level Math Utilities |  CHAPTER  15   |
|                                       |                  |
+---------------------------------------+------------------+
```

## 15.1 REDUCE - Parse And Execute Partial ExpresSIONS

Category: MTHUTL     File: AB&CLC::MS

Name:(S) REDUCE - Parse And Execute Partial ExpresSIONS

Purpose:
Parse and execute partial expressions in calc mode.

Entry:
P = 0

Exit:
P = 0

Calls: NTOKEN, RANGE, MEMBER, PUSH, BLDCON, NRMCON, STAKUP, STAKDN, FNARG, ARYARG, ARGLMT, PUSH11, INSRTO, ORIGIN, SKPARG, PARPRP, COMPIL, ARGCNT, PRCDNC, CLCEXP, CLCBTS, STKBAK

Uses....... Everything

Stk lvls: 6

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/13/83 | SA | Added documentation |
| 08/03/83 | SA | Static fix to Bug 9597. Packable BSS 3 created below label SO-30. |

## 15.2 NRMCON - Convert BLDCON Constant into Usable Form

Category: MTHUTL File: AB&CLC::MS

**Name:(S) NRMCON - Convert BLDCON Constant into Usable Form**

**Purpose:**
Converts a 12-digit constant built by BLDCON into a nice normalized number taking into account overflow and underflow with appropriate trap settings.

**Entry:**
Exit conditions of BLDCON.

**Exit:**
A = 12-digit normalized number.
XM=0 iff number ok (no overflow or underflow)
May generate warning message if XM=1.

**Calls:** SFLAGS, MFWRNQ

**Uses.......**
A-D,D0,D1,R0,P

**Stk lvls:** 3

**History:**

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |
| 12/16/83 | FH | Added more documentation, changed name from GRONK to NRMCON, made a supported entry point |

## 15.3 BLDCON - Build A Constant For CalC MODE

Category: MTHUTL File: AB&CLC::MS

Name:(S) BLDCON - Build A Constant For CalC MODE

Purpose:
Build a constant for calc mode.

Entry:
Exit conditions of NUMSCN.

Exit:
```
   If XM = 0:  (no Overflow or Underflow)
     B      =  Normalized unsigned 12-digit number.
   If XM = 1:  (Overflow or Underflow occurred)
     B(B)   =  Token indicating overflow (=tBIG) or
               underflow (=tSMALL).
```

Calls: None.

Uses.......
A,B,C, XM.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |
| 12/16/83 | FH | Added more documentation |

## 15.4 READIN - Read Something In

Category: MTHUTL File: AB&EXP::MS

Name:(S) READIN - Read Something In

Purpose:
Probably.

Entry:
Unclear.

Exit:
Unclear.

Calls: None.

Uses.......
D,P,C[S].

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |

## 15.5 RSTST - Restore Status Bits

Category: MTHUTL File: AB&EXP::MS

Name:(S) RSTST - Restore Status Bits

Purpose:
Restore status bits saved in STSAVE.

Entry:
None.

Exit:
Status bits restored.
Carry clear.

```
Calls:     None.

Uses.......
           A[A],C[X].

Stk lvls:  0

History:

  Date     Programmer              Modification
--------   ----------   ----------------------------------
           SA           Wrote
11/01/83   NM           Added documentation
```

## 15.6 SMALL - Create Special Consts

Category: MTHUTL    File: AB&FCN::MS

```
Name:    SMALL  - Create Special Consts
Name:(S) BIG    - Create Special Consts
Name:    BIG+   - Create Special Consts
Name:(S) HUGE   - Create Special Consts

Purpose:
     Create constants MAXREAL, INF, EPS.

Entry:

Exit:
  SMALL: C[W] = EPS.
         Mode unchanged.
         P=14.
  BIG:   C[W] =+/-9.99999999999E499 (sign preserved from
                                      entry).
         DEC mode.
  BIG+:  C[W] = 9.99999999999E499.
         DEC mode.
  HUGE:  C[W] = 09999999999999F00 (infinity).

Calls:     None.
```

Uses.......
C. SMALL uses P.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 15.7 SIGCHK - Report Signaling NaN

Category: MTHUTL File: AB&FCN::MS

Name:(S) SIGCHK - Report Signaling NaN

Purpose:
Check for signaling NaN and report "Signaled Op" if found.

Entry:
Number in A.
DEC mode.

Exit:
Number in A.
Carry clear.

Calls: uRES12, SIGTST.

Uses.......
A-D,P,R3,S7-S11.

Stk lvls: 3

History:

| Date | Programmer | Modification |
|---|---|---|

| | | |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Attempted to document |

## 15.8 RND-12 - Round A 12-digit Fp Number

Category: MTHUTL File: AB&UTL::MS

Name:(S) RND-12 - Round A 12-digit Fp Number

Purpose:
    Round of a floating-point number at specified digit.

Entry:
    A = number (12-digit floating-point).
    P points to digit where rounding is to take place. See detail, below.

Exit:
    P=0.
    A=Rounded (not IEEE-rounded) 12-digit form.
    If P=15 on entry, no rounding was done.
    Carry set iff rounding overflowed (returns MAXREAL).

Calls: None.

Uses.......
    A,B,P.

Stk lvls: 0

Detail:
    Typically called after IF12A, which sets P to point at the first fractional digit.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/17/83 | NM | Attempted to document |

## 15.9 A-MULT - Multiply Two 20-bit Hex Integers

Category: MTHUTL File: AB&UTL::MS

Name:(S) A-MULT - Multiply Two 20-bit Hex Integers

Purpose:
Multiply two 20-bit hex integers.

Entry:
A[A], C[A] are operands.

Exit:
P preserved.
A[A]=product.
Carry set if no problem.
Carry clear -> overflow. Returns FFFFF.

Calls: None.

Uses.......
A[A],B[A],C[A],C[14].

Stk lvls: 0

| Date | Programmer | Modification |
|---|---|---|
| | SA | Created |
| 10/18/83 | NM | Attempted to document |

## 15.10 SHF10 - Shift to normalize

Category: MTHUTL File: JT&MTH::MS

Name:(S) SHF10 - Shift to normalize

Purpose:
Normalize 15 form in AB.

Entry:
Finite (possibly denormalized no.) in AB

Exit:
AB is normalized (clean 0s). P=C(S), C(S)=B(S), B(S)=0
Carry clear.

Calls: None

Uses.......
Inclusive: C(S) (see exit conditions)

Stk lvls: 0

## 15.11 SQR70 - Set SB according to Reg C

Category: MTHUTL File: JT&MTH::MS

Name:(S) SQR70 - Set SB according to Reg C

Purpose:
To set or clear Sticky Bit (SB) for C#0 or C=0 resp.

Entry:
C=0 if SB=1 is desired, else C#0

Exit:
SB=0 if C=0, else SB=1.
Carry Clear.

Calls: (none)

Uses.......
Inclusive: C(A)

Stk lvls: 0

Algorithm:

```
     =SQR70   SB=0
              ?C=0     W
              GOYES    SQR80
              C=C-1    X
              CSR      X
      SQR80   RTNCC
```

## 15.12 INF*0 - Inf*0 exception

Category: MTHUTL     File: JT&MTH::MS

Name:(S) INF*0 - Inf*0 exception

Purpose:
To create a 15-form NaN result with Inf*0 msg code.

Entry:
No conditions.

```
CODE:  =INF*0  P=     0
               LC(2)  =eIF*ZR
               GOTO   INVNaN
```

Exit:
(See INVNaN)

Calls: Goes to INVNaN

Uses.......
Inclusive: P; A,B,C(A); HD.ST.[XM,SB]

Stk lvls: 0

## 15.13 XYEX - EXCHANGE X & Y

Category: MTHUTL File: JT&MTH::MS

Name:(S) XYEX - EXCHANGE X & Y

Purpose:
To exchange the internal nos. Y=(A,B) & X=(C,D) .

Entry:
(A,B)=Y & (C,D)=X

Exit:
(A,B)=X & (C,D)=Y
Does not alter carry

Calls: (none)

Uses.......
Inclusive: A,B,C,D

Stk lvls: 0

Detail:
Swaps entire regs (A with C and B with D)

## 15.14 SPLITA - SPLIT A

Category: MTHUTL File: JT&MTH::MS

Name:(S) SPLITA - SPLIT A

Purpose:
To convert an external (12 dig.) form into an internal (15 dig.) form

Entry:
A=x' (external no.)

Exit:
(A,B)=x (normal internal form of x')
CR.set => excptn'l operand (i.e. NaN or Inf )
CR.clr => finite operand
--see DETAIL below for normal internal form defn.

Calls: (none)

Uses.......
Inclusive: A,B (actually B[14..5] ends up in A[14..5] )

Stk lvls: 0

Detail:
DEFN: The "normal internal form" of
1) NaN is A(A)=00F01 & B(XS)=F (i.e.mant#0).
2) Inf is A(A)=00F00 & " .
3) finite no is a normalized no.(no denorm.).

## 15.15 CLRFRC - Clear fractional part

Category: MTHUTL File: JT&MTH::MS

Name:(S) CLRFRC - Clear fractional part

Purpose:
Clears fractional part of quantity in A/B, preserving the sign of the argument. Returns the result in A/B. Carry set if no fractional part.

Entry:
A/B -- 15-digit form of quantity

Exit:
A/B -- quantity with fractional part cleared
Carry=set if no fractional part
Carry=clear otherwise
DECMODE

Calls: INFR15

Uses.......
Inclusive: A(A),B,C(A),P

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/24/82 | PM | Documented routine |
| 09/23/82 | SB | Packed out =CLFRCF Entry |
| 12/02/82 | JT | Corrected docum. for stk lvls. (INFR15 calls FINITA now) |

## 15.16 IF12A - Integer/Fraction Split

Category: MTHUTL File: JT&MTH::MS

Name:(S) IF12A - Integer/Fraction Split
Name:(S) INFR15 - Integer/Fraction Split

Purpose: Find decimal (used by INT15 & FRAC15). Returns position of decimal encoded in P (see below).

Entry: Standard Math - 12 dig: IF12A, 15 dig: INFR15

Exit: Encoded location of decimal in P.

Alters:
IF12A: A,B,C[A],P,CARRY
INFR15: C[A],P,CARRY

Stk Lvls: 1

Note:

| ARGUMENT | RETURN (P) | [Notation: EXP(X)=E] |
|---|---|---|
| NaN or INF | 15 | |
| Standard 0 | 13 (standard 0 has E=0) | |

| | |
|---|---|
| E<0 | 14 |
| 0<=E<=13 | 13-E |
| 13<E | 15 |

Note: If the Expon=14 (i.e. a 15 digit integer) then C(A) is 0. If Expon>14 (but finite) then C(A)=50000 on exit This is used in YX15 to determine if x is an even integer.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/23/82 | SB | 15-dig entry: P=15 for NaN or INF, Comments, description update, Standard header. |

## 15.17 SPLTAC - Split & normalize A & C

Category: MTHUTL File: JT&MTH::MS

Name:(S) SPLTAC - Split & normalize A & C

Purpose: Split & Normalize values in A & C.

Entry: A:X C:Y [12-digit forms]

Exit: A,B:X C,D:Y [15-digit forms]

Calls: SPLITA,SPLITC

Alters (INC): A,B,C,D,Carry

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|

| | | |
|---|---|---|
| 6/28/82 | SB | A field instead of W |
| 9/23/82 | SB | This routine moved (eliminate GOTO) |

## 15.18 SPLITC - SPLIT C

Category: MTHUTL File: JT&MTH::MS

Name:(S) SPLITC - SPLIT C

Purpose:
see SPLITA

Entry:
C=x' (external form)

Exit:
(C,D)=x (normal internal form)

Calls: (none)

Uses.......
Inclusive: C,D

Stk lvls: 0

Detail:
see SPLITA

## 15.19 uRES12 - User Result

Category: MTHUTL File: JT&MTH::MS

Name:(S) uRES12 - User Result
Name:(S) uRESNX - User Result (non exceptional)
Name:(S) uRESXT - User Result for exact results

Purpose:
To pack the 15-form input into a 12-form result for delivery to the user. This includes rounding according to the user's mode, checking for xcpt'ns & consulting relevent trap values, setting the xcptn flags, and sending off any warning messages or errors. The external default result (12 form) is returned in reg C.

Entry:
1.(A,B)&SB contain x (the unpacked result)
2.XM is set if x is the result of an xcpt (DVZ or IVL)
if XM=1 then P=(DZP,IVP or TYPO^0) tells which xcptn and C(A)=msg code (for specific xcptn e.g. 0/0,LOG(0),etc.)
**Note**: DZP=3; IVP=4; TYPO^0=14.

3.D1=top math stk -- only used for a wrn. msg., to check avail.mem. for a possible mem err.

```
CODE:   =uRESNX  GOSUB  uRND>P
----    =uRESXT  GOSUB  HTRAP
                 GOSUB  HNDLFL
                 GOTO   MESSG
```

Exit:
C:=x' (the 12digit packed result).
The XCPTN flags are set and any messages have been displayed (including errors).

Calls: uRND12, HTRAP, HNDLFL, MESSG

Uses.......
Inclusive: P; A,B,C,D; R regs [3]; ST.[7..11];
HD.ST.[SB,XM];

Stk lvls: 2 (provided that MFWRNQ uses <= 4 levs.)

NOTE:
TYPO^0 "xcptns" (0^0 & Inf^0) return 1. They are not IVL xcptns but do consult the IVL trap. No flags are raised, but TRAP(IVL)#0 gives a wrn'g while =0 gives an error. XM=1 & P=14 signals TYPO^0 "xcptn".

## 15.20 uRND>P - user ROUND

Category: MTHUTL    File: JT&MTH::MS

```
Name:(S) uRND>P  -  user ROUND
Name:(S) RND12+  -  Round 15-form
Name:(S) OVFL    -  Create overflow value

Purpose:
     To round an internal no. x to external form, according
      to the user's rounding mode.
     RND12+: Round according to status bits (s10,s11). Given
      by (0,0)=NEAR, (0,1)=ZERO, (1,0)=POS, (1,1)=NEG.

Entry:
     (A,B)&SB= x
     P=rounding position (e.g. P=2 for 12dig.;P=9 for 5dig.)
       2 <= P <= 13
     DEC MODE

Exit:
     C:= x' (rounded external form)
     sIX(s7):= inexact info.
     P:=OVP(2),UNP(1) or OKP(0)  (ovfl,unfl or ok)
     B(A)=msg code of OV or UN resp.

Calls:       NRMLAB, RNDNRM, BIASC+,-, BIASA-, HUGE20, BIG,
              uMODES

Uses.......
 Inclusive: P; A,B,C,D; R regs [3]; ST.[7(sIX),8..11];
            HD.ST.[SB];

Stk lvls:   1

NOTE:
     Original x is not always preserved !
     An inexact +/- 0 (i.e. SB=1) will be rounded to +/- 0
      with P=OKP and sIX(s7)=1 on exit.
```

## 15.21 RNDNRM - Round a Normal Number

Category: MTHUTL    File: JT&MTH::MS

**Name:(S) RNDNRM - Round a Normal Number**

**Purpose:**
To round the mantissa of a finite internal no. x, according to the rounding modes specified.

**Entry:**
(A,B)&SB = x
P=rounding position (e.g. P=2 for 12 digit round; P=9 for 5 digit round) 0<=P<=14
sINFRD(s10)&sNEGRD(s11) set for rounding mode (see =uMODES)
DECMODE

**Exit:**
(C,D) = x' (rounded value) (and D[S]=0)
sIX(s7) set iff the rounded result is inexact
P=0

**Calls:** None

**Uses.......**
Inclusive: P; C,D; ST.[sIX(s7)];

**Stk lvls:** 0

**NOTE:**
With an input of inex 0 in Rnd to Inf mode, the mantissa is rounded to 00...01 and its exponent is unchanged. In the other rounding modes the mantissa remains 0.

## 15.22 HTRAP - HANDLE TRAPS

Category: MTHUTL File: JT&MTH::MS

Name:(S) HTRAP - HANDLE TRAPS

Purpose:
To determine any trapping action that is specified (e.g. alter the IEEE default result or halt) on an xcptn .

Entry:
C= x' (the 12 form IEEE default result)
Trap to be checked is indicated by:
P=<xcpt> {OK,UN,OV,DZ or IV}
sIX(s7)= xact/inex info (esp. for P=OK)
B(A)=msg code (for UN,OV,DZ or IV only)

Exit:
C:= x'' (revised result --after consulting traps)
B(S):= 0 for an error (HALT) ; 9 otherwise (continue)
B(A)=msg code for IX,UN,OV,DZ or IV.
P = updated xcpt.
sIX(s7) reflects updated exact/inexact info
Sets DECMODE (only when GOSUB BIG is executed)

Special exit condition:
(preserved for USGOVF -- DISP USING OVFL)

Whenever HTRAP exits with B(S)=0 (i.e. Halt) then:

1) If TRAP(OVF) caused the Halt then s7 (sIX) is NOT altered from its entry state.

2) If TRAP(INX) caused the Halt then s7 (sIX) is set to 1 on exit.

Uses.......
Inclusive: A,B,C,D; ST.[sIX(s7)];

Calls: BIG

Stk lvls: 1

NOTE:

## 15.23 HNDLFL - HANDLE FLAG SETTING

Category: MTHUTL File: JT&MTH::MS

Name:(S) HNDLFL - HANDLE FLAG SETTING

Purpose:
To set user's xcptn flags (all at once).

Entry:
P=<xcpt>, <xcpt> in {OK,UN,OV,DZ,IV}.
sIX(s7)= inex info.

Exit:
user's xcptn flags have been updated.
D(X) will contain bit mask of xcptns set (b11 to b7
represents IV,DZ,OV,UN,IX)

Uses.......
Inclusive: A(A),D(X); R regs [3];

Stk lvls: 0

NOTE:
The info. from HTRAP [C,B(S),B(A),P & sIX] is preserved.

## 15.24 MESSG - MESSAGE

Category: MTHUTL File: JT&MTH::MS

Name:(S) MESSG - MESSAGE

Purpose:
To display a warning message without disturbing most of the CPU or Math Scratch Stack. It uses available memory instead, to preserve C,R0,R1,R2,R4, D0,D1, Status Bits, Math Scratch (=SCRST0) and RSTK levels.

Entry:
1) B(A)=msg code; B(W) used if msg has text insertion (see MFWRNQ).
2) B(S)= 0 for error
   = 9 otherwise
3) If B(S)=9 then
   P=0 ==> no msg (used to supress msg )
   P#0 ==> put out warning msg

4) D1=top of math stk (end of available memory)
   -- used only for mem chk when a warning is sent out.

Exit:
Displays warn/err msg & rtns to main driver on an err.

Calls: MFWRNQ or exits thru BSERR, CHKmem, SNAPLC, MOVEU3, MOVED3, SNAPR* .

Uses.......
Inclusive: P; A,B,D;R3; (unless an error occurs--BSERR)
The Math Scratch Area is saved to Available Memory since the display routines check Service Request and an Alarm calculation uses math scr.

Stk lvls: 2 1+[Levels(MFWRNQ) - 2(saved Levels)]

## 15.25 FINITA - Is (A,B) non-finite ?

Category: MTHUTL File: JT&MTH::MS

Name:(S) FINITA - Is (A,B) non-finite ?

Name:(S) FINITC - Is (C,D) non-finite ?

Purpose:
    To test for finite arguments.

Entry:
    FINITA: 15-form in AB
    FINITC: 15-form in CD

Exit:
    DEC Mode
    Carry Set indicates non-finite
    Carry Clear indicates finite

Calls: (None)

Uses.......
 Inclusive: Nothing

Stk lvls: 0

## 15.26 FNPWDS - Weed out NaNs and Infs

Category: MTHUTL File: JT&MTH::MS

Name:(S) FNPWDS - Weed out NaNs and Infs

Purpose:
    To handle NaN and Inf as arguments to functions.

Entry:
    AB=x

Exit:
```
    If x is
            1) finite   ==> RTNCC
            2) inf      ==> RTNSC
            3) NaN      ==> abort call'g fn (C=RSTK)
                            RTN with x (input NaN)
```

DEC Mode

Calls: FINITA

Uses.......
Inclusive: C(A)

Stk lvls: 0 (Uses C(A) to save the level.)

## 15.27 STAB1 - Store AB into scratch 1

Category: MTHUTL File: JT&MTH::MS

Name:(S) STAB1 - Store AB into scratch 1
Name:(S) EXAB1 - Exchange AB with scratch 1
Name:(S) RCCD1 - Recall CD into scratch 1
Name:(S) STAB2 - Store AB into scratch 2
Name:(S) EXAB2 - Exchange AB with scratch 2
Name:(S) RCCD2 - Recall CD into scratch 2
Name:(S) STCD2 - Store CD into scratch 2

Purpose:
To use R0-R3 as scratch space for 15-form numbers.

Entry:
Either AB or CD has a 15-form to be transfered with (R0,R1) or (R2,R3).

Exit:
Data transfer has taken place.

Calls: (none)

Uses.......
Inclusive: nothing

Stk lvls: 0

## 15.28 IDIVA - A-field Integer Divide

Category: MTHUTL    File: MN&UTL::MS

Name:(S) IDIVA - A-field Integer Divide

Purpose:
Compute A/C, A mod C.

Entry:
HEX or DEC mode according to arguments.
Dividend in A[A], divisor in C[A].

Exit:
Quotient in A[W].
Remainder in B[W],C[W].
Mode preserved
P=15.
Carry clear.

Calls: IDIV (falls through).

Uses.......
A,B,C,P

Stk lvls: 0

Algorithm:
Zero out nibs 5-15 of A and C.
IDIV.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/22/82 | NM | Added documentation |

## 15.29 IDIV - Full Word Integer Divide.

Category: MTHUTL File: MN&UTL::MS

Name:(S) IDIV - Full Word Integer Divide.

Purpose:
Perform HEX or DEC integer divide.

Entry:
HEX or DEC mode according to arguments.
Dividend in A.
Divisor in C.

Exit:
Quotient in A.
Remainder in B and C.
Mode preserved.
P=15.
Carry clear.

Calls: None.

Uses.......
A,B,C,P.

Stk lvls: 0

NOTE:
No provision is made if called with denominator = 0.
This code will get stuck in an infinite loop. CAVEAT
EMPTOR.

Algorithm:
```
     Align divisor with dividend, with P pointing at 1's
       digit of divisor.
     Divisor to B.  Clear A for result.
  1: While B>C do begin B=B-C W, A=A+1 P end
     CSR W, P=P-1.  If P wasn't zero, goto 1.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/20/82 | NM | Added documentation |

## 15.30 MPY - HEX * HEX Or HEX * DEC Multiply.

Category: MTHUTL File: MN&UTL::MS

Name:(S) MPY - HEX * HEX Or HEX * DEC Multiply.

Purpose:
Perform HEX mode or mixed mode full word multiply.

Entry:
If HEX * HEX multiply:
Mode = HEX.
Arguments in A and C.

If HEX * DEC multiply:
Mode = DEC.
Hex argument in C.
Dec argument in A.

Exit:
If HEX * HEX multiply: HEX result in A,B,C.
If HEX * DEC multiply: DEC result in A,B,C.
Mode preserved.
Carry clear.
P unaffected.

Calls: None.

Uses.......
A,B,C.

Stk lvls: 0

NOTE:
This routine provides a handy HEX to DEC conversion.
Performing a mixed-mode multiply with the hex argument
in C and a 0000000000000001 in A produces a DEC result
in C.

Algorithm:
Clear result (B).
1: CSRB.

```
      If low bit was clear, goto 2.
      Add A to result.
   2: Double A.
      If C#0 goto 1.
      Copy result to A and C.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/20/82 | NM | Added documentation |
| 10/15/82 | SA | Leaves result in A also. |

## 15.31 RNDAHX - Pops, tests,rounds,converts dec to hex

Category: MTHUTL File: PM&FLG::MS

Name:(S) RNDAHX - Pops, tests,rounds,converts dec to hex

Purpose:
Pops, tests, rounds, and converts a real number
to hex integer.

Entry:
number to be rounded and converted on top of
math stack

Exit:
A(A) -- rounded hex integer
Carry=Clear: negative integer
Carry=Set: nonnegative integer (incl -0)
fatal error if array or complex type, or NaN
HEXMODE
XM=0
P=0

Calls: ARGST-,DCHXF

Uses.......
Inclusive: A,B(S,A),C(A),D(A),P,SB,XM unless fatal error

Stk lvls: 3

NOTE:

| Input | Fatal Error Message |
|---|---|
| array | "eDATTY" |
| complex | "eDATTY" |
| NaN | "eIVARG" |
| conversion overflow | "eIVARG" |

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/11/82 | PM | Documented routine |
| 08/11/82 | PM | Redefined fatal error exits |
| 12/17/82 | PM | Fatal error for convers. ovfl. |
| 02/25/83 | PM | Removed unnecessary GOC |

## 15.32 SB15S - 15-digit subtract/add routine

Category: MTHUTL File: PM&STA::MS

Name:(S) SB15S - 15-digit subtract/add routine
Name:(S) AD15S - 15-digit subtract/add routine

Purpose:
Subtracts or adds, respectively, two 15-digit forms while preserving the meaning of SB to denote an inexact chain calculation.

Entry:
A/B,C/D -- standard floating point math inputs
SB,XM ---- indicate prior inexact or invalid operation

Exit:
A/B ------ standard floating point math outputs
SB,XM ----
Carry set iff XM=1 on exit (e.g., Inf-Inf NaN created)

Calls: AD15s,SAVESB,ORSB

exits through XMTEST

Uses.......
Inclusive: A,B,C,D,P,SB,XM,sIX(s7)

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/21/82 | PM | Documented routine |
| 01/31/83 | " | Handled arithmetic-created NaNs |

## 15.33 uRESD1 - Variation of uRES12

Category: MTHUTL File: PM&STA::MS

Name:(S) uRESD1 - Variation of uRES12

Purpose:
Similar to uRES12. Any XM exception is considered an invalid operation (not a divide by zero or 0^0 type). AVMEME, rather than D1, points to the end of available memory. Also, various entities are initialized on exit.

Entry:
A/B ----- 15-digit form for rounding, trap handling
XM ------ set iff invalid operation has occured
SB ------ set iff result is inexact
AVMEME -- points to end of available memory

Exit:
C ------- contains result
SB,XM = 0
s8-11 --- user modes
HEXMODE
Carry=Clear
P=14

Calls: ures12
exits through uMODES

Uses.......
Inclusive:
CPU: A,B,C,D,P,R3,SB,XM,sIX(s7),s8-11
RAM: STMTD1

Stk lvls: 3 = 1+uRES12

Note:
This routine may fail to properly display a warning message if the message involves text insertion.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/10/82 | PM | Documented routine |
| 11/17/82 | " | D1 preserved, falls through umode+ |
| 01/28/83 | " | Fatal errors halt execution immediately |

## 15.34 GETSA - Tests current statistical array

Category: MTHUTL File: PM&STA::MS

Name:(S) GETSA - Tests current statistical array
Name: GETSDO - Tests current statistical array

Purpose:
Gets the starting address of the current statistical array, to record and test the number of variables, and to test the length of this array. GETSDO does the same after saving DO in function scratch.

Entry:
Current statistical array name stored at =STATAR

Exit:
Carry=Clear:

```
        A(S) --- # variables <=F
        A(A) --- address of first element of current-stat array
        B(X) --- depend var#, independ var#, #variables
        C(A) --- number of elements in current stat array
        R2(S) -- same as A(S)
        R2(A) -- same as A(A)
        D0 ----- same as A(A)
        P=0
        HEXMODE
        F-R0-0 - original D0 if GETSD0
      otherwise: Fatal error
```

Calls:      ADRS50,B=DTOA,B=STAN
            GETSD0:  also SAVD0

Uses.......
 Inclusive: A,B,C(6-0),D(A),P,SB,R2,D0
            GETSD0: also F-R0-0

Stk lvls:   2

NOTE:
Fatal error if there is no current statistical array, or if the current statistical array is invalid.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/01/82 | PM | Documented routine |
| 06/25/82 | " | Replaced fatal errors with NaN's |
| 03/24/83 | " | Replaced NaN's with fatal errors |

## 15.35 SPLTAX - Split, normalize A; handle signal NaN

Category: MTHUTL     File: PM&STA::MS

Name:(S) SPLTAX - Split, normalize A; handle signal NaN
Name:(S) SIGTST - Handle signal NaN

Purpose:

SPLTAX: Splits and normalizes contents of register A, then ...
SIGTST: Test for a signalling Nan and replace such a NaN by a quiet NaN.

Entry:
SPLTAX:
A --- argument in 12 digit form
SIGTST:
A/B = 15 digit form to be tested

Exit:
A/B -- split and normalized argument
Carry=Set:
Signaling NaN replaced by a quiet NaN
XM=1
Carry=Clear, XM preserved, otherwise

Calls: INVNaN, SPLITA
may exit through invnan

Uses.......
Inclusive: A,B,C(A),P,XM

Stk lvls: 1

Note: Foreign NaNs are treated as signaling NaNs.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/21/82 | PM | Documented routine |
| 01/14/83 | " | Revised documentation |

## 15.36 STSCR - Push 15-Form Onto Math Scratch Stack

Category: MTHUTL File: PM&STA::MS

Name:(S) STSCR - Push 15-Form Onto Math Scratch Stack

Purpose:
    Pushes a 15-digit form onto top of math scratch stack

Entry:
    A(S) ---- sign
    A(A) ---- exponent
    B(14-0) - mantissa

Exit:
    P = 1
    Carry=Clear

Calls: GEXPAD,GSCPTR

Uses.......
Inclusive: C,D0,P

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| ??/??/82 | BS | Wrote and coded routines |
| 12/07/82 | PM | Packed and documented routines |
| 01/06/83 | " | Reviewed documentation |

## 15.37 RCSCR - Pop 15-Form From Math Scratch Stack

Category: MTHUTL    File: PM&STA::MS

Name:(S) RCSCR - Pop 15-Form From Math Scratch Stack

Purpose:
    Pops a 15-digit form from scratch stack

Entry:

Exit:
    C(S) ---- sign
    C(A) ---- exponent

D(14-0) - mantissa
A/B ----- unchanged
Carry=Clear
P = 1

Calls: GEXPAD,GSCPTR

Uses.......
Inclusive: C,D,D0,P

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| ??/??/82 | BS | Wrote and coded routines |
| 12/07/82 | PM | Packed and documented routines |
| 01/06/83 | " | Reviewed documentation |

## 15.38 RCLW1 - Recall 1st (Top) Math Scrtch Stack Entry

Category: MTHUTL File: PM&STA::MS

Name:(S) RCLW1 - Recall 1st (Top) Math Scrtch Stack Entry
Name:(S) RCLW2 - Recall 2nd Math Scratch Stack Entry
Name:(S) RCLW3 - Recall 3rd Math Scratch Stack Entry
Name: RCLW4 - Recall 4th Math Scratch Stack Entry
Name:(S) RCL* - Recall Selected Math Scratch Stack Entry

Purpose:
Move the 15-digit form in A/B to C/D and then recall the requested math scratch stack entry in A/B without removing that entry from the stack.

Entry:
(A,B) = 15-form number
RCL*:
P = 0 for 1st entry on math scratch stack
= n-1 for nth entry on math scratch stack

Exit:
    (A,B) = 15-form number from match scratch stack
    (C,D) = (A,B) on entry
    P = 1
    DECMODE
    Carry = Clear

Calls: GEXPAD,GSCPTR

Uses.......
 Inclusive: A,B,C,D,D0,P

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| ??/??/82 | BS | Wrote and coded routines |
| 12/07/82 | PM | Packed and documented routines |
| 01/06/83 | PM | Reviewed documentation |

## 15.39 STKCHR - Add a Character to a Stack Item

Category: MTHUTL File: SB&IO::MS

Name:(S) STKCHR - Add a Character to a Stack Item
Name: STKCH+ - Add a Character to a Stack Item

Purpose:
    Decrements stack pointer, checking av mem to be sure enough room exists. Character C(B) is then written to memory. STKCH+ is same except doesn't move stack pointer first.

Entry:
    C(B)=Character to be appended to stack
    D(A)=(AVMEMS)
    D1 points to stack

Exit:

Exits to MFERR with eMEM error if not enough room
D1 points to new stack character
Carry clear.

Calls: None

Uses.......
Inclusive: D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/20/82 | B.S. | Updated documentation |

## 15.40 TST12A - Compare numbers: 12-Digit arg's A,C

Category: MTHUTL File: SM&MTH::MS

Name:(S) TST12A - Compare numbers: 12-Digit arg's A,C
Name:(S) TST15 - Compare numbers: 15-Digit arg's A/B, C/D

Purpose: Determine relationship between numbers a & c.

Entry: TST12A: 12-digit arg's in A & C.
TST15 : 15-digit arg's in A&B and C&D.
P encodes predicate.

Exit: Carry set=TRUE, P has the cell# associated with the number pair, arg's in 15-dig form unchanged.

Calls: SPLTB,AFIN,CFIN,BIASA+,BIASC+,
BIASA-, BIASC-

Alters (INC): P,A,B,C,D,CARRY
Stk lvls: 1

NOTE: Predicate (INPUT) & Cell# (OUTPUT) Table

| Pred | 9-bias | P |
|---|---|---|
| < | 0001 | 1 |
| = | 0010 | 2 |
| <= | 0011 | 3 |
| > | 0100 | 4 |
| <> | 0101 | 5 |
| >= | 0110 | 6 |
| ? | 1000 | 8 |
| <? | 1001 | 9 |
| =? | 1010 | 10 |
| >? | 1100 | 12 |
| # | 1101 | 13 |

| Cell | Cell# | P |
|---|---|---|
| a<c | 0001 | 1 |
| a=c | 0010 | 2 |
| a>c | 0100 | 4 |
| a?c | 1000 | 8 |

["?" = Unordered]

(Pred is 9-bias of the system token)

Algorithm: Direct comparison of S,EXP, & MANTISSA.
History:

| Date | Programmer | Modification |
|---|---|---|
| 07/12/82 | SB | Documented |
| 10/06/82 | SB | Code Pack: Eliminate Proj Mode |
| 02/09/83 | SB | Code Pack: Consolidate a=NaN tests |
| 02/25/83 | SB | Code Pack: Eliminate GOTO LOGIC. |

## 15.41 BIASA+ - Add Exp bias to A

Category: MTHUTL File: SM&MTH::MS

Name:(S) BIASA+ - Add Exp bias to A
Name: BIASA- - Remove Exp bias from A
Name:(S) BIASC+ - Add Exp bias to C
Name: BIASC- - Remove Exp bias from C

Purpose: Add (or remove) EXP bias [50000] to 15-dig Num

Entry: 15-digit number in A&B or C&D, DEC Mode.

Exit: Unbiased or biased exponent, P=4.

Uses (INC): P, and A[A] (or C[A])

Stk lvls: 0

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 02/10/83 | SB | Removed =BIASAC entry. |

NOTE: BIASA+ = BIASA- (EXP+50000+50000=EXP)

## 15.42 MSN12 - Find most significant NaN, 12-Dig arg's

Category: MTHUTL File: SM&MTH::MS

Name:(S) MSN12 - Find most significant NaN, 12-Dig arg's
Name:(S) MSN15 - Find most significant NaN, 15-Dig arg's

Purpose: For 2-arg functions return most significant NaN.

Entry: [A,B]: x [C,D]: y (15-digit forms)

Exit: CC - Neither x nor y is Nan, reg's not altered.
CS - [A,B] has most significant NaN.

Calls: SPLTB,AFIN,CFIN,=TWONaN,SWAPXY.

Alters: Carry. If exit CS, also registers A,B,C,D.

Stack lvls: 1

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 9/23/82 | SB | Name change and 12-digit entry |
| 10/04/82 | SB | Code pack - Change near IX |

## 15.43 CLASSA - Classification of numeric arg

Category: MTHUTL File: SM&MTH::MS

Name:(S) CLASSA - Classification of numeric arg

Purpose: User classification of numeric argument

Entry: 12-digit argument x in A.

Exit: 12-digit y=CLASS(x) in C; -6<=y<=6

Calls: AFIN,MAKE1

Alters (INC): A,C,P,CARRY

Stk lvls: 1

Detail: Sign(y) = Sign(x)
Mag(y) = 1,2,3,4,5,6 (below)

| x | MAG(y) |
|---|---|
| zero | 1 |
| Denormalized | 2 |
| Normalized | 3 |
| Infinity | 4 |
| Quiet NaN | 5 |
| Signalling NaN | 6 |

| DATE | Programmer | Modification |
|---|---|---|
| 6/01/82 | SB | Documented |
| 10/25/82 | SB | Code Pack: Use MAKE1 |
| 01/06/83 | SB | SR# 30 - Distinguish Sig NaN. |
| 02/07/83 | SB | Update header. |

## 15.44 GETCON - Get constants from table

Category: MTHUTL File: SM&MTH::MS

Name:(S) GETCON - Get constants from table
Name:(S) GETVAL - Get constants from table
Name:(S) PI/4 - Fetch Pi/4 from table

Purpose: Access numeric constants stored in table.

Entry: Table index in P (Selects desired constant).

Exit: Constant selected in C

Alters (INC): C,D[A]

Stk lvls: 0

NOTE:
Presently used only for constant table starting at label TRC90. However by entering at label GETVAL, this code can be used to access constants stored in other tables. The 1st constant corresponds to P=14, the 2nd to P=13, etc.

Algorithm: Value of P determines offset from table start.

History:

| Date | Programmer | Modification |
|---|---|---|
| 6/07/82 | SB | Documented |
| 9/30/82 | SB | Use of D[A] instead of stack. |
| 01/06/83 | SB | New entry: PI/4 |
| 02/07/83 | SB | Move =PI/4 above header-Cosmetic change only. |

## 15.45 MAKE1 - Make 12-dig 1 in C and compare with B.

Category: MTHUTL File: SM&MTH::MS

Name:(S) MAKE1 - Make 12-dig 1 in C and compare with B.

Purpose: Make 12-dig 1.0 in C and test against value in B

Entry: DEC Mode

Exit: C: [0100000000000000], P=14; CARRY Set iff B=C

Alters: C,P,CARRY

Calls: Nothing

Stack Levels: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/02/83 | SB | Documented |

## 15.46 DBLSUB - Double Precision Subtract

Category: MTHUTL File: SM&MTH::MS

Name:(S) DBLSUB - Double Precision Subtract

Purpose: Dbl Precision subtract (used in TRIG Reduction).

Entry: A&C:Y, B&D:X 31-digit positive fixed point values. First 15 high order digits are in A & B. Notation: XH=high order word of X.

Exit: A&C:Z

Carry Clear: Z=Y-X
Carry Set : Z=Y (In this case Y<X)

| | A | B | C | D |
|---|---|---|---|---|
| (ENTRY) | YH | XH | YL | XL |
| (EXIT) | ZH | XH | ZL | XL |

Alters (INC): A,C,Carry

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 6/07/82 | SB | Documented |

## 15.47 DBLPI4 - Generate 31-digit PI/4 or 45

Category: MTHUTL File: SM&MTH::MS

Name:(S) DBLPI4 - Generate 31-digit PI/4 or 45

Purpose: Generate 31-digit value PI/4 -or- 45

Entry: sRAD Status bit (sRAD=1 ==> PI/4, ELSE 45)

Exit: Value in [B,D], P=5.

Calls: PI/4

Alters (INC): B,D,P,Carry

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 6/07/82 | SB | Documented |
| 10/05/82 | SB | Code Pack |
| 10/06/82 | SB | Code Pack - Eliminate call GETCON |

01/06/83 SB Fix header, Pack by moving the entry PI/4 to before GETCON.

## 15.48 TWO* - Double Precision Doubler

Category: MTHUTL File: SM&MTH::MS

Name:(S) TWO* - Double Precision Doubler

Purpose: Dbl Precision doubler

Entry: B&D:X (B:XH, D:XL)

Exit: B&D: 2*X

Alters (INC): B,D,Carry

## 15.49 SHFLAC - Double Precision Shift Left

Category: MTHUTL File: SM&MTH::MS

Name:(S) SHFLAC - Double Precision Shift Left
Name:(S) SHFRAC - Double Precision Shift Right

Purpose: Dbl Precision (Fixed Point) shifts

Entry: A&C:X (A:XH, C:XL)

Exit: A&C:10*X (or X/10)

Alters (INC): A,C,(SHFRAC Only - SB)

## 15.50 SHFRBD - Double Precision Right Shift

Category: MTHUTL File: SM&MTH::MS

Name:(S) SHFRBD - Double Precision Right Shift

Purpose: Dbl Precision (Fixed Point) right shift

Entry: B&D:X (B:XH, D:XL)

Exit: B&D:X/10

Alters (INC): B,D,SB

## 15.51 PI/2 - Generate PI/2

Category: MTHUTL File: SM&MTH::MS

Name:(S) PI/2 - Generate PI/2
Name:(S) PI/2D - Generate signed PI/2

Purpose: Generate Pi/2 (15-Digit form)

Exit: CD: 1.57079632679490

Calls: PI/4

Alters (INC): C,D,P,Carry

Stk lvls: 1

### 15.52 FLIP8 - Toggle status bits

Category: MTHUTL    File: SM&MTH::MS

Name:(S) FLIP8 - Toggle status bits
Name:(S) FLIP10 - Toggle status bits
Name:(S) FLIP11 - Toggle status bits

Purpose: Toggle Status bits

Exit: Toggled status, Carry set if new status = 0.

Alters (INC): Selected Status bit, Carry.

```
+--------------------------------------------------+------------------+
|                                                  |                  |
|   PARUTL  -  Parse Utilities                     |   CHAPTER  16    |
|                                                  |                  |
+--------------------------------------------------+------------------+
```

## 16.1 NTOKNL - Lex Analysis

Category: PARUTL     File: AB&LEX::MS

Name:(S) NTOKNL - Lex Analysis
Name:(S) NTOKEN - Lex Analysis
Name:(S) PRESCN - Lex Analysis
Name:(S) RESCAN - Lex Analysis
Name:(S) VRIABL - Lex Analysis
Name: SHFTKN - Lex Analysis
Name:(S) ALLDUN - Lex Analysis
Name: HOWARD - Lex Analysis
Name:(S) LEAVE - Lexical Analysis

Purpose:
The lexical analyzer scans strings of ASCII characters and associates unique numbers (tokens) with particular substrings (lexemes). The tokens are used by language parsing routines and interpreters.

Entry:
Many different entry points for different purposes.

NTOKNL - Looks for line number, or any other lexeme.
NTOKEN - Looks for any lexeme not a line number.

D1 is current input buffer position.
D0 is current output buffer position.
D(A) is end of output buffer.

PRESCN - Same as RESCAN, except output pointer is still in D0, instead of C(A).

RESCAN - Looks for another token corresponding to a lexeme.

IMPORTANT ENTRY POINT. There is where the lexical analyzer can be restarted if an undesired match occurred with an XWORD.

D1 is reset to start of lexeme to be rescanned.
A(A) is Lexbuffer pointer returned for token to be replaced.
C(A) is reset to start of token to be replaced.
This is the output pointer, which will be in D0 upon exit. This pointer is not actually used by this routine.
D(A) is end of output buffer.

VRIABL - Looks for Basic variable name.

D1 is current input buffer position.
R0 is end of output buffer (done by previous entry points).

SHFTKN - Places token in C(B) in front of tokens in A.

D1 is new input buffer position.
D0 is lexbuffer pointer.
D(A) is execaddress, if there is one.
R0 is end of output buffer.

ALLDUN - Restores output buffer pointer to D0.

D1 is new input buffer position.
D0 is lexbuffer pointer.
D(A) is execaddress, if there is one.
R0 is end of output buffer.

HOWARD - Restores output buffer pointer to D0.

D1 is new input buffer position.
C(A) is lexbuffer pointer.
D(A) is execaddress, if there is one.
R0 is end of output buffer.

LEAVE - Restores end of output buffer to D(A).

D1 is new input buffer position.
D0 is current output buffer position.
D(A) is end of output buffer.

Exit:
P=0.
D1 is new input pointer.
D0 is current output pointer.
A contains token, up to 14 nibbles in length.

B[A] is execution address (if there is one).
B[X] is numeric constant exponent, if there is one.
C[S] is lexbuffer pointer used for RESCAN.
D[A] is end of output buffer.

Calls: ARGCHK, BLDVAR, D=WORD, DGTSTR, GNXTCR, IOFNDO, LDZERO, NUMSCN, Range, SCAN, STLXPT, STRCHK.

Uses.......
A,B,C,P,D0,D1,R0,S0-S3,S11.

Stk lvls: 2

Detail:
The lexical analyzer consists of two parts: scanner and lexicon. The scanner is the code described here with several entry points, one major subroutine (NUMSCN) and many smaller subroutines.

The lexicon is a set of tables:

LXTYPT (lexical type table) is a table of character categories, or types, which lives in system ROM. This table helps the scanner reduce the time selecting which scanning method to use:
Type 0 - Direct: Use transfer character in type table as token.
Type 1 - Word: Scan text table for string match.
Type 2 - Relational: Scan for relational operator.
Type 3 - Number: Call NUMSCN to format constant.

LEXBFR (lexfile buffer) is an I/O buffer in system RAM which contains lextable IDs and maintable addresses.

LXSPDT (speed table) is an optional table within each lexfile which tells where in text table lexemes with a particular first character begin.

LXTXTT (text table) is a table in every lexfile containing the following text information:
Lexeme length - 1 nibble,
Lexeme text - 2-16 nibbles,
Lexeme token - 2 nibbles.

MAINT (main table) is a table in every lexfile which contains token information:
Text offset - 3 nibbles.
Locates text in text table; used in decompiling.
Execaddress - 5 nibbles.

```
                         Self-relative pointer to token's
                         execution address.
            Characterization - 1 nibble.
                              Syntactic class and spacing
                              information.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/01/83 | SA | Figured out register & subr usage |
| 10/17/83 | NM | Attempted to document |

## 16.2 SCAN - Scan LEXfile Text Table For Lexeme

Category: PARUTL  File: AB&LEX::MS

Name:(S) SCAN - Scan LEXfile Text Table For Lexeme

Purpose:
Scan LEXfile text table for text matching keyword machine is trying to parse.

Entry:
D[W] contains keyword machine is trying to parse (up to 8 bytes).
D1 = input pointer (pointing at data which was read into D[W].
D0 pointing at wordsize nibble of first keyword to examine in text table.

Exit:
D1 moved past lexeme in input stream.
Carry set -> lexeme not found.
Carry clear -> token in A[A].

Calls: None.

Uses.......
A[S],C,P,D0,D1.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/01/83 | NM | Added documentation |

## 16.3 NUMSCN - Scan Number In Lexical Analysis

Category: PARUTL File: AB&LEX::MS

Name:(S) NUMSCN - Scan Number In Lexical Analysis

Purpose:
Generate token for numeric constant or solitary ASCII period.

Entry:
D1 at start of numeric character string.

Exit:
DEC mode.
P=0.
S3=1 for incomplete exponent.
D1 past numeric character string.
A[B] = numeric token and mantissa or ASCII digit.
B[M] = right-justified mantissa.
B[X] = exponent.

Calls: DGTSTR, LDZERO, ROUND.

Uses.......
A,B,C,D,P,D1.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|

```
--------   ----------   ----------------------------------
04/01/83   SA           Figured out register & subr usage
10/17/83   NM           Attempted to document
```

## 16.4 LINEP - Parse Main Driver after ENDLINE

Category: PARUTIL    File: JP&PR1::MS

```
Name:(S) LINEP   - Parse Main Driver after ENDLINE
Name:(S) LINEP+  - Parse Main Driver from anywhere
Name:(S) LNPEXT  - Parse Main Driver external entry
Name:(S) LNEP66  - Parse Main Driver return entry

Purpose:  Main driver routine to parse a line:
          1) LINEP entry is called by MAINLP after
             ENDLINE is entered on an input line.
          2) LINEP+ entry is called to parse a
             line, regardless of where the line is.
             Used by direct execute keys (colon
             key definitions) and STARTUP.
          3) LNPEXT entry is the 'external parse'
             entry.  By setting f1RTN, it ensures
             that in all cases (including errors),
             control returns to the caller.  Used
             by TRANSFORM.

Entry:    3 entry points:
        1) LINEP  - Line to be parsed is in the display
                    buffer.
        2) LINEP+ - INBS points to start of input line.
        3) LNPEXT - External Parse Entry
                    Needed statuses (including S13) should
                    be saved.  INBS points to start of
                    input line.  OUTBS points to where
                    tokenized line should go.
                    AUTINC should be zero - may be default

Exit:
      LINEP:
          If valid program statement(s)
             It is edited into current program file
```

```
            If valid calculator BASIC statement(s)
              (including implied DISP)
               It is executed
            Else ERROR exit
               Error message displayed;
               Line redisplayed with cursor;
               Jump to MAINLP
        LNPEXT:
            S5=1 => Line# on line
            S5=0 => No line#
            Carry clear => Line parsed successfully. Compiled
                           line starts at address pointed to
                           by OUTBS.
                           Compiled line length in R3.
            Carry set   => Error in parse.
                           C(3-0) = error#.
                           If C(3-0) = 0000
                             Then found only tEOL ("null line")
                            (May be preceded by a line#; S5
                            indicates presence of a line#)

        NOTE: Any usage of LNPEXT entry rules out implied
              DISP in the case of failed implied LET parse.

  Calls:    GNXTCR, LINNP, NTOKEN, NTOKNL, CRGJMP, I/OAL+,
            OUT2TK, RANGE, EXPPAR, EXPEXC, MAKEBF, RTNSET,
            FILEP+, PEDIT, MOVEUA, SYCOLL, WSRO-3, AVS=DO
            CRLFOF,OVFLCK, TRNFCK, D1=IBS, OBCOLL, LDCSET,
            AUTCLR, LBLCK, PEDITD, SURSTU, RESPTR, OUTB+5,
            FSPC12, !CK, !CK3, RS-RO3, OUT3TK, OUT1TK, WRDSCN
            STMTL+, UPDIN+, OUTBYT, ELSEP, LNPOO, OBLCMP,GETLEE

  Uses:     A-D, RO-R3, D1,DO, SO-S11,
            S-RO-2, S-RO-3, STMTR1 (all 16 nibbles), STMTDO
            F1RTN (only used with LNPEXT entry)

   Stk Lvls: 7

  NOTES: A) Line parse only special checks for TRANSFORM
            (external entry) in four distinct places:
            1) eol,
            2) line#, followed by eol
            3) parse error
            4) correctly parsed line about to be edited into
               program memory.

         B) Implied DISP isn't legal immediately after THEN/ELSE.

         C) Any usage of LNPEXT entry rules out implied DISP in
            the case of failed implied LET parse.  For example:
            10 5*A    would be parsed as:
```

```
10 DISP 5*A
But:
10 A*5    would result in an error.
```

Detail:

```
Key RAM and CPU register usage:
    S-R1-0 Original error# before 1st RESTART
    S-R1-1 Original Error position before 1st RESTART
    S-R0-2 (Subr Save)                              GLOBAL
    S-R0-3 IF clause in progress                    GLOBAL
    S-R1-2 (RESTART ADDR), S-R1-3 (RESTART FLAG)    GLOBAL
    STMTD0 (RESTART PTR)                            GLOBAL
        S4 - No restore of input pointer            GLOBAL
        S5 - Line number found, program stmt        GLOBAL
        S6 - Pending THEN                           GLOBAL
        S7 - Multi-statement line                   TEMP
             Always CLEARED by EXPPAR call          GLOBAL
        S8 - Delete (for PEDIT)                     TEMP
        S9 - Middle of IF  (for ERROR)              TEMP
       S10 - Implied LET Error                      GLOBAL
        R3 - Error Msg Ptr & Line position if IMPLET Err
         D - End of available memory
```

Available status for a Parse routine: S8, S9
These 2 status bits are clear on entry for all parse routines.

Algorithm:

```
   Entry point for TRANSFORM (LNPEXT) saves return
   stack level in S-R0-2 and sets f1RTN  => A:

LINEP: (normal statement parse entry point)
   Copy Display Buffer to Command Stack (MAKEBF)
   Set INBS to start of input line in command stack
   Send Carriage Return & Line Feed  (CRLF0F)
    (so next character will clear display buffer)
   Clear externally invoked flag (f1RTN)

A: Set OUTBS to AVMEMS (Collapses Output buffer)
   Point D1 to start of input line
   Clear S0-S11, S13
   Set D(A) = End of Available Memory
         D0 = OUTBS (Output buffer start)
   Call Block 1

   Retokenize lexeme
   If line#
     Set S5; Decrement D0 (delete statement
     length byte at buffer start); Output line#
     Call Block 5
```

```
          If tEOL
            If externally invoked (f1RTN set)
              THEN error
              ELSE clear AUTO flag; delete line
    B:    Decrement D0
          Call 1.
          Retokenize.

   B1:    If Begin BASIC command (S3=1)
            THEN goto E.
            ELSE If System Command (S3=0,S0=1)
                   THEN error
    C:           If !
                   THEN parse remark; goto 12
                   ELSE error.
         If externally invoked (f1RTN set)
           THEN error;
         Clear AUTO flag
         If tEOL (null line)
           THEN exit parse
           ELSE goto C.

         BLOCK 1:
         Save D0 (statement length byte) in INADDR;
         Increment D0; Clear RESTART flag (S-R1-3);
         Clear Err# (S-R1-0); Call NTOKEN;
         Set RESTART flag if XWORD or XFN &
         save RESTART address (S-R1-2).
         Save contents of LEXPTR (position of D1
         before NTOKEN call) in STMTD0 - will be
         needed to restore input pointer for RESTART.
         Clear Middle of IF flag (S9) - Allows Implied
          LET error to recover as Implied DISP

  Entry point for variable or FN after THEN/ELSE:

  C2:   If variable or FN:
          set implied LET error flag (S10)
          If no line# on line
            Clear AUTO flag
   G:     Try implied LET parse
          Goto 10.
        If looking at 1st lexeme on line
          If line# followed by !
            set S5; output line#; save D0 (location of
            statement length byte) in INADDR; increment
            D0; Parse remark; goto 12
        If not a terminator (eg not tEOL,@,!,tELSE)
          If legal implied DISP statement followed by
          a terminator
            If no line number on line
```

```
            Clear AUTO flag; goto 10:
    Restore D1,D0; return
    END OF BLOCK 1

  ***Block 5 only returns if a label is not found***

    BLOCK 5
    Save D0 (position of statement length byte) in
    INADDR; increment D0
    If quote
      Set appropriate flag(s);
      Step over it; Call FILEP+
      If legal
        THEN If matching closing quote
 8:              THEN if colon follows
                        THEN LEGAL LABEL;
                             Output tLBLST & label
                             If tEOL follows
                               THEN goto 13
                               ELSE goto 11 (parse as @)
                        ELSE RESPTR; Return
                 ELSE RESPTR; Return
        ELSE RESPTR; Return
    If 1st character is letter
      RESPTR; GNXTCR; FILEP1; Goto 8
    END BLOCK 5

D:  If not Calculator BASIC (S0=0)
      THEN If begin BASIC (S3=1)
             THEN error
             ELSE goto C.
E:  If in IF statement (S-R0-3 nonzero)
F:    If not legal after THEN/ELSE (S2=0)
        THEN error
      If pending THEN (S6=1)
        If token is IF token
          THEN error

    If XWORD
      THEN Output 3-byte token
      ELSE Output 1-byte token
    Calculate Parse address
    Clear flags (S0,S8,S9,S10)
    Gosub to Parse routine   (CRGJMP)
    If Middle of IF return  (Carry Set)
      THEN Extended IF token already output;
           INADDR points to following byte;
           D0 is pointing past that byte
           S9 is set (middle of IF flag)
           S-R0-3 nonzero (IF in progress)
 H:        If S5=1
```

```
              THEN goto B1
              ELSE goto D

 10: Normal stmt return (carry clr)
     Get Next Token
     If ELSE
       If no pending THEN (S6=0)
         THEN error
         ELSE Clear S6; Decr D0; Output t@;
              Call STMTLN, UPDIN+; Output tELSE
              Call ELSEP; goto 10
     Check legal stmt terminators (@,!,EOL)
      Clear S7
      If @  (Multi-statement line)
 11:    THEN Set S7, Output t@
        ELSE If !   (Remark)
               THEN Output t!, Remark; goto 12
               ELSE If EOL
 12:                  THEN Output tEOL
                      ELSE Error Exit --> Excessive Chars
 13:  Output terminator
      Clear S10 (Implied LET error flag)
      Calculate & write out statement length
      If multi-statement line
        If S5=1
          THEN Call 5; Goto B
          ELSE Call 1; Goto D
      Set AVMEMS to D0
      If line# found (S5=1)
        If externally invoked (f1RTN set)
          THEN exit with carry clear
          ELSE Edit line into program memory (PEDIT)
               Return to Main Loop
      Calculate output buffer length, move to I/O buffer
      area; call SYCOLL (Resets AVMEMS,OUTBS to SYSEN)
      Execute Calc. BASIC Stmt (BSCEXC)

     See the portion of the algorithm handled in IFP
     in JP&PR3
```

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/08/82 | S.W. | Updated documentation |
| 10/15/82 | S.W. | Added call to D1=IBS |
| 01/07/83 | S.W. | Added algorithm |
| 06/03/83 | JP | Set AVMEMS @ D0 before PEDITD call |
| 11/01/83 | S.W. | Modified documentation header. |

## 16.5 LBLINP - Parse Line Number or Label

Category: PARUTL    File: JP&PR1::MS

```
Name:(S) LBLINP  -  Parse Line Number or Label
Name:(S) LBLNIF  -  Parse Line Number or Label after THEN/ELSE
Name:(S) LINP    -  Parse Line Number only

Purpose:
     Parse line number or label:
     LBLINP or LBLNIF entry allows line number or label
     LINP entry looks for line number only

Entry:
      DO points past last token written to output buffer
      D(A) contains (AVMEME)
      3 entry points:
      1) LBLINP - D1 pointing to alleged line# or label
      2) LINP   - D1 pointing to alleged line#. S9=1
      3) LBLNIF - Exit conditions from NTOKNL: P=0,
                  A(B) contains token to check, D1 past
                  alleged line# or label.
                  S9=0 => Allow line# or label
                  S9=1 => Allow line# only

Exit:
     Carry clear
           Line# or label found and tokenized
           D1 past line# or label
           DO past tokenized line# or label
           If line# found,
              A(3-0) contains line#
              The following 11 nibbles are output:
              tLINE# 00000 <4 nib BCD line#>
           If label found, it is output in 1 of 2
              formats using either LABELP or FSPC10:
              tLBLRF <string expr>  - LABELP
              tLBLRF tLITRL <ascii label> - FSPC10

     Carry set
           LBLINP entry => 1st char not letter | line#
           LINP   entry => Line# not found
           LBLNIF entry:
```

```
              S9=0 on entry => 1st char not letter | line#
              S9=1 on entry => Line# not found

Calls:        NTOKNL, LINPN2, LABELP, OUT3TK, OUT2TC
              OUTBYT, RESPTR, FSPC10 (golong)

Uses.......
 Exclusive:   A,B,C,D1,S9,S10,D0,D1
 Inclusive:   A,B,C,D1,S9,S10,D0,D1,S0-S3,S7,S11,D(S),R0,R1,R3,P

Stk lvls:     5

Detail:       S9 used by LBLINP entry only
             S10 used by LABELP to ensure no reserved word check

Algorithm:

      If next token = line#  (LINPN2)
         Output line# token  (OUTBYT)
         Zero out Line# jump address field
         Output line#        (OUT1TK)
         Return, carry Clear
      If S9=1 (Line# Parse only)
         Return, carry Set
       else
         Output Label Reference Token  (OUTBYT)
         Restore Input pointer  (RESPTR)
         Set No RESERVE word parse flag (S10)
         Parse label  (LABELP)
         If legal label
            If string expression
               RTNCC  (Label already output)
            else
               golong to Output Literal Token & Label
         else
            Back up Output pointer over Label Token
            RTNSC   (Illegal first character found)

NOTE:

Tokenized form:

<lineno> ---> (Lineno Token) (5 nib jump addr) (4 nib Line#)
<label>  ---> (Label Ref Token) (String Expression)
<label>  ---> (Label Ref Token) (Literal Token) (ASCII Label)

History:

  Date      Programmer           Modification
--------    ----------    ---------------------------------
07/08/82    JP            Modified documentation
```

| | | |
|---|---|---|
| 11/23/82 | JP | Removed Stack level saving |
| 11/29/82 | JP | Removed S2/Label found flag |
| 11/01/83 | S.W. | Cleaned up documentation header |

## 16.6 EOLCK - Check for EOL,@,!,ELSE

Category: PARUTL    File: JP&PR1::MS

```
Name:(S) EOLCK   -  Check for EOL,@,!,ELSE
Name:(S) EOLCKR  -  Check for EOL,@,!,ELSE

Purpose:
     Checks for tEOL, @, ! , tELSE
     EOLCKR entry calls RESPTR before checking.

Entry:
     EOLCKR - NTOKEN (or WRDSCN) has already been
              called; D1 past keyword/character to
              check (except if token was tEOL)
     EOLCK  - D1 at optional blanks preceding
              keyword/character to check.

Exit:
     P=0
     A(B) = Token found
     D1 past the keyword/character found
     Carry Set =>
       Statement terminator found (tEOL, tELSE, @, !)
     Carry Clr =>
       Statement terminator not found

Calls:       WRDSCN, RESPTR

Uses.......
 Exclusive: A-C,D1,R1,R2,P
 Inclusive: A-C,D1,R0-R2,S0-S3,S11,P

Stk lvls:   4

Detail:     D0 is preserved from entry
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | JP | Modified documentation |
| 11/02/83 | S.W. | Modified documentation - Routine doesn't use D0. |

## 16.7 WRDSCN - Keyword Scan from Table

Category: PARUTL File: JP&PR2::MS

Name:(S) WRDSCN - Keyword Scan from Table
Name:(S) WRDSC+ - Keyword Scan from Table

Purpose:

WRDSCN tries to match the text pointed to by D1 with any of the keywords specified by the caller; the acceptable keyword tokens are listed in table format immediately following the call to WRDSCN or WRDSC+. If one of the specified keywords is found, its corresponding tokenization is output and control branches to the label specified by the WRDSCN table.

To accomplish this, WRDSCN repeatedly calls NTOKEN until a token match is found or until all keyword tables in the HP-71 have been searched.

The WRDSC+ entry point is identical to the WRDSCN entry, except that WRDSC+ first calls RESPTR.

Entry:

D(A) = (AVMEME)
Table address is on return stack upon entry
(ie. table immediately follows GOSUB.)
D0 points into output buffer

WRDSC+: LEXPTR contains address pointing to optional blanks preceding characters to tokenize.
WRDSCN: D1 at optional blanks preceding characters to tokenize.

Exit:
P=0
Match found=>
No return to caller; control transferred to specified label.
Token output to address pointed to by D0
Specified token in register A
D1 past specified keyword
D0 past keyword tokenization

Match not found=>
Return with carry clear
Last token found in A(B)
D1 past corresponding keyword

Calls: NTOKEN,RESCAN,OUTNBS,RESPTR,XCHECK,XCHEK1

Uses.......
Exclusive: A,B,C,R1,R2
Inclusive: A,B,C,R1,R2,S0-S3,S11,R0

Stk lvls: 3

Detail:

Sample call:

```
GOSUBL =WRDSCN
CON(2) =tBASE          1-byte token
REL(3) =FIXP           If tBASE found, goto FIXP
CON(6) =tANGLE         3-byte token
REL(3) OPTP10          If found, goto OPTP10
CON(6) =tROUND         3-byte token
REL(3) OPTP20          If found, goto OPTP20
CON(2) 0               00 byte terminates table
  ........
code continues here
```

---------------------------------------------

How it works:

Calls the lexical analyzer and scans through table trying to match one of the tokens(XWORD or regular) and jumps to an address specified in the table table for that token.
If the token returned by the lexical analyzer is not matched but is an XWORD, the lexical analyzer is restarted and the table is re-scanned from the beginning.
If no match can be found then execution continues

following the end of the table.

The table consists of any number of entries, where each entry is a token followed by a 3-nibble relative address which is branched to if that token is matched. A token may be either 2 or 6 nibbles long, depending on whether it is an XWORD/XFN/FFN token versus a mainframe token. The table is terminated by a 00 token; the table is immediately followed by the code to handle the "otherwise" case (ie. the table has been skipped over).

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/07/82 | JP | Modified documentation |
| 10/17/82 | B.S. | Modified routine to use 3 nibble relative entries instead of 4 nibble absolute. |
| 02/11/82 | B.S. | Modified routine to handle FFNs |
| 11/02/83 | S.W. | Modified header documentation. |

## 16.8 SYNTXe - "Syntax" Parse Error Exit

Category: PARUTL   File: JP&PR2::MS

Name:(S) SYNTXe - "Syntax" Parse Error Exit
Name:(S) IVEXPe - "Invalid Expression" Parse Error Exit
Name:(S) IVPARe - "Invalid Parameter" Parse Error Exit
Name: ERR3 - "Invalid Parameter" Parse Error Exit
Name:(S) MSPARe - "Missing Parameter" Parse Error Exit
Name:(S) IVVARe - "Invalid Variable" Parse Error Exit
Name:(S) ILCNTe - "Illegal Context" Parse Error Exit
Name:(S) EXCHRe - "Excess Characters" Parse Error Exit
Name:(S) QUOEXe - "Quote Expected" Parse Error Exit
Name:(S) PRNEXe - ") Expected" Parse Error Exit
Name:(S) FSPECe - "Invalid Filespec" Parse Error Exit
Name:(S) PARERR - Generic Parse Error Exit

Purpose:

Parse ERROR Exit Routines.
The 1st 11 entry points above all fall into PARERR.
Depending on entry conditions, PARERR may:
1) Display the error message and redisplay the line, with the cursor flashing on the character pointed to by D1 or (LEXPTR).
or
2) Attempt to reparse the statement as an implied DISP. (S10=1,S9=0 on entry)
or
3) Attempt to reparse the statement as an implied GOTO <label>. (S9=S10=1 on entry)
or
4) Restart the lexical analyzer and reparse the entire statement. (RESTART flag nonzero)

Entry:

S4=1 if D1 set at error position.
S4=0 if LEXPTR contains address of error position.
S10=1 if implied LET error (try implied DISP)
S10=S9=1 if middle of IF stmt and implied LET error

This entry condition is handled by the driver:
```
S-R1-3 (RESTART Flag) = 0 => Don't restart
                      = F => Normal restart
                      = E => Restart of extended IF
```

PARERR - Lower 4 nibbles of D0 contain error#

Exit:

If S10=0, (S-R1-3)=0 on entry
Exit through MFERR:
Display error message
Redisplay Input Line with Cursor at Error
Returns to Main Loop

If RESTART flag set (S-R1-3)#0 on entry
exit through RESTAR

If 'Normal' implied LET error (S10=1 & S9=0)
Try implied DISP parse

If Implied LET error & Middle of IF (S10=S9=1)
Try implied GOTO <label> parse

Calls: RESPTR, R3=D10, D1C=R3, EOLCK, RSTRT?, TRNFCK, EOLCK+, NTKEN+, UPDIN+, LBLINP

Uses: A-C, R0,R3, D0,D1, S4,S8-S10

Stk lvls: 1

6 if Implied LET/Middle of IF/Restart

Algorithm:
```
If S4=0
  THEN RESPTR
If RESTART flag (S-R1-3) set
  THEN goto RESTAR;
  ELSE If previously restarted (S-R1-0 [err#] #0)
         THEN Restore D1 to original error position
              using S-R1-1; Set D0 from S-R1-0;
       If Implied LET error (S10=1)
         Restore D1,D0 from R3; Clear S10;
         If not in middle of IF (S9=0)
           THEN try implied DISP
           ELSE Decrement D0 4 nibbles
                (over tEXTIF & stmt length byte);
                Recover old INADDR from S-R0-0;
                Call GOSUBP;
       Handle as error.
```

Note:

If error is ILLEGAL CONTEXT & S9 is set, then S10 is cleared. This prevents illegal context errors immediately after THEN/ELSE from being interpreted as labels.

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/07/83 | S.W. | Added algorithm |
| 02/04/83 | JP | Added mneumonc entry point names |

## 16.9 RESTAR - Restart Lex Analyzer

Category: PARUTIL File: JP&PR2::MS

Name: RESTAR - Restart Lex Analyzer
Name:(S) REST* - Restart Lex Analyzer

**Purpose:**

Restarts the Lexical Analyzer when the parse of an XWORD token fails; allows the parser to find smaller keywords in the same LEX file, as well as similarly spelled keywords in other LEX files.

The RESTAR entry point is used by the parse error driver to try all possible statement parses, before reporting an error; the original parse error and position is saved and is later restored if all subsequent parse attempts fail.

The REST* entry point is used by a LEX file when a parse fails and it is known that RESTAR will find a subsequent statement parse in the mainframe which can give a clearer, more coherent error message. This entry point ensures that the caller's error number and error position is NOT preserved anywhere - it is as though the keyword was never found.

**Entry:**

```
 (STMTDO) = Input pointer for restart
 (S-R1-2) = Restart Address

2 entry points:
RESTAR - If RESTAR hasn't been previously called
          Then C(A)=0
               DO=Latest error# generated
               D1=S-R1-0
               A(A)=Error position
          Else...
               (S-R1-0)=Original error#
               (S-R1-1)=Original error position
         If not failed label parse after THEN/ELSE
          Then S8=0
               (INADDR) = addr of last stmt length byte
               C(S)#E iff Extended IF
          Else...
               S8=1
               R3(A) pts 2 nibs past last stmt len byte


REST*  -  (INADDR) = address of last stmt length byte
          (S-R1-3)#F iff Extended IF
```

**Exit:**

Control is turned over to the main parse driver.

**Calls:** RESPTR,RESCAN,R.STPR,STLXP2,SVRST2,EXTIF+

**Uses.......**

```
 Exclusive: A,C,D1,D0,S8
 Inclusive: A-C, D1,D0, S0-S3,S8,S11, R0,R3

Stk lvls:   3

Detail:
     The component parts of RESTART are as follows:
     S-R1-0  Original error#; set prior to 1st time
               through RESTAR
     S-R1-1  Original error position; set prior to 1st
               time through RESTAR
     S-R1-3  Flags the parse error handler whether or
             not to RESTART the lexical analyzer.
             If S-R1-3 is nonzero, STMTD0 contains the
             address at which to set D1 to restart and
             S-R1-2 contains the restart address.
             S-R1-3 is cleared when NTOKEN is first
             called; It is set (along with STMTD0) when
             the begin BASIC token is an XWORD.
     S-R1-2  Contains RESTART address.  Set initially
             when NTOKEN first called.  Updated when
             RESCAN called in RESTAR.
     STMTD0  Contains address at which D1 should be at
             when restarting the lexical analyzer.  Set
             and cleared with S-R1-3.

Algorithm:
     If 1st time thru RESTART for this lexeme
     (S-R1-0 contains 0)
       Save err# in S-R1-0 & position in S-R1-1;
     Clear RESTART flag (S-R1-3);
     Get input ptr from STMTD0 & write out to LEXPTR
     (needed 'cause RESCAN doesn't save as NTOKEN does);
     Retrieve RESTART addr for lexical analyzer (S-R1-2);
     Restore D0 from INADDR;
     Call RESCAN; Set RESTART flag (S-R1-3) if XWORD/XFN;
     Save RESTART address in S-R1-2;
     Goto M (main parse driver - JP&PR1).

History:

  Date      Programmer              Modification
--------    ----------   ----------------------------------
07/06/82    JP           Modified documentation
08/23/82    S.W.         Added documentation on S-R1-2, S-R1-3
                          and STMTD0.
11/15/82    S.W.         Deleted error exit option - wasn't
                          used anywhere
05/24/83    S.W.         Added REST* entry point for use by
                         language extensions; this is an alterna??
                         tive to the 'usual' error exit (the
```

usual error exit saves the original error and restores it if no other parse works). REST* can be used ONLY if it i?? known that restart will eventually give control to a mainframe parse routine; REST* can be useful to prevent obscure error messages. If a previous parse error occured, the first one generated in the 'usual' way is preserved; otherwise the next error generated in the 'usual' way (not using REST*) is preserved.
For example:
The HPIL parse for ON INTR, may choose to suppress its error message/position, in favor of any one given by ON ERROR|TIMER|<expr>

## 16.10 GNXTCR - Get Next Non-blank Character

Category: PARUTL File: JP&PR2::MS

Name:(S) GNXTCR - Get Next Non-blank Character
Name:(S) OAGNXT - Output byte, Get Next Non-blank Character
Name: GNXCR+ - Get Next Non-blank Character

Purpose:
Gets next non-blank character.

OAGNXT first outputs a byte from A(B) before scanning for the next non-blank character.

GNXCR+ first increments D1 by 2 before scanning for the next non-blank character.

Entry:
OAGNXT - A(B) contains byte to output
D(A) = (AVMEME)
D0 points to where byte to be written
D1 points to where to begin scanning for next non-blank character.

GNXTCR - D1 points to where to begin scanning for next non-blank character.
GNXCR+ - D1 points 2 nibbles prior to where to begin scanning for the next non-blank character.

Exit:
D1 points to next non-blank character
A(B) = Next non-blank Character
C(B) = Ascii Blank
P = 0
Carry set

If not enough memory to output byte, generates MEMERR (OAGNXT entry only)

Calls: OUT1TK - (ONGNXT Only)

Uses: A(B),C(B), DO (ONGNXT Only),D1, P

Stk lvls:
GNXTCR: 0
GNXCR+: 0
ONGNXT: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/07/82 | JP | Modified Documentation |
| 09/24/82 | FH | Modified Documentation |
| 11/02/83 | S.W. | Fixed documentation header |

## 16.11 RESPTR - Restore Input Pointer

Category: PARUTL File: JP&PR2::MS

Name:(S) RESPTR - Restore Input Pointer

Purpose:
Restores D1 to its position prior to NTOKEN call

Entry:
    (LEXPTR) = address of input pointer (advanced past leading blanks) prior to last call to NTOKEN.

Exit:
    D1 re-positioned
    Carry clear

Calls: none

Uses: A(A), D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | S.W. | Added documentation |

## 16.12 COMCKO - Check Comma & Output Comma Token

Category: PARUTL File: JP&PR2::MS

Name: COMCKO - Check Comma & Output Comma Token
Name:(S) COMCK+ - Check Comma & Output Comma Token

Purpose:
    Checks for tCOMMA & outputs it if found.

    COMCKO entry requires that NTOKEN be called before checking for tCOMMA.

    COMCK+ entry assumes that NTOKEN has already been called.

Entry:
    D(A) = (AVMEME)
    D0 = pointer to where tCOMMA to be output
    2 entry points:

```
        1) COMCK0 - D1 at opt. preceding blanks before
                    alleged comma.
        2) COMCK+ - A(B) contains byte to compare against
                    tCOMMA.

   Exit:
        P        = 0
        Carry set => tCOMMA found & output
                     D0 incremented past tCOMMA
                     COMCK0 entry:
                      D1 pts past ascii comma
                     COMCK+ entry:
                      D1 preserved from entry

        Carry clr => tCOMMA NOT found
                     D0 preserved from entry
                     COMCK0 entry:
                      A(B) = token found
                      D1 advanced past corresponding text
                     COMCK+ entry:
                      A(B) preserved from entry
                      D1 preserved from entry

        If tCOMMA found, but not enough memory to
        output it, exits to MEMERR

   Calls:     NTOKEN, COMCK1

   Uses:      C, D0, P (COMCK+ entry)
              A-C, D1,D0, S0-S3,S11, R0, P (COMCK0 entry)

   Stk lvls:  3

   History:

      Date     Programmer            Modification
    --------   ----------   ----------------------------------
    05/11/83   S.W.         Added documentation
```

16.13 #CK - Check for #

Category: PARUTL File: JP&PR2::MS

Name:(S) #CK - Check for #

Purpose:
Compares next non-blank character against ascii #

Entry:
D1 points at optional blanks preceding character to
compare against

Exit:
P = 0
D1 points to next non-blank character
A(B) = Next non-blank character
Carry clear => Character is #
Carry set => Character is not #

Calls: GNXTCR

Uses: A(B), C(B), D1, P

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/03/83 | S.W. | Added documentation header |

16.14 NXTP - NEXT statement parse

Category: PARUTL File: JP&PR3::MS

Name:(S) NXTP - NEXT statement parse

Purpose:
Parses NEXT Statement. Also useful for
simple numeric variable parse.

Entry:
D(A) = (AVMEME)

D1 at alleged simple numberic variable
D0 points into output buffer

Exit:
Carry clear =>
Simple numeric variable found and output
P = 0
Carry clear
D1 advanced past variable
D0 points past tokenized variable

Else error exit to PARERR with eILVAR

Calls: VARP

Uses.......
A-C, D0, D1, S0-S3, S11, R0

Stk lvls: 4

NOTE:
This also serves as parse for NEXT statement

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/03/83 | S.W. | Added documentation |

## 16.15 VARP - Variable Parse

Category: PARUTIL File: JP&PR3::MS

Name:(S) VARP - Variable Parse
Name: VARP05 - Variable Parse

Purpose:
Checks for a variable token. If found, it is output; if the token is not a variable token, an error exit is taken.

VARP entry assumes that D1 points to optional blanks preceding the text to check.

VARP05 entry assumes that NTOKEN has already been called, and that the token to check is in register A.

```
Entry:
     D(A) = (AVMEME)
     D0 points into the output buffer
     2 entry points:
     1) VARP   - D1 at optional blanks preceding text to
                 be examined.
     2) VARP05 - Register A contains alleged variable token.
                 D1 points past the corresponding text as per
                 NTOKEN exit.
                 (LEXPTR) as per NTOKEN exit.

Exit:
     Return to caller =>
     Variable parsed
     Tokenized variable written to output buffer
     D0 past variable tokenization in output buffer
     D1 past variable name
     Carry set =>
            Numeric variable found
     Carry clr =>
            String variable found

     Error exit if variable not found or if MEMERR

Calls:      NTOKEN, OUTVAR

Uses.......
 Exclusive: A,D0,D1
 Inclusive: A,B,C,S0-S3,S11,D0,D1,R0

Stk lvls:   3

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | JP | Modified documentation |

## 16.16 ARRYCK - Parses Doubly Dimensioned Array

Category: PARUTL    File: JP&PR3::MS

```
Name:(S) ARRYCK  -  Parses Doubly Dimensioned Array
Name:    ARRY01  -  Parses Singly Dimensioned Array

Purpose:
     ARRYCK entry is useful for parsing one or two dimensional
     arrays.

     ARRY01 is useful for parsing a single numeric expression
     followed by a closing parentheses; this could be a single
     dimension array parse or TAB parse.

Entry:
     D(A) = (AVMEME)
     D1 points at input stream
     D0 points into output buffer
     2 entry points:
     1) ARRYCK - D1 @ Left parentheses.
     2) ARRY01 - D1 past left parentheses.

Exit:
     Valid parse =>
      Return to caller with carry Set
      Subscript(s) output
      D1 points past the closing parentheses
      D0 points past the output subscript(s)
      ARRYCK entry:
         B(0) = # subscripts    (1 or 2)

     Else Error exit
         Invalid or non-numeric expression
         No closing paren

Calls:       NUMCK, COMCK1

Uses.......
 Exclusive: A,B(A),C,D0,D1
 Inclusive: A-C,D(15-5),D0,D1,R0,R1,S0-S3,S7,S11,FUNCD0

Stk lvls:   5

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | JP | Modified documentation |

## 16.17 NUMCK - Valid Numeric Expression Check

Category: PARUTL File: JP&PR3::MS

```
Name:(S) NUMCK   - Valid Numeric Expression Check
Name:(S) NUMC++  - Move D1 1-Byte, Do Valid Numeric Expr Check
Name:    NUM+O   - Move D1 1-Byte, Output Byte, Ck for Num Expr
Name:    NUMKO   - Output Byte, Check for Valid Numeric Expr

Purpose:
     Checks for and Outputs Valid Numeric Expression
     Error Exit if not found

Entry:
     D(A) = (AVMEME)
     D1 points at input stream
     D0 points into output buffer
     4 entry points:

     NUMCK   - D1 points at optional blanks preceding
               alleged numeric expression.
     NUMC++  - D1 is 1-byte prior to alleged numeric expr
     NUMC+O  - D1 is 1-byte prior to alleged numeric expr
              A(B) = byte to write to output buffer prior
               to parsing the numeric expression.
     NUMCKO  - D1 points at optional blanks preceding
               alleged numeric expression.
              A(B) = byte to write to output buffer prior
               to parsing the numeric expression.

Exit:

     Valid numeric expression parsed =>
         Return to caller with carry clear
         P=0
         Tokenized expression written to output buffer
         D0 points past the tokenization
```

```
          Register A contains the tokenization of the text
           FOLLOWING the numeric expression
          D1 points past the corresponding text
          R3(9-5) = the input pointer to the numeric expr
          R3(A)   = the pointer to the tokenized num. expr
          NUMCK entry:
           R3(A)   = D0 on entry
           R3(9-5) = D1 on entry
          NUMC++ entry:
           R3(A)   = D0 on entry
          NUMC+0 entry:
           The value in A(B) on entry was output prior to
           the tokenized numeric expression.
          NUMCK0 entry:
           R3(9-5) = D1 on entry
           The value in A(B) on entry was output prior to
           the tokenized numeric expression.

      Error exit - Invalid or non-numeric expression
```

**Calls:** r3exp+ (EXPPAR,R3=D1C), D1C=R3

**Uses:** A-C,D(15-5), R0,R1,R3, S0-S3,S7,S11, FUNCD0

**Stk lvls:** 4

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |
| 11/11/82 | S.W. | Added entry points NUMC+0 and NUMCK0 |
| 05/12/83 | S.W. | Eliminated NUMCK+ entry point |

## 16.18 STRGCK - Valid String Expression Check

Category: PARUTL File: JP&PR3::MS

**Name:(S) STRGCK - Valid String Expression Check**

Purpose:
  Valid String Expression Check

Entry:
  D1 @ Start of Alleged String expresssion
  D(A) = (AVMEME)
  D0 points into output buffer

Exit:
  Valid string expresion =>
   Return to caller with carry clear
   P=0
   Tokenized string expression written to output buffer
   D0 past string expression tokenization
   A = tokenization of text FOLLOWING string expression
   D1 past corresponding text of tokenization in A

  Else error exit

Calls: r3exp+ (EXPPAR,R3=D10), NUMCK

Uses: A-C,D(15-5),R0,R1,R3,S0-S3,S7,S11,FUNCD0,D0,D1

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |
| 07/06/83 | S.W. | If invalid expr, don't restore ptr |

## 16.19 COMCK - Comma Check

Category: PARUTIL File: JP&PR3::MS

Name:(S) COMCK - Comma Check
Name: COMCK1 - Comma Check

Purpose:
  COMCK entry checks to see if the following

```
        text's tokenization is tCOMMA.

        COMCK1 entry checks to see if A(B) contains tCOMMA.

   Entry:
        COMCK  - D1 points at optional blanks preceding
                 text to tokenize.
        COMCK1:  A(B) = Token to Check

   Exit:
        P=0
        Carry set   => tCOMMA found
                       A(B)=C(B)=tCOMMA
                       COMCK entry:
                        D1 past ascii comma

        Carry clear => tCOMMA NOT found
                       C(B)=tCOMMA
                       COMCK entry:
                        A contains text's tokenization
                        D1 past corresponding text

   Calls:      NTOKEN    - COMCK entry only

   Uses:       A(B), C(B), P              -   COMCK1 entry
               A-C, D1, P, S0-S3,S11, R0  -   COMCK entry

   Stk lvls:   3  -  COMCK  entry
               0  -  COMCK1 entry

   History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |

## 16.20 OUTLIT - Output Delimited Literal

Category: PARUTL     File: JP&PR3::MS

Name:(S) OUTLIT - Output Delimited Literal

Name:(S) OUTLI1 - Output Delimited Literal
Name: DATACK - Output Literal Delimited by Quotes

Purpose:

OUTLIT and OUTLI1 entry points output a string of literals delimited by a specified delimiter (this delimiter may or may not be a quote). The only difference between these two entry points is that OUTLIT takes an error exit if no closing delimiter is found; OUTLI1 simply returns with the carry set in this case.

DATACK entry parses a string delimited by either single or double quotes. If no closing delimiter is found, DATACK takes an error exit.

Entry:

D(A) = (AVMEME)
D1 points into the input stream
D0 points into the output buffer
3 entry points:
1) OUTLIT - D1 points at the delimiting character
A(B) contains the ascii delimiter
P=0
2) DATACK - D1 points at optional blanks preceding
the alleged single or double quote.
3) OUTLI1 - D1 points at 1st character after the
delimiter.
A(B) contains the ascii delimiter.
P=0

Exit:

Carry clr =>
D1 is advanced to the character following the
closing delimiter.
The literal up through the closing delimiter
has been written to the output buffer.
D0 points past the closing delimiter.

Carry set (OUTLI1 entry only) =>
D1 is 2 nibbles past D0 (Endline)
All characters, up to but not including D0,
have been output
D0 points past the characters which have been
output

Else error exit (OUTLIT, DATACK only)
w/ D1 at 0D - Error is : Quote Expected

Calls: OUT1TK, GNXTCR

Uses.......

Exclusive: A,B,C,S4,D0,D1
Inclusive: A,B,C,S4,D0,D1

Stk lvls: 2 (OUTLI1)
3 (OUTLIT)

NOTE: It may be desirable to limit usage of OUTLIT to delimiters which are single or double quotes, since the error message generated is "Quote Expected".

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |
| 10/12/82 | S.W. | OUTLI1 entry doesn't error exit |

## 16.21 OUTVAR - Output Parsed Variable

Category: PARUTL File: JP&PR3::MS

Name:(S) OUTVAR - Output Parsed Variable

Purpose:
Writes tokenized variable in A to the output buffer

Entry:
P=0
D(A) = (AVMEME)
D0 points into output buffer
Register A contains variable tokenization
from NTOKEN call

Exit:
Carry Clear
Variable tokenization written to output buffer
D0 past the tokenization

Calls: OUT1TK, ARANGE

Uses: A, C(A), D0

Stk lvls: 2

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |

## 16.22 FSPECp - File Specification Parse

Category: PARUTL File: JP&PR3::MS

Name:(S) FSPECp - File Specification Parse
Name: FSPC10 - Outputs Literal File Name

Purpose:
File Specficiation Parse

FSPECp accepts string expressions as valid file specifiers. Quoted strings are considered string expressions.

Unquoted strings are carefully parsed to ensure they conform to the correct syntax. File names (if they're given) must start with a letter and, unless a poll handler responds, are limited to 8 characters. Remaining characters may be letters or digits. Parse includes any device specifiers that are given. If a device is included, a file name is optional.

If a valid file name is followed by '@' or by any char. not in the ascii range of '.' - 'z', the file specifier is considered to be terminated.

If a valid file name is followed by ':', FSPECp attempts to parse the device that should follow. If the device is not MAIN, PORT, CARD, or PCRD, a device poll is done.

If a valid file name is followed by any other character (this file names over 8 characters long), a poll is done.

FSPC10 entry is used to write a legally parsed file name to the output buffer; it is generally called after FILEP has been called successfully. Its entry conditions are matched by the exit conditions from FILEP.

```
Entry:
     D(A) = (AVMEME)
     D0 points into output buffer
     FSPECp - D1 at start of alleged file specifier
                in the input stream
     FSPC10 - File name in A
              P=0
              C(S) = #NIBS-1 in the file name

Exit:
  FSPECp entry
  ------------
     Carry Clear:
        P=0
        File specification accepted & output
        D0 past tokenized file spec. in output buf
        D1 past valid file specification
        S7=1 iff String expression

     Carry Set:
        P=0
        R3(A)=D0 on entry; R3(9-5)=D1 on entry
        S7=1
           Reserved word in A
            (KEYS,ALL,TO,INTO,CARD)
           Reserved word has been output
           D0 past output reserved word in output buffer
           D1 past reserved word in input buffer
        S7=0
           Bad file parse
            (unrecognized device, extraneous chars after
             file name, invalid 1st character in file name)
           D1 restored to what it was on entry
           C(A)=D0 on entry

     Else hard-wired error exit:
        Possibilities:
        Bad Port#            (from NUMCK)
        No closing paren     (to ERR01)

  FSPC10
```

```
  ------
        File name written properly to output buffer:
         tLITRL <ascii file name>
        D0 points past output file name
        Carry clear

Calls:      FILEP,OUTNBC,POLLD+/hFSPCp,POLLD+/hDEVCp,D1C=R3
            RANGE,OUTBYT,WRDSCN,RESPTR,D=AVME,GNXTCR,OUT1TK
            NUMCK,AVS=D0

Uses........
 FSPC10   : C(B), D0
 FSPECp   : A-C,P,XM,D(15-5),D0,D1,S0-S3,S7,S10,S11,R0-R3,
            FUNCD0

Stk lvls:   5 (FSPECp)
            2 (FSPC10)

Detail:     File specifiers which are unquoted strings are
            tokenized with a special 1-byte token preceding
            them:  tLITRL <unquoted string>

            File specifiers which are string expressions
            or reserved words are NOT preceded by any such
            special byte.

            For HPIL tokenization, see detail under
            the following poll's documentation: pFSPCp and
            pDEVCp.

Algorithm:

FSPECP: Try Mainframe File Parse (FILEP)
        If Mainframe file  (Carry set)
           If string expression (S7=1)
              Return CC
           else                  (Unquoted literal)
              If mainframe terminator
                 Output filename  (OUTNBC)
                 RTNCC
              else
     1:          If current char  = ":"
                    If filename specified
                       Output filename          (FSPC10)
                    If Mainframe Device word
                       Output Device word
                       If PORT
                          If "(" follows
                             Verify Port#       (NUMCK)
                             Verify ")"
                       RTNCC
```

```
                else
                   Restore Input Pointer (RESPTR)
                   POLL for Device Parse
                   Return with carry as set
             else
                Restore D1  (R3)
                POLL for <file spec> Parse
                RTN with carry as returned
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/06/82 | J.P. | Modified documentation |
| 04/08/83 | S.W. | If Invalid Filespec but not reserved word (on exit carry set & S7=0), then D1 restored to what it was on entry before return to calling routine. |

## 16.23 FLTYPp - Parse File Type

Category: PARUTIL   File: JP&PR3::MS

Name:(S) FLTYPp - Parse File Type

Purpose:
Parse file type specifier

```
Entry:      D(A) = (AVMEME)
            D1 points into input stream at optional blanks
             preceding the alleged file type
            D0 points into output buffer

Exit:       Carry clear =>
             P=0
             Valid file type found
             Tokenized file type (2 bytes) written to
              output buffer
             D0 past the tokenization in the output buffer
             D1 advanced past the corr. text
```

Else error exit to PARERR with eFTYPE

Calls: FASCFD, OUT2TK, GNXTCR, STLXPT

Uses: A-C, R3, S10, D0,D1, P

Stk lvls: 4

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/15/82 | FH | Designed and coded. |
| 11/15/82 | S.W. | Hard-wired error exit |

## 16.24 FILEP - File Name Parse

Category: PARUTL File: JP&PR3::MS

Name:(S) FILEP - File Name Parse
Name:(S) LABELP - Label Reference Parse
Name:(S) FILEP1 - Literal File Name Parse
Name:(S) FILEP- - Subprogram Name Parse
Name:(S) FILEP+ - Label Declaration Parse
Name:(S) FILEP! - Literal File Name Parse

Purpose: Parses a file name or a label.
Depending on the entry point, it can allow string expressions and unquoted strings, or it can be limited to unquoted strings alone. However, only unquoted strings are checked for conformance to legal file name syntax, ie limited to 8 characters or less of letters and digits, starting with a letter.

FILEP and LABELP allow string expressions and unquoted strings. FILEP, however, checks an unquoted string to ensure it is not one of the reserved words (TO,ALL,KEYS,CARD,INTO). LABELP does not make this special check. These entry points are useful for file name and label reference; for example GOTO/GOSUB parse calls LABELP.

FILEP1, FILEP-, and FILEP+ are all useful entry points for parsing literals which must conform to file name standards; included in this category would be label declarations and subprogram names in SUB statements. These entry points do not check for file reserved words.

FILEP! is similar to FILEP+ above, except it can be set to allow less than eight characters.

```
Entry:    D1 points at input stream
          6 ENTRY POINTS:

          1) FILEP  - D1 points to optional blanks
                      preceding file name.
                      D(A) = (AVMEME)
                      D0 points to output buffer
          2) LABELP - Same as FILEP, except S10 must be
                      set to ignore file reserve words.
          3) FILEP1 - (LEXPTR) = address to restore input
                      pointer to; points to possible blanks
                      preceding file name.
          4) FILEP- - D1 at optional blanks preceding file
                      name.
          5) FILEP+ - D1 pointing at first character in
                      the file name.
          6) FILEP! - C(S)=#characters to allow - 1.

Exit:
          P=0
          S10=0 (all entries except FILEP+/FILEP!)
          S7=0 (all entries except LABELP/FILEP - see below)
          CARRY SET => IF S7=1: string expr found & output
                                NTOKEN done on following
                                data (LABELP/FILEP only)
                       IF S7=0: File name in A. D1 past
                                the last legal character.
                                C(S) set for WP write.

          CARRY CLR => IF S7=1  Reserve word found, token
                                output & in A(B),B(B).
                                D1 past the reserve word.
                                (FILEP only)
                       IF S7=0: Illegal 1st character. D1
                                pointing to the character.
```

```
                         R3(A)=D0 @ entry; R3(9-5)=D1 @ entry
                         Use D1C=R3 to restore D1

Calls:    CATCHR, RESPTR, r3expp (EXPPAR,R3=D1C),
          GNXTCR, BLANKC, WRDSCN,

Uses:     A-C,D(15-5),S0-S3,S7,S10,S11,D0,D1,R0-R3,FUNCD0
            (FILEP/LABELP entry)

          A,B(A),B(S),C,D(S),S1,S2,S7,S10,D1
            (FILEP1, FILEP-)

          FILEP+ entry uses everything FILEP1 uses except S10.
          FILEP! entry uses everything FILEP+ uses except S7.

Stk lvls:
 FILEP, LABELP           - 4
 all other entry points  - 3

History:

  Date      Programmer   Modification
 --------   ----------   ------------
 07/08/82   S.W.         Updated documentation
 07/27/82   S.W.         Now allow unquoted 'reserve
                          words' as file names, provided
                          they're followed by a colon.
 10/18/82   JP           Removed PCRD as reserve word
 11/23/82   JP           Clearing S10 on exit
```

## 16.25 CATCHR - Categorize Character

Category: PARUTIL     File: JP&PR3::MS

```
Name:(S) CATCHR  -  Categorize Character
Name:(S) CATCH+  -  Convert to Uppercase, Categorize Character
Name:(S) CATC++  -  Convert to Uppercase, Categorize Character

Purpose:
     Categorize character in A(B) as a
     digit or letter or special character.
```

CATCH+ and CATC++ entries convert a lowercase letter to uppercase before categorizing it.

```
Entry:
     3 entry points:
     1) CATCHR - A(B) = character to categorize
     2) CATCH+ - A(B) = character to categorize
                 P=0
     3) CATC++ - D1 points to character to categorize.
                 P=0

Exit:
     P=0
     A(B)=Character that was categorized
            (a letter gets converted to uppercase
             for CATC++ and CATCH+ entries)

     Carry set:
      Character is a digit or letter
      S1=1 iff it's a digit

     Carry clear:
      S2=1 iff special character:  * + - . / blank

Calls:      CONVUC,DRANGE,ARANGE,RANGE

Uses:       C(A), S1,S2       - CATCHR entry
            A(B),C(A), S1,S2  - CATC++,CATCH+ entries

Stk lvls:   2
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | JP | Modified documentation |
| 09/02/82 | S.W. | Changed RANGE call to DRANGE |

## 16.26 EXPPAR - Expression Parse

Category: PARUTL     File: SB&EXP::MS

Name:(S) EXPPAR - Expression Parse
Name:(S) EXPPLS - Expression Parse for Left of Equal Sign
Name:(S) EXPP10 - Expr Parse (specify start of parse stk)

Purpose:
Parse an expression and compile correct code for it
Also parses dummy array references
EXPPLS will stop parsing when a valid left-hand-side has been found.

Entry:
D0 is pointer to output stream
D1 is pointer to input stream
EXPPLS requires LeftSd(S7) to be set on entry.
EXPP10 requires LeftSd(S7) to be clear on entry and that D(A) be set to where the parse stack should start.

Exit:
```
If dummy array found then
Carry set
S0 -- 1 (invalid expression)
S1 -- Set by last NTOKEN
S2 -- Set by last NTOKEN
S3 -- 1 (not valid string expression)
S7 -- Clear if EXPPAR, unchanged if EXPPLS
D0 -- Points past code compiled for dummy array
D1 -- Points past first token not used in expression
A  -- Contains first token not used for dummy array
P  -- 0
XM -- 0

else
Carry clear
S0 -- 0 if valid expression found, 1 otherwise
S1 -- Set by last NTOKEN
S2 -- Set by last NTOKEN
S3 -- 0 if valid string expr. found, 1 otherwise
S7 -- Clear if EXPPAR, unchanged if EXPPLS
D0 -- Points past code compiled for expression
D1 -- Points past first token not used in expression
A  -- Contains first token not used in expression
XM -- Set iff expression is clearly a value expr
P  -- 0
D(A) -- (MTHSTK)
(PRMCNT) set non-zero if expression contained user FN
```

Calls: NTOKEN, OUT1TK, OUTNIB, OUTVAR, OUTLIT, OUTBYT, RANGE, CMPBNC, SCAN, DELET1, DELET2, LOOK, LOOK2, GNXCR+, OUTNBS, PARMCK, BOPCOM, CONCOM, PUSH-P, PUSH-3, INSRT1, RESPTR, CKLFSD

**Uses:** A,B,C,D(15-5),R0,R1,S0,S1,S2,S3,S7,S11,Carry
FUNCDO,PRMCNT(first nib)

**Stk lvls:** 3

**Detail:**

Internal representation of non-terminals is:

00 -- Primary
01 -- S-expr
02 -- Factor
03 -- Term
04 -- Sum
05 -- Relation
06 -- Conjunction
07 -- Expression
08 -- N-func-ref
09 -- S-func-ref
0A -- Substring ref
0C -- StartR (Reference expression)
0D -- StartS (Reference expression w/substring)
0E -- StartV (Value expression)

This parser is essentially a stack automaton. The stack builds from high memory down to lower memory. All stack elements are 2 bytes (4 nibs) in length although 2 or more elements may be used to hold extra information if needed.

If EXPPLS is called with LeftSd set, the parser will stop when it sees an reference expression or a substring reference expression followed by an equals sign.

Code is compiled from low memory toward high memory. The code pointer and the stack pointer are checked to make sure they never collide. MEMERR is called if there is such a collision.

Value expressions are indicated upon return by the XM bit. This is used to determine whether a parameter in a CALL statement is a reference or value parameter. It is also used to determine whether an expression would be a valid destination address for an assignment such as the INPUT statement.

## 16.27 P1-10 - Numeric Operand Found

Category: PARUTL File: SB&EXP::MS

```
Name:(S) P1-10   - Numeric Operand Found

Purpose:
     Point of reentry for numeric funny functions

Entry:
     P      = 0
     D0   = Output ptr (points past last nib of FFN code)
     D(A) = Stack pointer
     D1   = Input ptr (points past last char in FFN text,
               which is probably the closing paren)
     If a funny function is re-entering here, it should
     have set the XM bit to indicate that a value expression
     has been parsed.

NOTE:
     At this point a numeric operand has just been compiled.
     Funny functions are a special type of function that
     allow the expression parser to be extended to include
     that have special parse and/or execution requirements.
     See IDS for a complete description of how to implement
     a funny function.

History:

  Date      Programmer               Modification
 --------   ----------   ----------------------------------
 09/27/83   B.S.         Added documentation
```

## 16.28 SE1-10 - String Operand Found

Category: PARUTL File: SB&EXP::MS

Name:(S) SE1-10 - String Operand Found

Purpose:
    Point of reentry for string funny functions

Entry:
```
P     = 0
D0    = Output ptr (points past last nib of FFN code)
D(A)  = Stack pointer
D1    = Input ptr (points past last char in FFN text,
             which is probably the closing paren)
```
If a funny function is re-entering here, it should have set the XM bit to indicate that a value expression has been parsed.

NOTE:
    At this point a numeric operand has just been compiled. Funny functions are a special type of function that allow the expression parser to be extended to include that have special parse and/or execution requirements. See IDS for a complete description of how to implement a funny function.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/27/83 | B.S. | Added documentation |

## 16.29 ACCEPT - Funny function parse error reentry point

Category: PARUTL      File: SB&EXP::MS

Name:(S) ACCEPT - Funny function parse error reentry point

Purpose:
    This is the point where funny function parse routines should reenter if they detect an error

Entry:
    D(A) is stack pointer

A(W) set by last call to NTOKEN (flexible, doesn't matter if an error is being flagged)
D0 is output pointer (flexible, doesn't matter if an error is being flagged)
D1 is input pointer should point past first token not used in expressin (flexible, doesn't matter if an error is being flagged).
Status bits set by last NTOKEN call (or equivalent)

Exit:
See exit conditions for EXPRDC

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/01/83 | B.S. | Added documentation |

## 16.30 CONCOM - Compile a Numeric Constant

Category: PARUTL File: SB&EXP::MS

Name:(S) CONCOM - Compile a Numeric Constant

Purpose:
Compiles a numeric constant (Single digit, Long Int or Long Real)

Entry:
D0 is output pointer
A,B set by NTOKEN
D(A) = (AVMEME)
P = 0

Exit:
Carry clear if constant found, set otherwise
P = 0

Calls: DRANGE,OUT1TK,OUTNBS,RANGE

Uses.......

Inclusive: A(W),B(W),C(W)

Stk lvls: 1

```
+-----------------------------------------------+------------------+
|                                               |                  |
|  POLL   -  Poll Interface Descriptions        |  CHAPTER  17     |
|                                               |                  |
+-----------------------------------------------+------------------+
```

## 17.1 pCMPLX - Complex Number Operation Poll

Category: POLL    File: AB&FCN::MS

Name:(S) pCMPLX - Complex Number Operation Poll

Type: FPOLL

Purpose:
    Look for handler to perform complex operation:
    Function, Store or Recall.

Should poll be "Handled" (return with XM=0)?:
    Yes.

Meaning of "Handling" Poll (what does code do if handled?):
    Handler has performed complex operation. If poll is
    not handled, calling code errors out (eDATTY).

Entry conditions for handler (registers, ST, RAM, etc.):

```
     Carry set on entry.
     B[A] = Poll number.
     HEX mode.
     P=0.
     (FUNCD0) contains PC, pointing past token.
     (FUNCD1) contains 2-nibble token.
     (AVMEME) contains stack pointer.
     If token is a function token of one parameter (e.g.,
       SIN(Z)), then R1 = Real part of argument,
                    R0 = Imaginary part of argument.
     If token is a function token of two parameters (e.g.,
       Z*W), then R0 = Imaginary part of argument at top
                       of stack (second argument),
                  R1 = Real part of second argument.
                  R2 = Imaginary part of first argument.
                  R3 = Real part of first argument.
       If either argument is real, the imaginary part will
       be represented as 000000000000900.
```

If token is a comparison token, entry conditions are the same as for other two-parameter functions. The predicate can be obtained by looking at (PC). (or maybe (PC-1)?)
If token is cR->C, it means that a real value is being assigned to a variable whose type is not real, short or integer. The value to be assigned is at the top of the stack and the variable destination information occupies STMT scratch as set up by DEST routine.
If token is cC->C, it means that a value which is neither real or string is being assigned to some variable. The value to be assigned is at the top of the stack and the variable destination information occupies STMT scratch as set up by DEST routine.
If token is cRCL, it means that a complex number needs to be recalled (put on the stack).
R1[A] points at the value to be recalled.
D[S] is odd iff value is COMPLEX SHORT.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.
For functions and comparisons, result pushed on math stack (handler must do available memory check and error out if insufficient memory), complete with stack signature. D1 = stack pointer.
For store, no further exit conditions.
For recall (token = cRCL), value has been pushed on stack, D1 = stack pointer, B[A] = address of variable register, B[S] = E iff COMPLEX, F iff SHORT COMPLEX.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
1

What registers/RAM may be used if handled?:
A-D,D0,D1,P,R0-R4, function scratch RAM.

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P.

Envisioned application(s):
Extension of mainframe functions to complex arguments.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/20/83 | NM | Attempted to document |

## 17.2 pTRFMx - Poll for TRANSFORM Execution

Category: POLL      File: FH&TFM::MS

Name:(S) pTRFMx - Poll for TRANSFORM Execution

Type: FPOLL

Purpose:
Ask for an address to call for line-by-line transformation, and a similar address to call for line-by-line inverse transformation should that become necessary. The interface for these routines is defined in the Detail below.

Should poll be "Handled" (return with XM=0)?:
Yes.

Meaning of "Handling" Poll (what does code do if handled?):
The required information is present in the registers.

Entry conditions for handler (registers, ST, RAM, etc.):

```
RO(A)  =  Source file type
R1(A)  =  Destination file type
SO     =  Set if dest type # source type, means
          that a transform IS required (sTFREQ)
S5     =  Set if transform is in place (sTFINP)
/OPTN  =  TRANSFORM option set by extended TRANSFORM
          parse (or zero if mainframe parse), as in:
              TRANSFORM F INTO DATA FF,R
          where R means random I/O records (no overlap)
          See detail below for address of /OPTN
/PARM1,
/PARM2 =  TRANSFORM destination file create parameters
          set by extended TRANSFORM parse (or zero if
```

```
                mainframe parse), as in:
                    TRANSFORM F INTO DATA FF,R,50,128
                where /PARM1 =  50 = number of records,
                      /PARM2 = 128 = record size
                See detail below for addresses of /PARM1 and
                /PARM2
     P      =   0
     Carry  =   Set on entry
     B[A]   =   Poll number
     HEX mode
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

```
     R0(A)  = Address of handler routine which can read
                one line in from source and transform it.
                See Detail below for handler interface.
              ** 5 nibbles before this address is stored the
                relative address of handler routine which can
                read one line in from source and transform it
                in the INVERSE direction; 0 if none exists.
                Interface is same as that of a normal handler
                routine.  See Detail for handler interface.
              ** 10 nibbles before this address is stored
                the relative address of a routine which will
                finish the fully transformed destination file
                before it is closed (e.g., to chain a BASIC
                file in RAM before leaving it); 0 if no such
                routine is needed.  See Detail below for the
                finish-up routine interface.
     R0(S)  =   Copy code of destination file type
     S5     =   Entry condition (sTFINP)
     S0     =   1 if transform handler routines must be called
                  to perform transformation (even though source
                  and dest file types may be the same)
            =   0 if no transform handler routines need to be
                  called (source file and dest file type must
                  be the same)
     HEX mode
     XM     =   0
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

```
     Entry conditions preserved
     HEX mode
     XM     =   1
```

Available subroutine levels:

```
     3
```

What registers/RAM may be used if handled?:

```
     A-D, D0, D1, P, R0, R1, R2, S0
```

What registers/RAM may be used if not handled?:
    A-C, D[15-5] D0, D1, P, R2

Special memory/pointer considerations (are pointers funny?):
    No.

Envisioned application(s):
    Conversion between OEM file types and TEXT(LIF1) for
    purposes of listing and interchange.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/27/83 | FH | Added new documentation header. |

*************************************************************

DETAIL:

INTERFACE TO TRANSFORM HANDLER ROUTINE
--------------------------------------

Purpose:
    Read line from source file, transform it into destination
    type and leave it in output buffer. No messages should
    be directly issued by this routine.

Entry:
```
R4(15,14)       = Source FIB#
Input, output buffers collapsed to SYSEN
At least 150 bytes + LEEWAY available memory guaranteed
/LINE#          = 0 or previously returned BCD line #
/SFIB#          = Source FIB#
/OPT#           = Option from extended TRANSFORM statement
                  execution; 0 if from normal TRANSFORM
/PARM1,/PARM2 = Destination file create parameters from
                  extended TRANSFORM statement execution;
                  0 if from normal TRANSFORM
P               = 0
```

Exit:
```
OUTBS  @  Start of transformed line. If original line
          was copied into available memory start, OUTBS
          may point immediately after the original line.
          Must be collapsed to /SYSEN if fatal error.
AVMEMS @  End of transformed line unless fatal error.
          Must be collapsed to /SYSEN if fatal error.
S7     =  1 iff end of file found on source file (sEOF)
/LNLEN =  Full length in nibs of input line.  Unneeded
```

```
              if fatal error.
   /LINEM  =  BCD line number of current line.  Used in
              reporting error messages.  If sequential
              line number is to be used, set to 0.
   P       =  0
  Carry clear:  Successful transformation
  Carry set:    Error occurred
   C(3-0)  =  Error code
   C(S)    =  0 if error was fatal (unrecoverable).
           #  0 if error was recoverable.
```

Allowed to use.......
All CPU registers, S0-S11, S13, Statement and Function scratch RAM, SNAPBF, RSTKBF, /LNLEN, /LINEM, /FLAG, INBS, OUTBS, AVMEMS

Stk lvls: 6 (max)

************************************************************

INTERFACE TO TRANSFORM HANDLER FINISH-UP ROUTINE

Purpose:
To finish up the destination BASIC file after all TEXT lines have been transformed into BASIC. There are several cases to be dealt with:

End of a Dry Run (always out-of-place transform):
If the destination file is on an external medium, a first pass or "dry run" is conducted without creating the dest file, in order to determine its necessary data size. This routine calculates the needed parameters to create the file (see CRTF utility), and stores them in /PARM1 and /PARM2.

End of a Normal Run, Out-of-Place Transform:
If the destination file type requires a subheader or Implementation field, it must be properly initialized since CRTF stored a default value in it when the file was created (see CRTF utility). For example, if the destination is a BASIC file, hex value 00000000000F must be written to the header to indicate that the link chain heads have not been computed for this file (file has not been "chained"). If the file is in memory, the the proper link chain heads are computed and written to the subheader. File data, such as links between subprograms in BASIC files, may need to be updated.

End of a Normal Run, In-Place Transform:
If the source file type had a subheader or Implementation field, it must be removed. If the destination

file type requires a subheader or Implementation field, it must be inserted after the file header and set to the proper value. File data, such as links between subprograms in BASIC files, may need to be updated.

End of an Inverse Transformation (Always In-Place):
If the source file type had a subheader or Implementation field, it it is still there but may need to be updated to reflect the new state. File data, such as links between subprograms in BASIC files, may need to be updated.

```
Entry:
     Output buffer collapsed (OUTBS, AVMEMS point to SYSEN)
     R4(15-14) =  FIB# of destination file. Each line of the
                  source file (including EOF) has been read,
                  transformed, and written to the dest file.
                  The End of Data field in the FIB is set to
                  this new end of file and, if the dest file
                  is in memory, any excess nibs beyond the
                  end of file have been removed from the
                  file chain.  File is now rewound.
     S5      =  1 iff transformation is in place (sTFINP)
     S6      =  1 iff at end of inverse transformation
                  (sTFINV)
     S9      =  1 iff at end of dry run (sDRYRN)
     P       =  0

Exit:
     P       =  0
    Carry clear:
     No error
    Carry set:
     C(3-0) =  Error code (will be treated as fatal error,
               with no possibility of recovering dest file)

Uses.......
 Inclusive: May use any CPU register, S10

Stk lvls:   6 (max)

*************************************************************
```

TRFMBF FIELDS USED BY TRANSFORM ROUTINES

| Symbol | TRFMBF Offset | Size | Set by User* | Contents |
|---|---|---|---|---|
| /ERRCD | 0 | 4 | | Error code |
| /SFIB# | 4 | 2 | | Source FIB# |

| | | | | |
|---|---|---|---|---|
| /DFIB# | 6 | 2 | | Dest FIB# |
| /SFTYP | 8 | 4 | | Source file type |
| /DFTYP | 12 | 4 | | Dest file type |
| /COPYC | 16 | 1 | | Dest file copy code |
| /STAT | 17 | 4 | | Statuses during Xform |
| /DLEN | 21 | 5 | | Dest file len (DESTLEN) |
| /NUMLN | 26 | 5 | | Line count (NUMLINES) |
| /LINE# | 31 | 5 | H | Line # |
| /OPTN | 36 | 2 | X | Transform Option |
| /PARM1 | 38 | 5 | X | File Create Parameter 1 |
| /PARM2 | 43 | 5 | X | File Create Parameter 2 |
| /LNLEN | 48 | 6 | H | Input line length |
| /FLAG | 53 | 7 | H | Free for use by handler |

* Where 'H' indicates the field is set by the handler,
  and 'X' indicates the field is set by the extended
  TRANSFORM execution routine

## 17.3 pTIMR# - Poll Timer# > 3 for ON/OFF TIMER

Category: POLL File: JP&EXC::MS

Name:(S) pTIMR# - Poll Timer# > 3 for ON/OFF TIMER

Type: POLL

Purpose:
    Poll on Timer# > 3 for ON TIMER and OFF TIMER statements
    Allows Lex File to extend these statements to more than
      3 timers

Should poll be "Handled" (return with XM=0)?:
    No - If this poll is handled
            Return is through NXTSTM to continue
            statement/program execution.

Meaning of "Handling" Poll (what does code do if handled?):
    For ON TIMER: Set up the bookkeeping
                  Activate the appropriate timer
    For OFF TIMER: Deactivate the appropriate timer

```
Entry conditions for handler
     B[A] = Poll number (pTIMRM)
     HEX mode.
     P=0.

     A(A)   = TimerM > 3 in HEX
     DO     @ past TimerM expression
              If ON TIMER:  DO @ tCOMMA
                            Comma before timer interval
              If OFF TIMER: DO @ Remark or tEOL or t@

     PCADDR @ Statement length byte for statement
              (PCADDR) + 2 @ tON or tOFF

Normal exit conditions from handler if handled (ST, RAM,
registers, etc.):
     Return through NXTSTM to continue statement execution
     HEX mode.
     NOTE:
       If binary code invokes BASIC through CALL:
          PCADDR must be saved on the GOSUB stack before CALL
            Call PSHUPD
          and restored before NXTSTM is jumped to
            Call POPUPD

Normal exit conditions from handler if not handled:

     Carry clear
     HEX mode.
     XM=1.

Error exit conditions from handler:
     There is no error return from this poll

Available subroutine levels:
     --POLL handler is one level shallower than caller--
     6 levels available

What registers/RAM may be used if handled?:
     --A-D, DO, D1, P always available--
     This is a Statement Execute
     All RAM and registers allowed during Statement Execute

What registers/RAM may be used if not handled?:
     --A-D, DO, D1, P always available

What registers/RAM may be used if error exit:
     No error return allowed

Special considerations :
```

Tokenized form of statements:
ON TIMER:
Stmt Length, tON, Timer expression, tCOMMA,
Interval expression, tGOTO or tGOSUB, statement ident.

OFF TIMER:
Stmt Length, tOFF, Timer expression

To service a Timer when it goes off:
Respond to pSREQ poll to set sExcept
to indicate an Exception has occured
Respond to pExcept to actually service the timer

To execute Timer branch:
Use GOTO+ entry point after:
Setting sGOSUB (S3) if GOSUB
Reactivating Timer if GOTO
Setting sEXTGS (S5) to indicate external entry
Setting sXWORD (S9) for line# searching
Pushing Return Address (from Timer interrupt)
on stack
Tracing FROM line
(see ONTIMR for parallel code)

Envisioned application(s):
Extending Timers to an infinite number with a Lex file that allocates an I/O buffer to keep track of pending timers.

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/19/83 | JP | Added Poll |
| 04/19/83 | JP | Revised Poll documentation |

## 17.4 pCOPYx - Poll for COPY to external device

Category: POLL File: JP&EXC::MS

```
Name:(S) pCOPYx  -  Poll for COPY to external device

Type:       POLL

Purpose:
     Poll for COPY utility execute
     External source or destination file specifier found OR
     Destination device on PORT is of unknown type

Should poll be "Handled" (return with XM=0)?:
     Yes - If successful COPY occurs

Meaning of "Handling" Poll (what does code do if handled?):
     COPY source file to destination file on appropriate
     device

Entry conditions for handler (registers, ST, RAM, etc.):
     B[A] = Poll number (pCOPYx)
     HEX mode.
     P=0.

     If D(0) = External Device (D(0)>=8)
        sEXTDV = 1 (S0)
        sUNDEF = 1 (S1) if both filenames undefined = 0
        sDEST  = 0 (S3)
        A      = First 8 characters of source filename
                 Blanked filled
        R0(0-3)= Last 2 characters of source filename
                 Blanked filled if none
        D(A)   = Source device information from RDINFO
                 D(0)   = Device type
                 D(1-4)= Devices internal coding

                 HPIL used Device 8 --- see NOTE below
        R2     = Destination device info from RDINFO
        SAVSTK   holds source and destination information
                 (See Special Memory/Pointer Considerations)

     If D(0) = Unknown device type (1 < D(0) < 7)
        A      = First 8 characters of destination filename
                 (blank filled)
        R0(0-3)= Last two characters of destination filename
                 (blank filled)
        D(0)   = Device type
        D(1)   = Extender#
        D(2)   = PORT #
        STMTR0 = Start of source file
        SAVSTK   holds source and destination information
                 (See Special Memory/Pointer Considerations)

Normal exit conditions from handler if handled (ST, RAM,
```

registers, etc.):
Carry clear
HEX mode.
XM=0.
R1 = Start of file just copied if TO MAINFRAME
Source file copied to destination file on appropriate device.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.
COPY was not sucessful due to indicated Error Number

Available subroutine levels: 6
POLL handler is one level shallower than caller--
COPYu uses 6 levels; The handler must be able to
Return to POLL

NOTE:
HPIL uses Device Type=8
This device type is set in response to pFSPCx poll when the file specifier is being evaluated
Other device handlers must be assigned their special special device type by the Resource Allocation Czar (See HP-71 IMS Volume 1)
Respond to pFILXQ for non HPIL device to gain control of the File Specification execute

Devices on PORTS (ex: EEPROM) should use Device types between 3 and 6. This device type will be encoded in the ID of the module plugged into a PORT.
These Device types must be assigned by the Resource Allocation Czar (see HP-71 IMS Volume 1)

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0,R1,R2,S2,S3,S4,S5,S6,S7,S8,S9
Dont' use STMTD0 (saved status for CHAIN)

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
R1,S4,S5,S6,S7,S8,S9
Don't use STMTD0 (saved status for CHAIN)

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P
R0,R1,R2,S2,S3,S4,S5,S6,S7,S8,S9
Don't use STMTD0 (saved status for CHAIN)

Special memory/pointer considerations (are pointers funny?):

The SAVSTK area has been moved toward LOW memory due to the issuing of the POLL. Therefore, all offsets into the SAVSTK area must SUBTRACT the 1POLSV (62 decimal) from the SAVSTK pointer to access the file information.

```
Saved information:
SAVSTK-5 (- 1POLSV) = Source Device information  5 nibs
SAVSTK-25(- 1POLSV) = Source filename           20
SAVSTK-30(- 1POLSV) = Destination Device info    5
SAVSTK-50(- 1POLSV) = Destination filename      20
```

Envisioned application(s):

Allow COPY TO filename:TAPE
Allow COPY TO filename:PORT(1) where EEPROM in PORT(1)
Allow COPY TO a special device in a PORT
Allow COPY TO an external device NOT HPIL

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/19/82 | JP | Added documentation |
| 12/18/82 | JP | Combined pCOPYd with pCOPYx |
| 03/21/83 | JP | Changed entry conditions (STMTR0) |
| 05/11/83 | JP | Modified documentation |
| 08/11/83 | JP | Restricted STMTD0 usage |

## 17.5 pCURSR - Cursor Key with non BASIC file Poll

Category: POLL      File: JP&MEM::MS

Name:(S) pCURSR - Cursor Key with non BASIC file Poll

Type: FPOLL

Purpose:
Fast Poll to allow the Cursor Keys to be used with non BASIC files:
Cursor Up, Cursor Down, Cursor Top, Cursor Bottom

Should poll be "Handled" (return with XM=0)?:
No this is a TAKE OVER poll

Meaning of "Handling" Poll (what does code do if handled?):
Perform Cursor Key on file, return to MAIN30/MAINLP
See notes below.

Entry conditions for handler:
Carry set
B[A] = Poll number = pCURSR
HEX mode.
P=0.

| Type of Key: | Status: sCURUP (S2) | sCURBT (S3) |
|---|---|---|
| Cursor Bottom | 0 | 1 |
| Cursor Top | 1 | 0 |
| Cursor Up | 1 | 1 |
| Cursor Down | 0 | 0 |

Call RDCHD+ to get Filetype returned in R2

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

HEX mode
Perform Cursor Key on file
GOVLNG to MAIN30

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
S2 and S3 must be preserved

Available subroutine levels: 5
FPOLL handler is two levels deeper than caller
Invoked from CURSOR keys --- top level

NOTE:
The file type of the current file can be determined:
Call RDCHD+; R2 = File type on return

What registers/RAM may be used if handled?:

A-D, D0, D1, P always available--
Anything may be used: Status,R0-R4, SCRTCH...
CURRL (4 nibs) holds the current BASIC file line number
  This field may be used if CURRENT file is line numbered

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P always available
NOTE: D[A] is sacred in FPOLL!!--
R0-R4

Special memory/pointer considerations (are pointers funny?):
Take care when returning to the MAIN LOOP

MAIN30 is the return point for Cursor Keys in BASIC
  The line has been decompiled
  The prompt is sent and the display built (BLDDSP)
MAINLP is the return point if NOTHING is displayed
CR/LF with no delay has been sent (S-CRLF) prior
  to displaying the line.

Envisioned application(s):
Allow Cursor keys to display lines of a non BASIC file

The handler is responsible for maintaining the "Current file" position. Possibly an I/O Buffer can be used.

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/01/83 | JP | Added poll |
| 04/14/83 | JP | Revised documentation |
| 06/02/83 | SW | If null file, check for AUTO mode;<br>Not AUTO mode => goto MAINLP<br>AUTO mode => display curr line<br>(Before, went to MAINLP regardless) |

## 17.6 POLL - Poll LEX Files with Process Number

Category: POLL File: JP&POL::MS

Name:(S) POLL - Poll LEX Files with Process Number
Name:(S) POLLD+ - Poll LEX Files adjusting AVMEME in D(A)

Purpose:
Poll All LEX Files for Special Processing
Pass a Process #, and Parameters to each LEX File

POLLD+ entry used for routines needing to pass AVMEME in register D(A). This value is adjusted to reflect the save area used by POLL. Currently used by Parse and Decompile.

Entry:
POLLD+: Sets D(A) to what AVMEME will be during poll, then falls through to POLL
Used during Parse and Decompile

Example:

```
GOSBVL =AVS=DO      Set AVMEMS @ DO
GOSBVL =POLL        Issue Poll
CON(2) =pDEVCp      Device Parse Poll
GOSBVL =D=AVME      Reset D(A) @ AVMEME
GOC    ErrRtn       Error Return
?XM=0
GOYES  Handle       Handled by LEX File
...                 Not handled
```

POLL: Process# @ Calling Routine Return Address
Process# = CON(2)
HEX Mode

Example:

```
GOSBVL =POLL       Issue Poll
CON(2) =pFILXQ     File execute poll
GOC    ErrRtn      Error Return
?XM=0
GOYES  Handle      Handled by LEX file
...                Not handled
```

Assumes:
MTHSTK is active when called
Any routine polling with active stack must update AVMEME to top of stack pointer
Uses SNAPBF for temporary storage of registers
Uses SAVSTK to stack POLL information
Moves memory from FORSTK --> AVMEME before Save
Memory check w/o LEEWAY for Poll Save Area
Calling routine return address saved on GOSUB stack

```
            Will be updated if memory moves during poll

        Entry to LEX File Poll Routine:
          Carry Clear to indicate Normal Poll (see FPOLL)
          B(A) = Process#
                 B(B)   = Process #
                 B(2-4) = 0
          A,D,D0,D1 = Original Contents from Calling Routine
          R registers, Status are untouched by POLL
          R0-R3, status cannot be destroyed while identifying
            Process#.
          3 levels of subroutine stack saved
          One more stack level available than routine issuing
            the Poll

  Exit:
        Carry set
          Insufficient Memory to Issue Poll OR
          Error return / "Something Funny" from LEX File
          All registers & pointers preserved from LEX File
              EXCEPT A,B
          A,C have the same value on return
              The contents of C on return from LEX file are
              saved in A, then put back into C before return
              Allows LEX Files to return Error # in C(0-3)

          If not enough memory to save POLL info
              C(0-3) <-- eMEM

          A routine issuing POLL should check for CARRY
            If there was not enough memory to issue the poll
            this exception should be noted/indicated.

        Carry clear
          Look @ XM to determine if handled
          If XM=0
             Process has been handled by LEX File
             All regs & ptrs preserved from LEX File
                 EXCEPT B,C
             A is NOT destroyed

          If XM=1
             Process has NOT been handled
             Registers & pointers restored to Entry values
                 EXCEPT B,C
             A is not destroyed

          A POLL responder must return with CARRY CLEAR
            . if NOT error return or NOT handled

          A POLL responder must return with CARRY SET
```

```
           if Error return
           C(0-3) should hold the Error Number
              It will be saved in A, then restored to C

       Take over handler
         The poll responder does not return to the POLL routine
         This is allowed by certain individual polls as
             indicated in their documentation headers
         The handler MUST:
             GOSBVL =COLLAP  to collapse POLL Save info
             GOSBVL =POPUPD  to pop POLL issuer's rtn address

  Calls:      SALLOC, CRGJMP, FIRSAV, RESRTN, RESSVA,

              GLXPOL, SNAPSV, SNAPRS, MOVEU3, SNAPBF
              RSTK<R, PSHSTL, MEMCKL

  Uses:
   Exclusive: B,C,SNAPBF,P, SAVSTK, 2 levels in RSTKBF
   Inclusive: B,C,SNAPBF,P, SAVSTK, 2 levels in RSTKBF

  Stk lvls:   Preserves all levels
              POLL saves Calling Return Address on GOSUB
              stack so it will be updated if memory moves

  Detail:
           B(A)       = Process#
           B(B)       = Process#
           B(2-4)     = 0

           Save Stack:
                                                     Low Memory

  =1Ap     A                               16
  =1Dp     D                               16
  =1D1p    D1                               5
  =1D0p    D0                               5
  =1POL#p  Poll Number                      5
  =1RTN2p  Rtn Level 2                      5
  =1RTN3p  Rtn Level 3                      5
  =1BPOSp  Relative Position in LEX Buffer  5 @ SAVSTK - 5
                                              High Memory
           Length of Save Stack =          62 = 3E hex

           GOSUB Stack:

            ----------------
   GSBSTK -> |F| Rtn Addr 1 |                6
            |---------------|

           Return Type = F indicate an Update Address
```

```
Original contents of A,D,D0,D1 sent to each LEX File
Carry clear, XM=0 on call to each LEX File
R registers and status untouched

3 levels of subroutines saved
  Return Level 1 is saved on GOSUB stack as update
    address incase memory moves during a Poll
  Return Level 2 and 3 are saved in SAVSTK area
Information saved by POLL is stacked from SAVSTK
  toward LOW memory.
    AVMEME is adjusted above saved information.
    AVMEME is readjusted when POLL returns
This allows POLL to be called "recursively"

If "Error" Response from LEX File  (Carry Set)
   All registers & pointers left intact EXCEPT A,B
   A is used to save C during restore
   A,C have the returned value of C when rtn to Caller
   C(0-3) <-- Error number

If "normal" Response from LEX File (XM=0)
   All registers & pointers left intact EXCEPT C,B
   C has the value of A on return

If no LEX File respond (XM=1)
   Restore A,D,D0,D1 to entry values
   RTNSXM to Calling Routine
   C is NOT set to A

If LEX File wishes Poll to continue to others
   Carry must be clear, XM=1
   If "non-original" contents of A,D,D0,D1 are
   to be sent to other LEX Files, the information
   above SAVSTK can be altered by the LEX File
```

**Algorithm:**

```
Save A,D,D0,D1,Rtn Lvl 1 temporary in SNAPBF (SNAPSV)
If not enough memory to save info            (SALLOC)
   Restore saved registers and pointers      (SNAPRS)
   Adjust return address past Process #
   C <--- Error Number                       (eMEM)
   RTNSC
Save Return Level 2 & 3 cus SALLOC uses 2    (D,D0)
Allocate SAVSTK area                         (SALLOC)
Restore Rtn Lvl 3,2 to stack
Move temporary save info to SAVSTK           (MOVEU3)
Read Return Level 1, read process# @ Rtn address
Write process # over Return Level 1 location
Adjust Return Level 1 past Process #, saving in A(A)
Save Rtn Levels 2,3 in SAVSTK
```

```
        Save 2 levels in RSTK Buffer                  (R<RSTK)
        Push Rtn Level1 on GOSUB stack to be updated (PSHSTL)
          Make sure LEEWAY is NOT checked when pushing
        Restore Return Levels                         (RSTK<R)
        Initialize Relative Offset into LEX Buffer to 0
   1:   Get LEX File POLL address                     (GLXPOL)
        If LEX file POLL address                  (Carry Clear)
           Save updated Rel. Position into Buffer in SAVSTK
           Push LEX file jump address on stack
           Restore registers,pointers,process#        (RESSVA)
           Pop LEX file jump address
           Clear XM flag
           Gosub to LEX File Poll Routine             (CRGJMP)
           Clear B(S) to save carry from LEX file return
           If Carry set  (Error response from LEX File)
              Set A=C to preserve C during restore
              goto 2;
           If LEX File responded                     (XM=0)
              Set B(S) = 1
              goto 2;
           else        (No response)                 (XM=1)
              Restore Relative Position in LEX Buffer
              goto 1; (Continue polling)
     2:    Save current A,D,D0,D1 in SNAPBF and
           Restore return lvls 2,3; Release SAVSTK  (RESRTN)
           Restore current A,D,D0,D1,Rtn Lvl 1      (SNAPRS)
           Push Rtn Lvl 1 back on stack
           Set C=A
           Return indicating carry from LEX File    (B(S))
    else
          (No more LEX Files in LEXBUF)
        Restore A,D,D0,D1 from SAVSTK               (RESSVA)
        Save A->D1,Restore Rtn Lvls, Release SAVSTK (RESRTN)
        Restore A->D1,Rtn1 from SNAPBF              (SNAPRS)
        Push Rtn Lvl 1 back on stack
        RTNSXM
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | JP | Modified documentation |
| 10/14/82 | JP | No Leeway check when allocate Save Area |
| 10/14/82 | JP | Rewrote to interface to SNAPBF |
| 01/31/83 | JP | After SALLOC, restore RSTK from D0 |
| 02/05/83 | JP | Rewrote to save Rtn Addrs on GSBSTK |
| 02/05/83 | JP | Set XM=1 if Carry set/Error return |
| 02/15/83 | JP | No Leeway Check when PSHSTK called |
| 03/01/83 | JP | Added lPOLra to D(A) in POLLD+ entry |
| 06/01/83 | JP | Added MEMCHK of (lPOLSV + lRTNADR) |

## 17.7 FPOLL - Fast Poll all LEX files with Process #

Category: POLL File: JP&POL::MS

```
Name:(S) FPOLL   - Fast Poll all LEX files with Process #

Purpose:
     Poll LEX Files FAST, nothing is saved

Entry:
     Process# @ Calling Routine Return Address
     Process# = CON(2)

     Example:

     GOSBVL =FPOLL
     CON(2) =pMNLP      Main Loop Fast Poll

     At entry to LEX File POLL routine:
        Carry Set to indicate FAST Poll
        B(A) = Process#
               B(B)   = Process#
               B(2-4) = 0
        D(A) = Relative Position in LEX Buffer
               Must be preserved ALWAYS !!!!!
                 If the Poll Handler is responding
                 and handling the poll such that
                 the Poll will stop: D may be used.
        R0,R1,R2,R3 intact
        A LEX File may not destroy R0-R3 while determining wheth??
          to respond.  Individual POLL routines must be checked ??
          register usage when responding.

        Stack levels are 2 deeper than caller

Exit:
     P=0
        Assuming no LEX File has set P
     XM=0
        Process has been handled by LEX File

     XM=1
```

```
            No response to Poll

         If a LEX File wants the Poll to continue to others
            XM=1 on return
            Registers needed to be passed to other LEX Files
            must be preserved !!!!!

         Typically a fast poll must continue to ALL LEX files
```

```
Calls:      GLXPOL

Uses.......
 Exclusive: A(A),B(A),C(A),D(A),D0,D1
 Inclusive: A(A),B(A),C(A),D(A),D0,D1

 D(A) cannot be destroyed by any LEX File
 R0-R3, status must remain intact while determining if
   reponding to poll.

Stk lvls:   2

Algorithm:

     Initialize Relative Offset to LEX Buffer to 0
1:   Get LEX File Poll Address              (GLXPOL)
     If LEX File Poll Address        (Carry clear)
        Save Relative position in LEX Buffer   (D)
        Retrieve Process #
        Clear XM
        Gosub to LEX File's Poll routine w/ Carry set
        If LEX file did not respond         (XM=1)
           Restore relative position in LEX buffer
           goto 1;
        else
           Adjust Return Address past Process#
           RTN
     else
        Adjust Return Address past Process#
        RTNSXM

History:

  Date      Programmer                Modification
--------    ----------    -----------------------------------
07/13/82    JP            Modified documentation
06/09/82    JP            Packed out CRGJMP/set carry:FPOL40
```

## 17.8 pPARSE - Parse Take Over Poll

Category: POLL  File: JP&PR1::MS

Name:(S) pPARSE - Parse Take Over Poll

Type: FPOLL

Purpose:
Parse take-over to allow a LEX file to parse an input line as other than BASIC

Should poll be "Handled"
Don't worry about XM, since if handled, there's no return

Meaning of "Handling" Poll (what does code do if handled?):
Parses line, acts accordingly, returns to MAINLP.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set
B[A] = Poll number.
HEX mode.
P=0.
INBS points to input line

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Return to MAINLP

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
5

NOTE:

--SCRATCH RAM TO CONSIDER BELOW:--
--STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
--LEXPTR.--

What registers/RAM may be used if handled?:
A-D, D0, D1, P

R0-R4, S0-S11, STMT/FN scratch

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P
R0-R4, S0-S11, STMT/FN scratch

Special memory/pointer considerations (are pointers funny?):
No

Envisioned application(s):
'Auto Comment'
Alternate language parse (in conjunction with pEDIT)

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/15/83 | S.W. | Added poll |

## 17.9 pFSPCp - POLL for File Specifer Parse

Category: POLL File: JP&PR3::MS

Name:(S) pFSPCp - POLL for File Specifer Parse

Type: POLL

Purpose:
POLL for File Specification Parse.
Unquoted string is not a legal mainframe file name.

Either:
a) the 1st character isn't a letter or colon
(device specifier starting with a character
other than a colon)
OR
b) Valid file name is followed by a
"non-terminating" character, ie one in the
ASCII range of "." to "z" (with the exception
of ":" and "@"). The character may be a part
of the file name (as in a file name with more

than 8 characters or a file name that starts with a letter, but contains a character other than a letter/digit) OR it may be a delimiter between the file name and the device specifier.

Should poll be "Handled" (return with XM=0)?:
Yes - If file specificer is recognized

Meaning of "Handling" Poll (what does code do if handled?):
Parse and tokenize file specification analogous to the mainframe tokenization:

Filename over 8 characters or a file name with a non letter/digit character embedded in it.
tLITRL <ascii file name>
Ex: ABC_X or ABCDEFGH

Filespec beginning with character other than a letter or a colon:
tCOLON tLITRL <ascii file specifier>
Ex: /WAND

In the first case above, if the valid file name is immediately followed by a 'non-terminating' character not recognized by the responder (letter in the ascii range '.' to 'z' not including letters/digits or '@'), a poll to pDEVCp may be appropriate.
tLITRL <ascii file name> tCOLON tLITRL <ascii device>
Ex: ABC_X.DISC or ABCDEFGHI/DISC

Entry conditions for handler (registers, ST, RAM, etc.):
S4=S10=S7=0
B[A] = Poll number (pFSPCp)
HEX mode.
P=0.
D(A) = (AVMEME)
D1 @ Start of File specification
(D1 points past any preceding blanks)
D0 @ Position in Output Buffer to begin output of File specification
R3(5-9)=D1 @ start of file specfication input
R3(A) = D0 @ start of file specifcation output

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
P=0
S4=S7=S10=0
HEX mode.
XM=0.

File specification is accepted and output @ D0
See NOTE below
D0 past last token of file specificaiton
D1 past file specification in input buffer
R3 intact from entry

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
P=0
HEX mode.
S4=S7=S10=0
XM=1.
R3 intact from entry

Error exit conditions from handler (POLL only):
P=0
Carry set.
HEX mode.
S7=S10=0
R3 intact from entry

Available subroutine levels: 6
POLL handler is one level shallower than caller--
FSPECp uses 5; therefore Handler can use 6

What registers/RAM may be used if handled?:
A-D, D0, D1
R0,R1,R2,R4
STMTD1, S-R0-0, S-R0-1, SCRTCH, all of function scratch

What registers/RAM may be used if not handled?:
A-D, D0, D1
R0,R1,R2,R4
STMTD1, S-R0-0, S-R0-1, SCRTCH, all of function scratch

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1
R0,R1,R2,R4
STMTD1, S-R0-0, S-R0-1, SCRTCH, all of function scratch

Special memory/pointer considerations (are pointers funny?):
No.

Detail:
If HPIL is plugged in, it will answer this poll. Therefore, any other LEX file answering this poll should use an anologous tokenization scheme for the file name/device specifier tokenization so that file specifier execution works properly.

To get control during execution, respond to pFILXQ. This is a poll that is NOT answered by HPIL. If this poll is not answered by another LEX file, another poll is sent out later which HPIL answers.

For more information on how HPIL tokenizes devices, see the Detail portion of the documentaion on pDEVCp.

Envisioned application(s):

Handle external file specifiers

```
A123456789
A123456789/DISC
A123.WAND
AB_X.DISC
/WAND
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | JP | Added documentation |
| 05/07/83 | JP | Modified documentation |

## 17.10 pDEVCp - Poll for Device Specifier Parse

Category: POLL File: JP&PR3::MS

Name:(S) pDEVCp - Poll for Device Specifier Parse

Type: POLL

Purpose:

POLL for unrecognized device specifier following ":". If a file name preceded the colon, it has already been written to the output buffer.

Should poll be "Handled" (return with XM=0)?:

Yes if Device specifier is recognized by handler.

Meaning of "Handling" Poll (what does code do if handled?):

Parse and output tokenized form of device specifier

See detail below

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number (pDEVCp)
HEX mode.
P=0.
S4=S7=S10=0
D1 past colon in file specification
If a filename was specified, its tokenization was written to the output buffer & D0 points past the last character of the filename
D(A) = (AVMEME)
R3(A) = D0 @ start of tokenized filespec in output buffer
R3(9-5) = D1 @ start of file spec in input buffer

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
S4=S7=S10=0
P=0
HEX mode.
XM=0.
Tokenized device written to output buffer
D0 points past the tokenization
D1 is past the corresponding text in the input buffer
R3 preserved from entry

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
S4=S7=S10=0
P=0
HEX mode.
XM=1.
Tokenized device specifier written to output buffer
D0 points past tokenization
D1 points past device specifier in input buffer
R3 preserved from entry

Error exit conditions from handler (POLL only):
Carry set.
HEX mode.
P=0
S10=0
R3 preserved from entry

Available subroutine levels: 6
FSPECp used 5 levels

NOTE:

If HPIL is plugged in, it responds to this poll; it accepts ALL device specifiers following the colon. Therefore, all LEX files should tokenize device specifiers in the same manner so that during execution the filespec execution routines work properly.

Respond to pFILXQ to gain control at execution. HPIL does not respond to pFILXQ.

Detail:

HPIL tokenizes devices as follows:

```
device word:  (:TAPE)
 tCOLON tLITRL <ascii device word>

accessory ID: (:%32)
 tCOLON t% <expr> [ tCOLON <expr> ] [ tSEMIC <expr> ]

volume label: (.LABEL1)
 tCOLON tSEMIC <literal up to 6 chars> [ tSEMIC <expr> ]
                                          Loop #
address:      (:1)
 tCOLON <expr> [ tSEMIC <expr> ]
        (seq#)       (loop#)

assign word:  (:TV)
 tCOLON tLITRL <assign word> [ tSEMIC <expr> ]

"*"           (*)
 tCOLON t*
```

What registers/RAM may be used if handled?:

A-D, D0, D1
R0,R1,R2,R4
STMTD1, S-R0-0,S-R0-1, SCRTCH
All of function scratch

What registers/RAM may be used if not handled?:

A-D, D0, D1, P
R0,R1,R2,R4
STMTD1, S-R0-0,S-R0-1, SCRTCH
All of function scratch

What registers/RAM may be used if error exit?:

A-D, D0, D1, P
R0,R1,R2,R4
STMTD1, S-R0-0,S-R0-1, SCRTCH
All of function scratch

Special memory/pointer considerations (are pointers funny?):

No

Envisioned application(s):
ABC:TAPE
:TAPE

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/19/82 | JP | Added documenation |
| 05/08/83 | JP | Modified documentation |

## 17.11 pRUNft - Poll on RUN with unknown filetype

Category: POLL File: JP&SYS::MS

Name:(S) pRUNft - Poll on RUN with unknown filetype

Type: POLL

Purpose:
Poll on RUN with file of "unknown" filetype
Filetype is NOT BASIC or Binary

Should poll be "Handled" (return with XM=0)?:
No - this is a take-over Poll

Meaning of "Handling" Poll (what does code do if handled?):
Take over the RUN execution of the file

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number (pRUNft)
HEX mode.
P=0.
A(A) = File type (HEX)
R2(A)= File type (HEX)
D1 @ File length (offset) field in file header
5 nibble read @ D1 = Offset to next file
Current D1+ offset --> Next file start

```
            CAT file length =
              File offset value - Offset to data of file

      sCONT  = 1 if CONT            (S10)
      sCONTK = 1 if CONT/RUN key    (S9)
      sCHAIN = 1 if CHAIN statement (S11)
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

```
     Carry clear
     HEX mode.
     XM=0.
     GOVLNG to MAIN05
     or
     GOVLNG to BSCEXT to exit the BASIC interpreter
       This is done by BASIC and Binary programs
       Filetype read, Buffers are flushed
       A fast poll is issued: pBSCex

     See NOTE below
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

```
     Carry clear
     HEX mode.
     XM=1.
     Preserve Status
```

Error exit conditions from handler

```
     Error returns are ignored.
     If the POLL returns:
        If carry set ---> "eMEM" from POLL
           else      ---> "eFTYPE" from RUN
```

Available subroutine levels: 7

```
     --POLL handler is one level shallower than caller--
     RUN is a top level statement/command
```

NOTE:

```
     Any Lex File running a non BASIC file should:
         Clear the SUSP annunciator
         Set the PRGM annunciator    (see SFGPGM)
         Collapse all BASIC stacks   (see CLPSTK)
         Set CURRST, CURREN @ file   (see EDIT20)

     Responder should issue a pRUNnB (RUN non BASIC) Poll
         The mainframe issues a "pRUNnB" when
         running a Binary file with the filetype in A(A)
```

What registers/RAM may be used if handled?:

--A-D, D0, D1, P always available--
RUN is in complete control at this point

What registers/RAM may be used if not handled?:
--A-D, D0, D1, P always available
Global status (S12-S15) are sacred

What registers/RAM may be used if error exit?:
No error exit allowed

Envisioned application(s):
Extend RUN statement to handle other file types

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/16/82 | JP | Added Poll |
| 01/16/83 | JP | Check carry from Poll |
| 04/19/83 | JP | Updated Documentation |
| 04/24/83 | JP | Changed entry conditions |

## 17.12 pRUNnB - Poll before non BASIC file exec (BIN)

Category: POLL File: JP&SYS::MS

Name:(S) pRUNnB - Poll before non BASIC file exec (BIN)

Type: POLL

Purpose:
Poll before starting execution or continuing execution of a non BASIC file

Poll before running a BINARY file

Should poll be "Handled" (return with XM=0)?:
No - let this poll go to all other Lex files

Meaning of "Handling" Poll (what does code do if handled?):

```
        Perform any "system" or special initialization needed
        before Binary file is executed.


   Entry conditions for handler (registers, ST, RAM, etc.):
        B[A] = Poll number (pRUNnB)
        HEX mode.
        P=0.

        R2(A)= File type of file to execute
        D0   @ Start of code to execute
        BASIC stacks have been collapsed

        General purpose poll

        Mainframe poll will ALWAYS be Binary exeucute
          R2(A)= fBIN
          D0   @ Start of binary file
          If sCONT (S10) = 1
             Executing a CONT statement
             CNTADR is always zero, unless a Lex file has
               set this  (see pBSCex Poll)
             Therefore, CONT is always a RUN
          If sCONTK (S9) = 1
             RUN or CONTK hit
          If sCHAIN (S11)= 1
             CHAIN statement
          SUSP annunciator has been cleared
          PRGM annuciator and PgmRun flag have been set
          Current file pointers @ Binary file

   Normal exit conditions from handler if handled

        This poll should NOT be indicated as handled so
        other Lex files can "set-up" before execution.

        If Lex file wants to be the ONLY Lex file to handle:
        then:

        Carry clear
        HEX mode.
        XM=0.
        D0 must be PRESERVED!!!
        Preserve S3

   Normal exit conditions from handler if not handled

        Carry clear
        HEX mode.
        XM=1.
        Status intact: sCONT(10), sCONTK(S9), sCHAIN(S11),S3
```

Error exit conditions from handler:
     Error return has no meaning ---

Available subroutine levels:
     --POLL handler is one level shallower than caller--

     This is a RUN... therefore all levels (6) available
     Must be able to return to POLL routine

NOTE:
  See Special memory/pointer considerations
What registers/RAM may be used if handled?:
     --A-D, D0, D1, P always available--
     --R0-R4, scratch RAM?--

What registers/RAM may be used if not handled?:
     --A-D, D0, D1, P always available
     --R0-R4, scratch RAM?--

What registers/RAM may be used if error exit (POLL only)?:
     No error exit allowed

Special memory/pointer considerations (are pointers funny?):

Binary Files will always be RUN/CONT from the start of
  the file... it is "impossible" to systematically
  return a meaningful CONTinue address through the BASIC
  loop. If a Binary file wishes to implement CONT...
  it should respond to the pBSCex poll:
     If current filetype is Binary and sERROR=1
        Update CNTADR @ Binary code to CONTinue at
        Set the SUSP annunciator (SFLAGs)

If a Poll Handler intends to CALL BASIC from within:
  Return Address to Poll must be saved on the GOSUB stack
    (Use PSHUPD and POPUPD)
  The FORSTK must be adjust OVER the Poll Save information
    before the CALL and readjust after. CALL uses the
    FORSTK pointer to save information and if FORSTK is
    not adjusted, Poll Information is overwritten

| Before CALL | | After CALL | |
|---|---|---|---|
| D1=(5) | =FORSTK | " | |
| A=DAT1 | A | " | |
| C=0 | A | " | |
| LC(2) | =lPOLSV | " | |
| A=A-C | A | A=A+C | A |
| DAT1=A | A | DAT1=A | A |

Envisioned application(s):

Set up I/O buffers before Binary execution or some type of non-BASIC file.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/16/82 | JP | Added Poll |
| 01/16/83 | JP | Generalized Poll for any file type |
| 04/22/83 | JP | Upgraded documentation |
| 04/25/83 | JP | Pass File type in R2 instead of A |

## 17.13 pBSCen - Poll entering BASIC interpreter

Category: POLL File: JP&SYS::MS

Name:(S) pBSCen - Poll entering BASIC interpreter

Type: FPOLL

Purpose:
Fast poll when entering BASIC interpreter

Should poll be "Handled" (return with XM=0)?:
No - Either this poll is a TAKE OVER poll
or it should continue to ALL LEX files

Meaning of "Handling" Poll (what does code do if handled?):
Take over BASIC interpreting
Set up information/buffers/flags before execution
begins, then let Poll continue

Entry conditions for handler (registers, ST, RAM, etc.):
--Carry set on entry iff fastpoll--
B[A] = Poll number (pBSCen)
HEX mode.
P=0.

If PgmRun (S13)
Program about to be executed (RUN/CONT/SST)
If NoCont (S14)

```
            SST (Single stepping)
         If sCONT (S10)
            Continue
         If sCONTK (S9)
            CONT or RUN Key
       R0 @ EOL or "@" prior to statement to execute
     else
       Keyboard execution from Statement Buffer
       R0 @ Statement length byte of statement
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

```
     HEX mode.
     This poll should never be "normally" handled
     Either the LEX file takes over or allows other
     LEX files to respond.
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

```
     HEX mode.
     XM=1.
     Global status intact
     Do not use S3
     R0 must be PRESERVED !!!!!
```

Available subroutine levels:

```
     --FPOLL handler is two levels deeper than caller--
     This is a "top level" poll --- 6 levels available--
     Must be able to return to Poll routine
```

NOTE:

```
  GOSUB,CALL,FNx invoked from the keyboard will appear
  as Keyboard Execute.  The PgmRun flag will be clear.

  Program execution will begin with NO indication.

  For CALL: pCALSV polls when CALL execute begins
      FNx:  pFNIN  polls when FNx executes begins
```

```
Binary Files will always be RUN/CONT from the start of
  the file... it is "impossible" to systematically
  return a meaningful CONTinue address through the BASIC
  loop.  If a Binary file wishes to implement CONT...
  it should respond to the pBSCex poll:
     If current filetype is Binary and PgmRun=1
        Update CNTADR @ Binary code to CONTinue at
        Set the SUSP annunciator (SFLAGs)
```

What registers/RAM may be used if not handled?:

```
     --A-C, D[15-5] D0, D1, P always available (FPOLL only)--
```

--NOTE: D[A] is sacred in FPOLL!!--
--R1-R4, ST (low 12), scratch RAM--
This is a top level poll ... nothing else is going on

Envisioned application(s):
Implement BREAKPOINT capability within program:
Set sExcept at entering to allow checking after each statement

Indicator to FORTH/VISICALC type applications that BASIC has been invoked.

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/16/83 | JP | Added Poll |
| 04/23/83 | JP | Updated/revised documentation |

## 17.14 pBSCex - Poll to Exit BASIC Interpreter

Category: POLL    File: JP&SYS::MS

Name:(S) pBSCex - Poll to Exit BASIC Interpreter

Type: FPOLL

Purpose:
Fast Poll when Exiting the BASIC interpreter
Indicates program/statement execution is stopping
Caused from:
  End of line of statement execution
  Program ENDing or STOPping
  Halt from: ATTN key, SST, PAUSE, Error
  Ending a Binary program
System is about to return to MAINLP

Should poll be "Handled" (return with XM=0)?:
No - This poll should never be "handled"
Either the LEX file "TAKES OVER"

```
        or responds "not handled" so other LEX files may
        respond.

   Meaning of "Handling" Poll (what does code do if handled?):
        Clear/update information
        If TAKE-OVER ---> gain control after BASIC execute
                          before we go back to MAIN LOOP

   Entry conditions for handler (registers, ST, RAM, etc.):
        --Carry set on entry iff fastpoll--
        B[A] = Poll number (pBSCex)
        R2(A)= Filetype
        HEX mode.
        P=0.
        Math stacks have been collapsed
        Exceptions are checked PRIOR to this poll
          See pExcpt Poll
        If not Error Exit (not sERROR)
          Buffers have been flushed
        If NoCont (S14):
           If Program was running (and BASIC file)
              SUSP is lit
              CNTADR updated
              CURRL updated
           Halting due to one of the following:
              ATTN Key   (ATNFLG RAM is non-zero)
              END/STOP or end of program  (sENDx=1)  (S1)
              Error      (sERROR=1) (S0)
              SST
              PAUSE
              END(DEF), END(SUB), RETURN from keyboard
          (Error Exits can be trapped with pERROR,pWARN polls)
        If not NoCont (S14=0)
           If PgmRun (S13) --> Program was running
           sENDx=1 if STOP/END statement


   NOTE:

        GOSUB, CALL, FNx invoked from the keyboard
        will enter and exit as Keyboard Execute...
        The PgmRun flag will NOT be set!!!!
        RETURN,ENDSUB,ENDDEF clear PgmRun before exiting

        For CALL: pCALRS polls when CALL is ending
        For FNx:  pFNOUT polls when FNx  is ending

   Normal exit conditions from handler if not handled (ST, RAM,
   registers, etc.):
        HEX mode.
        XM=1.
```

Preserve sENDx, sERROR, PgmRun, NoCont, ATNFLG

Available subroutine levels: 6
--FPOLL handler is two levels deeper than caller--
This is a top level Poll --- 6 levels available, unless TAKING OVER... then 7

What registers/RAM may be used if not handled?:
--A-C, D[15-5] D0, D1, P always available (FPOLL only)--
--NOTE: D[A] is sacred in FPOLL!!--
--R0-R4, scratch RAM--

Special Considerations:
Binary Files may return through this exit point

A Binary file taking an error exit through the mainframe can "SUSPend" a binary program by setting CNTADR at the address to continue at within the file and setting the SUSP annunciator.

Envisioned application(s):
Allow a LEX file to gain control after BASIC execution

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/20/82 | JP | Added poll/documentation |
| 01/16/82 | JP | Modified poll |
| 04/23/82 | JP | Revised/updated documentation |
| 04/25/82 | JP | Pass filetype in R2 |

## 17.15 pExcpt - Poll on Exception after Stmt Execute

Category: POLL File: JP&SYS::MS

Name:(S) pExcpt - Poll on Exception after Stmt Execute

Type: FPOLL

Purpose:

Fast poll to indicate Exception has occured
Allows servicing of Exceptions at the end of each statement execute.
The Exception flag (Except (S12)) must have been set in response to pSREQ or prior to re-entry to BASIC loop (@ RUNRT1)

Should poll be "Handled" (return with XM=0)?:
NO - This poll must continue to all LEX files

Meaning of "Handling" Poll (what does code do if handled?):
You can process YOUR exception, but indicate the Poll was NOT handled.

Entry conditions for handler (registers, ST, RAM, etc.):
--Carry set on entry iff fastpoll--
B[A] = Poll number (pExcpt)
HEX mode.
P=0.
Except (S12) = 0 from Mainframe
  Subsequent "responders" may set this to cause Except next time around.
PgmRun = 1
  If program running
NoCont = 1 (S14)
  If execution NOT to continue
    Caused from SST, PAUSE, END/STOP, END(DEF), END(SUB), RETURN from Keyboard
The attention key HAS NOT been checked, yet
  ATNFLG RAM location#0 if ATTN Key hit
  The ATTN Key will cause program execution to stop
DSPSTA (RAM) holds S0-S11
  sENDx = 1 (S0) if END/STOP or End of Program
RSTK(3) Third Return Stack Level (0,1,2)
  = D0 setting from RUNRTN
  Points at EOL or @ following statement just executed.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Response to this poll should NEVER indicate "handled

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
S12-S15 must be preserved
Stack levels: 0,1,2 preserved

Available subroutine levels: 4
--FPOLL handler is two levels deeper than caller--

This poll is issued from the TOP level
But the Current D0 is 3rd level on stack
This value cannot be lost; nor the return address to FPOLL
Preserve levels: 0,1,2

NOTE:
Error Exit to BASIC loop does NOT check exceptions
Low status are restored from DSPSTA at the End of the Poll.
Math stack has been collapsed
ATTN key has been checked---causing NoCont to set (S14)
Timers (1-3) will be checked after the pExcept poll

What registers/RAM may be used if handled?:
N/A

What registers/RAM may be used if not handled?:
--A-C, D[15-5] D0, D1, P always available (FPOLL only)--
--NOTE: D[A] is sacred in FPOLL!!--
--R0-R4, S0-S11

Envisioned application(s):
Service external alarms/timers
Service ON INTR statement
Implement BREAKPOINT capability in BASIC
Checking next statement to execute for Breakpoint
Setting Except (S12) so pExcpt will occur at the end of the next statement
Servicing ON TIMER# > 3 from an extended statement

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/16/83 | JP | Added poll |
| 04/04/83 | JP | Status saved/restored in DSPSTA |
| 05/07/83 | JP | Updated documentation header |
| 05/18/83 | JP | Attn Key not checked before poll |

## 17.16 pZERPG - Poll to zero program information

Category: POLL File: JP&SYS::MS

Name:(S) pZERPG - Poll to zero program information

Type: FPOLL

Purpose:

Fast poll to allow future statements to zero addresses and RAM associated with extending a statement, adding a statement or application.

This poll issued when zero program information due to an END, ENDALL, EDIT, Program Edit....

Issued from CLRSTK/CLPSTK/ZERPGM routine.

Should poll be "Handled" (return with XM=0)?:

No - This poll should continue to ALL Lex files

Meaning of "Handling" Poll (what does code do if handled?):

Zero appropriate RAM / addresses associated with statement or application.

Entry conditions for handler:

Carry set on entry
B[A] = Poll number (pZERPG)
HEX mode.
P=0.

BASIC stacks have been collapsed to appropriate level.
CONT, ON ERROR, ON ERROR GOSUB, ON INTR, ON TIMER statement addresses have been zeroed
Timer alarm RAM has be zeroed
SUSP annunciator/flag has be cleared

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

This poll should never be "handled".
Always return with XM=1

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

HEX mode.

XM=1.
R registers intact. Status intact

Available subroutine levels:
--FPOLL handler is two levels deeper than caller--
The invoking routine (CLRSTK/CLPSTK/ZERPGM) uses 3 lvls
Therefore, a handler may use ONLY 1 lvl.

Use RSTK<R to save 3 levels in RSTKBF circular buffer
Use R<RSTK to restore 3 levels

What registers/RAM may be used if not handled?:
--A-C, D[15-5] D0, D1, P always available (FPOLL only)--
--NOTE: D[A] is sacred in FPOLL!!--
Do not use an R registers, please !!!!
Do not use Status
Do not use S-R0-0

Envisioned application(s):
Extend or add a statement (like ON INTP) and need
to zero the RAM address associated with the statement.

Zero I/O Buffer associated with an application because
all other program information is being zeroed.

Note:
Do not use S-R0-0 under ANY circumstances.
(counted on by PURGE ALL)

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/04/83 | JP | Added poll |
| 04/23/83 | JP | Revised/updated documentation |
| 05/13/83 | JP | Changed Usage documentation |

## 17.17 pIMCHR - Poll for unrecognized IMAGE char

Category: POLL  File: MB&IMG::MS

Name:(S) pIMCHR - Poll for unrecognized IMAGE char

Type: FPOLL

Purpose:
To alert LEX files that an unrecognized character was found while parsing an IMAGE string. If a LEX file doesn't handle it, "Invalid IMAGE" error will result.

Should poll be "Handled" (return with XM=0)?:
Yes.

Meaning of "Handling" Poll (what does code do if handled?):
Unrecognized character was accepted, IMAGE token stream was adjusted (if necessary) to process the character at execution.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set (fast poll)
B[A] = Poll number.
HEX mode.
P=0.
R0(A)=points to current position in BldIMG token stream. If any tokens are to be appended to the stream, they should be added below this point. Pointer goes to D1, usually.
R0(9-5)=execution pointer. Next time execution of an IMAGE field starts, it will start here.
R1(A)=address of unrecognized character which caused the poll. Pointer goes to D0, usually.
R1(9-5)=length of IMAGE string (in nibbles)
R1(S)=counter for complex numeric fields
R2(A)=counter for digits in numeric field (also for A's in a literal field, but this counter is not used).
R3(A)=Program Counter (D0 at entry of USING routine)
R3(9-5)=address of start of IMAGE string.
See USING routine header for explanation of status bits.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.
D1=points to current positon in BldIMG token stream. If tokens have been added, D1 must have been moved; if not, D1 must have been set to the address in R0(A).
D(A)= AvMemSt
R1(A)=address of next parse character in IMAGE string. This pointer should be moved past the character

which caused the poll.
P is used to jump (see NOTE)
P = 0: jump to Nxtfld
1: jump to CkDlm+ (S3 must=1)
2: jump to IMGxq1
Other fields in R registers should be untouched, unless the poll handler has specific reasons to change them.
Status bits should be untouched, unless the poll handler has specific reasons to change them.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=1.
R registers untouched.
(If not handled by any LEX file, IMAGE routines issue an "Invalid IMAGE" error.)

Available subroutine levels:
5

NOTE:
IMAGE parsing and execution are very involved. Study the USING routine header and pIMbck, pIMcpi, pIMXCH and pIMXQT poll documentation to learn more about the process. The USING routine header describes the meaning and values of the IMAGE tokens.

The IMAGE string and BldIMG token stream is kept in available memory, below AvMemEnd. The BldIMG token stream is built backwards (toward address 0) from the boundary of the IMAGE string:

```
      BldIMG tokens   IMAGE string   AvMemEnd
 ---------------------------------------------
     |               |   ?          |
 ---------------------------------------------
     ^                   ^ \
     |                   | unrecognized char
  pointer in R0(A)   pointer in R1(A)
```

Just because a character was encountered that your LEX file will accept, don't accept it blindly. For instance, don't accept a digit specifier in a literal field. For cases like this, you have to back up through the BldIMG tokens to the field delimiter to see what type of field is being processed. If the syntax of the new character doesn't meet your requirements, let the poll go on with XM=1 ("not handled").

Any strange characters put into the BldIMG stream by a poll handler which require special processing should be preceded by a uIMXCH token to alert the IMAGE execution routines that the poll handler will execute it. Similarly, any strange characters which will adversely affect the backward searching during parse should be "protected" by a uJMPst token (which jumps over 14 nibbles), or a uIMbck token (which causes a pIMbck poll so that the poll handler can do the backup). Backward searching during parse is performed for two reasons:

1) to search for an open parenthesis (either to match a closing paren, or at the end of the IMAGE string to verify no unmatched parens).
2) to search for delimiter (to initialize an output field, or to fill in the number of digits in a numeric field).

See the pIMbck poll documentation for appropriate use of the uIMbck token. See the pIMXCH poll documentation for examples of "protecting" the tokens.

Upon return from the pIMCHR poll, the poll handler can select three locations to jump to:

Nxtfld -- This routine intializes a new field, and will accept only the normal start-of-field characters (such as D,X,Z,A,S,etc.) If a normal start-of-field character is not found, another pIMCHR poll will be issued. Nxtfld should be used if the unrecognized character is, say, a new type of digit specifier, a new editing symbol, or a new delimiter. However, if the new character initiates an output field, you should jump to IMGxq1.

CkDlm+ -- This routine checks for editing characters, then accepts only a standard delimiter (comma,"/","@",etc.). If a delimiter is not found, another pIMCHR poll will be issued. In order to jump to CkDlm+, S3 must be set=1 ! CkDlm+ should be used if the unrecognized character, say, terminates a field, or describes an entire non-output field (such as a new symbol which sounds the beeper).

IMGxq1 -- This routine executes all pending IMAGE fields. This should be done any time a new output field is initiated. If the unrecognized character initiates a new output field, a uRESTP (restart IMAGE parse) token should be written into the BldIMG stream, and execution begun by jumping to IMGxq1. A good example is the complex field, which intercepts the pIMCHR poll and causes a jump to IMGxq1 upon return.

What registers/RAM may be used if handled?:
A,B,C,D,D0,D1,P
R registers only to adjust values for specific reasons

What registers/RAM may be used if not handled?:
A,B,C, D[15-5], D0, D1, P
Don't change AVMEME pointer, or write to available memory below AvMemEnd.

Special memory/pointer considerations (are pointers funny?):
The IMAGE string is stored just below AvMemEnd. The BldIMG token stream is stored below that. All this resides in available memory, so it is volatile (in the sense that someone can inadvertantly write over it, if they aren't careful).

Envisioned application(s):
Well....
1) Complex IMAGE fields
2) A symbol which causes a one-time parsing of the IMAGE string (and stores it in an I/O buffer) for subsequent execution. This would be much faster than parsing it each time.
3) Allowing the "%" symbol to generate digit output.
4) Specifying the "!" symbol, say, to generate a beep during IMAGE execution.
5) Using square brackets to allow multiple-character replication. E.g., "5[3DC]" would be equivalent to "3DC3DC3DC3DC3DC"
6) ... and so on ...

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Implemented, documented. |

## 17.18 pIMbck - Backward search, IMAGE parse

Category: POLL File: MB&IMG::MS

Name:(S) pIMbck - Backward search, IMAGE parse

Type: FPOLL

Purpose:
Allow LEX files to handle unknown tokens while performing backward search during IMAGE parse.

Should poll be "Handled" (return with XM=0)?:
Yes.

Meaning of "Handling" Poll (what does code do if handled?):
Backward search over unknown tokens was performed properly. One of two actions was performed:
1) unknown field was closed
2) unknown field was verified to be closed during final parentheses match.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
R1(A)=address of symbol which caused backward search (either a right parenthesis, or the end-of-image).
R2(A)=address (in BldIMG stream) of the uIMbck token which caused the poll.

R0(A)=current position in BldIMG token stream. Next token to be entered must be written below this address.
R0(9-5)=address to start next IMAGE execution
R1(9-5)=length of IMAGE string (#nibbles)
R1(S)=counter for 2 complex numeric fields.
R3(A)=Program Counter
R3(9-5)=address of start of IMAGE string.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.

D(A)=AvMemSt
D1=current position in BldIMG (taken from R0(A), adjusted if necessary)
Other R register fields unchanged
See NOTE below for changes to BldIMG stream.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
R registers untouched.
BldIMG token stream untouched.
If not handled by any LEX file, IMAGE routines will issued an "Invalid IMAGE" error.

Available subroutine levels:
4

NOTE:
The pIMbck poll is issued only when a uIMbck token is encountered during backward search in IMAGE parse. The only way a uIMbck token could have been entered into the token stream is for a LEX file to have inserted it during a pIMCHR poll.

Backward searching during IMAGE parse is performed for two reasons:
1) To search for an open parenthesis: either to match a closing parenthesis (S5=0), or at the end-of-image to verify no unmatched parentheses (S5=1). Use S5 to distinguish the two cases.
2) To search for a field delimiter: to initiatiate an output field, or to fill in the number of digits in a numeric field.

The pIMbck poll is issued only for case number 1 !! (The uIMbck token is ignored during backward search for a delimiter.)

This poll can be used by any new IMAGE syntax which uses parentheses to enclose a field (such as complex fields), or by an application which needs to know when the end-of-image has occurred (whether to check its own tokenization, or whatever).

Once this poll is issued, the backward search terminates -- if handled, parsing continues at the point where the backward search was caused; if not handled by any LEX file, "Invalid IMAGE" is reported. Typically, a LEX file would expect to handle this poll only once -- to close the pending field (such as to close a complex field), or, failing to close it, to trap the error ("field not closed", such as unmatched parentheses) at end-of-image. When it is handled properly (i.e., when the pending field is closed), the uIMbck token should be overwritten with another token so that the pIMbck poll is not issued again. For instance, the MATH ROM, when handling the pIMCHR poll for a complex field, inserts the following tokens in the BldIMG stream:

```
        uI  uC  u?  ...(existing BldIMG tokens)
       (3) (2) (1)
    where
        uI =uIMbck token
        uC =uCPLXC token
        u? =flag to indicate whether multiplied field.
        uX =uIMXCH token (below)
```

Later, when the closing parenthesis is found to close the complex field, the backward search will poll at token (3). The MATH ROM will overwrite this token with a uIMXCH:

```
        uX  uC  u?  ...(existing BldIMG tokens)
       (3) (2) (1)
```

Since the complex field was properly closed, a pIMbck poll need not be issued again. Note that if the closing parenthesis had not been found, the pIMbck token would still be there at end-of-image. End-of-image also performs a backward search to detect unmatched parentheses; during this pIMbck poll, the MATH ROM would find S5=1, issuing an "Invalid IMAGE" error.

If an application handles the poll and wishes the backward search to continue, it should either perform its own backward search (see "BACK2(" routine), or "erase" its uIMbck token from the BldIMG stream and reposition D1 and D0 so that the backward search is performed once again by the IMAGE routines. That is, subtract 2 from R1(A) (so that it will point to the symbol which causes the backward search), and restore D1 from R0(A) (the current position in the BldIMG stream). Or if the poll handler wants to be polled more than once, it can, each time, move its uIMbck token out the search area (write it below the current BldIMG address), and reposition D1 and D0 as above to regenerate the backward search.

What registers/RAM may be used if handled?:

A,B,C,D,D0,D1,P,R0(A),R2,R4
Other fields in R registers may be adjusted as necessary.
BldIMG tokens may be adjusted as necessary.

What registers/RAM may be used if not handled?:

A,B,C,D[15-5],D0,D1,P,R4
Do not write below AvMemEnd (contains BldIMG tokens)

Special memory/pointer considerations (are pointers funny?):
The BldIMG stream resides in AvMem, below AvMemEnd.

Envisioned application(s):

1) The MATH ROM uses the pIMbck poll to close complex image fields. At that time, it checks whether 2 (and only 2) numeric fields were included, and whether the field had a multiplier. It also generates a uIMXCH token to execute the complex field, and another one to output a right parenthesis.
2) Say a LEX file implements an IMAGE symbol "=" which causes pre-parsing of the image string (storing the tokens in an I/O buffer). The syntax might be that it must be the first character in the image string (even before a "#"). A pIMCHR poll would be issued for the "="; the poll handler would insert a uIMbck token as the first token in BldIMG stream. When the poll handler intercepts the pIMbck poll with S5=1, it would know that the entire image string had been parsed, and was ready to store away.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Implemented, documented. |

17.19 pIMcpi - Initializing IMAGE field in complex

Category: POLL File: MB&IMG::MS

Name:(S) pIMcpi - Initializing IMAGE field in complex

Type: FPOLL

Purpose:

Alert MATH ROM that a field is being initialized while a complex field is pending. Alert other LEX files that an output field is about to be initialized.

Should poll be "Handled" (return with XM=0)?:
Yes.

Meaning of "Handling" Poll (what does code do if handled?):
New field was verified to be numeric; total number of numeric fields in the complex field does not yet exceed 2.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
R1(A)=address of character in image string which initialized field (an output character such as D,Z,*,A,K)
R0(A)=current position in BldIMG token stream
R2(B)=proposed initializing token (identifies type of field)
R2(XS)=0 (flag for IMAGE routines; don't change)
R1(S)=counter for 2 complex numeric fields

R0(9-5)=address to start next IMAGE execution
R1(9-5)=length of IMAGE string (nibbles)
R3(A)=Program Counter
R3(9-5)=address of start of IMAGE string.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
P=0
XM=0.
R2(B)=symbol which caused initialization (must be in upper case; fetched from address in R1(A))
B(X)=contents of R2(X) from entry to poll handler
D1=current position in BldIMG stream (from R0(A))
S4=0 ("do not execute yet")
D(A)=AvMemSt

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
R registers untouched.

Available subroutine levels:
4

**NOTE:**

This is a specialized poll for the MATH ROM to handle complex image fields. With some creative coding, the pIMcpi poll can be used by other LEX files.

The pIMcpi poll is only issued if S7=1 ("complex field being parsed") during image parse; and only when a new output field is being initialized in the BldIMG token stream.

There are two classes of poll handlers for pIMcpi.

1) MATH ROM -- used to process numeric fields in a complex field.
   In a previous pIMCHR poll (issued at the "C(" symbol), the poll handler must have:
   a) set S7=1
   b) set R1(S)=2

2) Other LEX files desiring to detect the initialization of any field.
   In a previous pIMCHR poll (issued at the point a new unrecognized symbol was found), the poll handler must have:
   a) set S7=1
   b) set R1(S)=0 (the MATH ROM will still intercept the pIMcpi poll, but if R1(S) is=0, it will exit "not handled")
   c) S7 must be set=0 before execution of the IMAGE tokens begins. (S7=1 during execution will always invoke the MATH ROM; see pIMcpu poll documentation.)

Note that the pIMcpi poll was designed as a special poll for the MATH ROM. Its use by any other ROM will conflict with complex fields. In particular, a new symbol can use this poll as long as it and complex fields are syntactically mutually exclusive.

- -- If S7 has been set=1 by another LEX file then the MATH ROM will not handle the pIMCHR poll for a subsequent "C(" symbol. In other words, setting S7=1 will cause an "Invalid IMAGE" when a complex field is found.
- -- Any application handling this poll cannot allow its new symbol within a complex field, since the MATH ROM, if it intercepts the poll first, will try to process it. The counter in R1(S) will cause a conflict.

(Notwithstanding the above rule, there is probably a way for a pIMcpi poll handler to manage the use of R1(S) to allow complex fields within its own new field. See the MATH ROM code for complete details.)

What registers/RAM may be used if handled?:
A,B,C,D,D0,D1,P,R4,S4,S7
R registers may be adjusted as necessary
Tokens in BldIMG stream adjusted as necessary

What registers/RAM may be used if not handled?:
A,B,C,D[15-5],D0,D1,P,R4
Other R registers untouched
Don't write to AvMem below AvMemEnd (stores BldIMG)

Special memory/pointer considerations (are pointers funny?):
BldIMG tokens are stored in AvMem below BldIMG.

Envisioned application(s):

1) MATH ROM uses pIMcpi poll to process complex image fields. Checks that field is numeric, verifies that no more than 2 numeric fields are within the complex field.
2) Say a LEX file implements a numeric field descriptor which encloses negative numbers in parentheses. The syntax might be, say, "-DDD.D", where a leading "-" would identify this type of descriptor. E.g.,
   DISP USING "-3D.2D"; -36.25
   displays "( 36.25)".
   It would cause a pIMCHR poll for the "-" symbol. At that time, the LEX file could set S7=1, R1(S)=0. When the numeric field is initialized, the pIMcpi poll should be handled to 1) check to make sure it is a numeric field, 2) put appropriate execution tokens in the BldIMG stream to effect the right output, and 3) set S7=0. Note that this new descriptor would not be allowed with complex fields, either imbedded inside them, or vice versa (unless some very creative code was written).

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Implemented, documented. |

## 17.20 pIMXQT - Begin IMAGE execution

Category: POLL File: MB&USG::MS

**Name:(S) pIMXQT - Begin IMAGE execution**

**Type:** FPOLL

**Purpose:**
To alert LEX files that IMAGE fields are about to begin executing.

**Should poll be "Handled" (return with XM=0)?:**
No. The IMAGE routines do not check if poll handled.

**Meaning of "Handling" Poll (what does code do if handled?):**
None.

**Entry conditions for handler (registers, ST, RAM, etc.):**
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
R0(9-5)=address of token in BldIMG stream where execution is to start.
R1(A)=address of last character to be parsed in IMAGE string.
R3(A)=Program Counter (original D0 as passed to the USING routine, updated as expressions are executed).
RAM usage as shown below, in NOTE.

**Normal exit conditions from handler if handled (ST, RAM, registers, etc.):**
HEX mode.
XM=0.
See NOTE, below.

**Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):**
HEX mode.
XM=1.
See NOTE, below.

**Available subroutine levels:**
5

NOTE:

IMAGE parsing and execution are very involved. Study the USING routine header and pIMbck, pIMcpi, pIMXCH and pIMCHR poll documentation to learn more about the process. The USING routine header describes the meaning and values of the IMAGE tokens.

During parsing, the IMAGE string and BldIMG token stream is kept in available memory, below AvMemEnd. The BldIMG token stream is built backwads (toward address 0) from the boundary of the IMAGE string. At the time of the pIMXQT poll memory looks like this:

```
                                    (Old AvMemEnd)
-----------------------------------------------
    + | BldIMG tokens | IMAGE string    |
---/-------------------------------------------
  /             ^               ^
cont'd below    |            'x'= last character parsed
    xqt address in R0(9-5) points to execution token

-----------------------------------------------
 | 5 nibs |3 nibs| 5 nibs | 5 nibs | 5 nibs | BldIMG...
-----------------------------------------------
 ^      ^      ^        ^      ^         ^
 |      |      |        |      | length of IMAGE string
AvMemEnd |  status bits |   offset to 'x' above
        |           offset to start of IMAGE string
   5 zeros (stores offset
   to xqt address when necessary)
```

IMAGE execution begins every time a new output field is parsed, or when the end of the IMAGE string is found. By the time this poll occurs, all set-up for execution has been performed (all pointers and offsets have been stored away in AvMem). R1(A) contains the address of the IMAGE character which caused execution to start (a specifier for a new field, or a uIMend token).

What the poll handler does with the pIMXQT poll is up to it. The mainframe IMAGE execution routines should serve for any type of output (DISP USING, PRINT USING, OUTPUT USING, etc.), unless some LEX file wants to output to some non-standard device. If so, it would pick up the IMAGE execution at the pIMXQT poll and perform its own execution.

The most useful implementation of a pIMXQT poll handler is for ENTER USING (found in the HPIL ROM).

The ENTER USING execution routines are vastly different from the output routines, but use the same IMAGE token streams. Therefore, the ENTER USING code intercepts the pIMXQT poll and performs its own execution.

How the poll handler returns is also up to it. In the case of ENTER USING, the poll handler jumps directly back to entry point USGrst (restart parse), without exiting through the poll code. A poll handler may exit through the poll code after "handling" the poll, but it would want to adjust pointers in R0 and possibly in RAM, also.

If exiting through USGrst:
- -- RAM pointers, offsets and ST storage unchanged
- -- R3(A)=Program Counter
- -- other R registers unimportant

If exiting through poll code (XM=0):
- -- RAM pointers, offsets and ST storage unchanged
- -- R3(A)=Program Counter
- -- R0(9-5)=xqt address, pointing to a uRESTP token

What registers/RAM may be used if handled?:
A,B,C,D,D0,D1,P,R1,R2
R0 (to adjust address of execution token)
R3 (to adjust Program Counter)

What registers/RAM may be used if not handled?:
If truly "not handled":
A,B,C,D[15-5],P,R2
If handled, but leaving XM=1:
A,B,C,D[15-5],P,R2

Special memory/pointer considerations (are pointers funny?):
None. AvMem is available for writing to; this will not disturb the IMAGE routines.

Envisioned application(s):
1) ENTER USING routines use the pIMXQT poll to override the mainframe output code, instead executing code which inputs variables using the BldIMG token stream.
2) A LEX file may "pre-parse" an IMAGE string (and store it in an I/O buffer) for faster execution, eliminating the need to parse the IMAGE string every time the statement is executed. It could invoke the IMAGE parse routines and trap the pIMXQT poll before execution starts.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 17.21 pIMXCH - Unrecognized symbol in IMAGE execution

Category: POLL File: MB&USG::MS

Name:(S) pIMXCH - Unrecognized symbol in IMAGE execution

Type: FPOLL

Purpose:
Allow LEX files to execute unrecognized IMAGE tokens.

Should poll be "Handled" (return with XM=0)?:
Yes. If the poll is not handled by any LEX file, the IMAGE routines issue an "Invalid USING" error.

Meaning of "Handling" Poll (what does code do if handled?):
The symbol was executed by a LEX file, generating the appropriate output.

Entry conditions for handler (registers, ST, RAM, etc.):

```
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
RO(A)=address of uIMXCH token which caused poll.
If within a numeric field:
   RO(9-5)= counter for #zeroes in field
   RO(S)= flag to identify last numeric symbol:
          0= *
          1= Z
          5= D
R3(A)=Program Counter
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

HEX mode.
XM=0.
R0(A)= address+2 of next token to execute
in BldIMG stream
S5=0
R3(A)=Program Counter
RAM storage above AvMemEnd untouched.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

HEX mode.
XM=1.
R0,R3 and RAM storage above AvMemEnd untouched.

Available subroutine levels:

5

NOTE:

See NOTE under pIMXQT poll for RAM storage description.

The pIMXCh poll is issued only when a uIMXCH token is encountered when executing the BldIMG tokens. The uIMXCH token can only be placed by a poll handler which previously handled a pIMCHR poll; their combined purpose is to allow "strange" characters to be parsed and executed in a IMAGE string.

The uIMXCH token in the BldIMG stream should be accompanied by other tokens (or ASCII bytes) which the poll handler will use for identification and execution.

The pIMXCH poll is handled by the MATH ROM when executing complex IMAGE fields. The uIMXCH token is inserted in the BldIMG stream in two places: 1) at start of the complex field, so that the complex expression is evaluated, and a left parenthesis is output, and 2) at the end of the field, to close out the field and display a right parenthesis. In the first case a special token accompanies the uIMXCH token to identify it to the MATH ROM as a complex field. In the second case, only an ASCII ")" accompanies the uIMXCH token, which is all that is needed to signal that the right parenthesis need be displayed.
For the two cases of complex fields using the uIMXCH token, the partial tokenization looks like this (it's built backwards towards address zero):

```
case 1)
          uX  uC  u?  ...(existing BldIMG tokens)
         (3) (2) (1)
```

```
case 2)
                 uX  =)  ...(existing BldIMG tokens)
                (5) (4)

where
        uX = uIMXCH token
        uC = uCPLXC token
        u? = flag byte indicating "multiplied field"
        =) = ASCII ")"
```

The code in the MATH ROM looks for the appropriate byte values preceding the uIMXCH token to indicate the appropriate action.

If a uIMXCH token has been inserted within a numeric field, some extra steps have to be taken to insure the float-check (for D symbols), and the skip-check (for NaNs, Infs and overflows) are performed properly.

The float-check is performed to count the number of positions that editing symbols or sign symbols must float over leading zeroes (hence only performed for the D fields). The skip-check is performed to count the number of positions to fill with spaces (for NaN or INF) or *'s (for overflow). If the new symbol needs to be counted for either reason, you must follow the uIMXCH token with a "D" or "S" or something appropriate to cause the count to be incremented. This extra "D" or "S" should be protected from the execution routine; that is, the uIMXCH poll handler should position the execution pointer (passed back in R0(A)) past this extra character. On the other hand, to make the new symbol terminate either check, insert an EndNum token as an extra character. Both checks do not poll for uIMXCH; only the token executor issues a poll. Thus if the uIMXCH token involves pointers which might look like any of the symbols

  D S X M . C Z P R  uMULT, uSTRPT or a byte>E5

you will have to protect it with uSTRPT (which skips over 14 nibbles) or a uMULT (which skips over 10 nibbles).

For instance, say the new character "I" is allowed anywhere in an output field, having the same effect as the "parent" symbols (the rest of the symbols which define the type of field), except that the character in that position is displayed in inverse video. For instance, "AAIA" is equivalent to "AAAA", except that the third character is displayed in inverse video. Similarly, "DDID" is equivalent to

"DDDD", with an inverse video digit in the third position. Since "I" should be counted in the float-check and skip-check (since it is allowed in a numeric field), the (partial) token stream should look like this (it's built backwards towards address 0), using using "DDID" as an example:

```
    =D  =I  =D  uX  =D  =D
   (6) (5) (4) (3) (2) (1)
```

where
    =D = ASCII "D"
    =I = ASCII "I"
    uX =uIMXCH token to cause pIMXCH poll.
Token (3) would be inserted by the poll handler for a pIMCHR poll. Then, during execution, the float-check routine will count (4), and the pIMXCH poll handler will execute (5) when the poll is issued at (3). When returning from the pIMXCH poll, the execution pointer in R0(A) should be at (6).

Now say that the symbol "!<f,d>" causes a beep of frequency f, duration d; the new symbol can be inserted in any output field. Then "DD!<800,.5>D" would be tokenized as follows:

```
    =D   uJ  p2  p1  uS  =!   uX  =D  =D
   (9)  (8) (7) (6) (5) (4)  (3) (2) (1)
```

where
    uJ =uJMPst (jumps 14 nibs on backward search)
    p2 =5 nibble pointer to beep duration
    p1 =5 nibble pointer to beep frequency
    uS =uSTRPT (jumps 14 nibs in float-check)
    uX =uIMXCH token, to cause pIMXCH poll
    =D =ASCII "D"
    =! =ASCII "!"
Then, during a float-check, (5) will cause a jump over the pointers p1 and p2, to token (9); otherwise these pointers might be interpreted as executing tokens. Token (8) is included for backward searching during parse; it causes a jump over pointers p1 and p2 for the same reason. Token (4) will be executed by the poll handler when the pIMXCH poll is issued at (3).

What registers/RAM may be used if handled?:
    A,B,C,D,D0,D1,P
    R0 (to adjust pointer or counter)
    R1 (to adjust counter)
    R3 (to adjust Program Counter)

What registers/RAM may be used if not handled?:
A,B,C,D[15-5], D0, D1, P
R0, R3 untouched.
RAM storage above AvMemEnd untouched.
Expression stored in AvMem below AvMemEnd untouched.

Special memory/pointer considerations (are pointers funny?):
If the pIMXCH poll is issued while an output field is pending (that is, the expression has already been executed, but output not completed), the memory below AvMemEnd contains the expression, and may not be altered.

Envisioned application(s):
Complex IMAGE fields.
Some more are listed in NOTE, above.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 12/08/82 | MB | Documentation |

## 17.22 pIMcpw - Working on complex image field

Category: POLL File: MB&USG::MS

Name:(S) pIMcpw - Working on complex image field

Type: FPOLL

Purpose:
Alert MATH ROM to work on complex field.

Should poll be "Handled" (return with XM=0)?:
No.

Meaning of "Handling" Poll (what does code do if handled?):
Complex expression was evaluated, real or imaginary part has been put on stack, ready for formatted output.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
R0(A)=address of numeric delimiter (in BldIMG token stream) which caused the poll.
R3(A)=Program Counter

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
This poll can only be handled by the MATH ROM. It cannot exit through the poll routines with XM=0; it can only exit by jumping to USnm05.
HEX mode.
A(W)=numeric expression (either the real or imaginary part, as appropriate)
D1 points to AvMemEnd-16.
R registers untouched.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
R registers untouched.

Available subroutine levels:
7 (junk the two poll levels, and jump to USnm05)

NOTE:
This poll can only be handled by the MATH ROM, as part of complex image field execution.

What registers/RAM may be used if handled?:
A,B,C,D,D0,D1,P,R0(15-5),R1,R2,R3(9-5),R4
R0(A) should not be used
R3(A) should not be used

What registers/RAM may be used if not handled?:
A,B,C,D[15-5],D0,D1,P,R1,R2,R4

Special memory/pointer considerations (are pointers funny?):
At the time of the poll, AvMem is not used to store anything. If the poll is handled properly, the expression for output resides at AvMemEnd-16.

Envisioned application(s):
MATH ROM complex field output. Only.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 01/01/83 | MB | Implemented, documented. |

## 17.23 pWCRD8 - Poll To Write Copycode 8 File To Card

Category: POLL    File: MN&CD::MS

Name:(S) pWCRD8 - Poll To Write Copycode 8 File To Card

Type: POLL

Purpose:
Allow handler to copy a file with copycode of 8 out to card.

Should poll be "Handled" (return with XM=0)?:
Yes, if you do the copy.

Meaning of "Handling" Poll (what does code do if handled?):
The copy has been performed. The WHOLE thing... prompting, writing, verifying, etc. The copy code will perform a normal exit. If poll is not handled, copy code performs an eror exit.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.
Card header buffer (ID=bCARD) has been allocated and set up (as per FILCRD header) with:
  Name
  Filetype
  Creation date
  Subformat and track#.
R1[A] points at start of file header.
R2[A] points at card header I/Obuffer (past header).
A[3-0] contains filetype.

Normal exit conditions from handler if handled (ST, RAM,

registers, etc.):
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
5

What registers/RAM may be used if handled?:
A-D, D0, D1, P, R0-R4, all scratch RAM.

What registers/RAM may be used if not handled?:
A-D, D0, D1, P, R0-R4, all scratch RAM.

Envisioned application(s):
Somebody's got to know how to copy out a file with a crazy copycode like 8.

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/01/83 | NM | Added documentation |

## 17.24 pWCRD - Card Write Poll

Category: POLL File: MN&CD::MS

Name:(S) pWCRD - Card Write Poll

Type: FPOLL

Purpose:
Allow processing before writing out a card track.

Should poll be "Handled" (return with XM=0)?:
If polling should terminate, then poll should be

handled.

Meaning of "Handling" Poll (what does code do if handled?):
Code does nothing different if poll is handled.
Handling merely terminates polling, which is probably the desired result.

Entry conditions for handler (registers, ST, RAM, etc.):
We are about to prompt for a card.
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.
R1-R2 set up as FILCRD documentation explains.
The bCARD buffer contains the card header.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.
Card header modified as desired.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
Card header modified if desired.

Available subroutine levels:
2

NOTE:
If you modify the card header, you must recompute the card header checksum, or you will never be able to read back the card you have written.

What registers/RAM may be used if handled?:
A-D, D0, D1, P, R0, R3, R4, all scratch RAM.

What registers/RAM may be used if not handled?:
A-C, D[5-15], D0, D1, P, R0, R3, R4, all scratch RAM.

Envisioned application(s):
Setting up card header for partial card recovery.
It is highly doubtful whether partial card recovery can be done, but this is the hook which allows you to try it. The documentation for FNDPRT explains the meaning of the partial card recovery information fields. Good luck.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 08/01/83 | NM | Added documentation |

17.25 pRCRD - Poll After Reading Card.

Category: POLL File: MN&CD::MS

Name:(S) pRCRD - Poll After Reading Card.

Type: FPOLL

Purpose:
Poll after each card track is read.

Should poll be "Handled" (return with XM=0)?:
If it is desired to terminate polling, yes.

Meaning of "Handling" Poll (what does code do if handled?):
Code doesn't do anything different if poll is handled.
Handling simply stops polling, which may be desirable.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.
R1, R2 as defined in CRDFIL header.
bCARD buffer contains header of card just read in.
Code has just read a track and is about put up a
"trk <nnn> done" message.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
3

What registers/RAM may be used if handled?:
A-D, D0, D1, P, R0, R3, R4.
All scratch RAM.

What registers/RAM may be used if not handled?:
A-C, C[5-15], D0, D1, P, R0, R3, R4.
All scratch RAM.

Special memory/pointer considerations (are pointers funny?):
There is no available memory.

Envisioned application(s):
This is supposed to be the hook to allow partial card recovery. I am skeptical, but I'll keep it to myself. If the card was written by somebody who knows how to do partial card recovery, the header will contain data necessary to perform recovery. This poll is an opportunity to take the data and stuff it somewhere useful. One recovery scheme which worked very well is the past was storing the data in the space to be occupied by adjacent tracks IF the adjacent track has not been read yet. The flaw in this is what happens if that data is munched by an unsuccessful read in the adjacent track. The data is lost. So what to do? Maybe create an I/O buffer to hold the data. Of course that buffer had better be around before the read is initiated, since the read code sucks up all available memory to make room for the biggest card set possible. Good luck.

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/01/83 | NM | Added documentation |

## 17.26 pCRDAB - ABORT Card Read Poll

Category: POLL File: MN&CD::MS

Name:(S) pCRDAB - ABORT Card Read Poll

Type: FPOLL

Purpose:
Poll upon ATTN-key or timeout abort of card read operation.

Should poll be "Handled" (return with XM=0)?:
Yes, if...

Meaning of "Handling" Poll (what does code do if handled?):
... handler has cleanly terminated card read operation. This means collapsing the file to the proper size (which may be zero). If poll is handled, card reader code does not collapse file.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.
R1 and R2 have meaning as explained in CRDFIL header.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
3

What registers/RAM may be used if handled?:
A-D, D0, D1, P, R0-R4, all scratch RAM.

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P, R0, R3, R4, all scratch RAM.

Special memory/pointer considerations (are pointers funny?):
There is no available memory.

Envisioned application(s):
This is a chance to do partial card recovery with all that neat information saved during the pRCRD poll. See that documentation for appropriate caveats.

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/01/83 | NM | Added documentation |

## 17.27 pCONFG - Configuration Poll

Category: POLL File: MN&CNF::MS

Name:(S) pCONFG - Configuration Poll

Type: FPOLL

Purpose:
Poll at termination of configuration to allow:
1) Claiming of I/O buffers.
2) Changing configuration of machine.

Should poll be "Handled" (return with XM=0)?:
Yes, but ONLY IF reconfiguration is desired.

Meaning of "Handling" Poll (what does code do if handled?):
Calling code jumps to beginning of configuration code and reconfigures the system.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
--FPOLL handler is two levels deeper than caller--
LEXBUF (from where pCONF is invoked) saves 3 stack levels. A responder may use UP TO 3 levels

NOTE:
"Handling" the poll (returning with XM=0) is very serious business. It means that you want the machine reconfigured. Lazy writers of poll handlers who fail to RINSXM when they should can hang the machine in CONFIGURATION forever.

--SCRATCH RAM TO CONSIDER BELOW:--
--STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
--LEXPTR.--

What registers/RAM may be used if handled?:
All CPU registers.
All scratch RAM (I think).

What registers/RAM may be used if not handled?:
All CPU registers except D[A].
All scratch RAM (I think).

Special memory/pointer considerations (are pointers funny?):
May be in CALC mode.

Envisioned application(s):
Three main ones: 1) claiming I/O buffers, 2) creating I/O buffers, and 3) changing configuration.

1) Claiming: This is the time to reclaim I/O buffers to keep them from being deleted. Just before this poll, all I/O buffers are marked for deletion. To keep your I/O buffers from being deleted, you need to perform an I/ORES on those you want to keep. [Marking/unmarking for deletion consists of clearing/setting (respectively) the upper bit of the buffer ID number. Until the buffer is restored (unmarked), it will not be found with I/OFND because it will have a different number.]

2) Creating: This may be the time to create needed I/O buffers. Or you may have done it at wakeup time. Or maybe some other time. But maybe here.

3) Changing: There are certain ways software can change the configuration of the machine; specifically by doing FREE or CLAIM port. Sample situation: a plug-in may contain a ROM with a RAM intended only for the ROM's use. When polled at configuration

time, the ROM examines the RAM table and determines that the RAM living in the same plug-in is configured as system RAM. The ROM then performs all the trappings of FREEPORT except the configuration. It then indicates that the poll has been handled, and the code reconfigures the system. When this poll happens again (as it inevitably will), the ROM will see that its companion RAM is configured as IRAM, and will not repeat this monkey business.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/11/83 | NM | Added documentation |
| 07/05/83 | JP | Added stack level usage |

## 17.28 pWTKY - Poll When Waiting For Key

Category: POLL    File: MN&ED::MS

Name:(S) pWTKY - Poll When Waiting For Key

Type: FPOLL

Purpose:
Allow LEXFILE to circumvent waiting for and fetching ket# in KEYRD.

Should poll be "Handled" (return with XM=0)?:
Yes, if LEXFILE wishes to "press" a key.

Meaning of "Handling" Poll (what does code do if handled?):
Lexfile is "press"ing a key. If poll is handled, KEYRD goes on to process key returned by this poll.

If poll is not handled, KEYRD will look for repeating keys. Seeing none, KEYRD will pop the next key# from the keybuffer or, if buffer is empty, wait until a key is hit and then process it.

Entry conditions for handler (registers, ST, RAM, etc.):
- Carry set.
- B[A] = Poll number.
- HEX mode.
- P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
- HEX mode.
- XM=0.
- R0[B] contains key# (physical keycode).

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
- HEX mode.
- XM=1.

Available subroutine levels:
- 2

NOTE:

We are just entering KEYRD when this poll occurs.

This is the time to press a key. The time to provide a definition for a pressed key is the pKYDF poll.

What registers/RAM may be used if handled?:
- A-D, D0, D1, P, R0, R3.
- SCRTCH RAM.

What registers/RAM may be used if not handled?:
- A-C, D[15-5] D0, D1, P, R0, R3.
- SCRTCH RAM.

Special memory/pointer considerations (are pointers funny?):
- May be in CALC mode.

Envisioned application(s):
- External keyboard controller or remote keyboard.
- The poll handler may take over waiting for a key to go down if appropriate.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/19/83 | NM | Added documentation |

## 17.29 pKYDF - Poll To Define Key

Category: POLL      File: MN&ED::MS

Name:(S) pKYDF - Poll To Define Key

Type: FPOLL

Purpose:
Allow LEXFILE to define action/definition of a key.

Should poll be "Handled" (return with XM=0)?:
Yes, if you want to define or act on the key.

Meaning of "Handling" Poll (what does code do if handled?):
LEXFILE is either defining or otherwise acting on key.
Defining (returning with S0=1) means that the LEXFILE is returning a definition to whomever called KEYRD (CHEDIT, CALC mode editor, or whoever).
Acting on (returning with S0=0) means that the LEXFILE is using the key in some way (such as toggling a flag or ignoring) and KEYRD should not return a definition to the caller, but should instead get the next key to process.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.
R0[A]=keycode (from keycode map),
R0[9-5]=key# (physical keycode).

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=0.
If handled but not returning a definition ("acting on" a key): S0=0.
If returning a definition:
S0=1.
Definition pointer in DEFADR (in RAM) as follows:
DEFADR: length of string in bytes (2 nibs).
DEFADR+2: Key type (1 nib).
0 = Single ASCII character. Includes

```
                    control chars 0-31, which may cause
                    some action by caller.
                1 = ASCII control char + #40.  This is
                    a character in the range 0-31 which
                    is to be interpreted strictly as a
                    character, not as special action keys
                    (cursor-right, etc.).  To return
                    char #01, DEFADR should point at #41
                    byte, etc.
                2 = User defined key--terminating.
                4 = User defined key--non-terminating.
                6 = User defined key--immed execute.
              8-F = LEX table entry, with lower 3 bits
                      as follows:
                        0: Parentheses needed.
                        1: Trailing space needed.
                        2: Leading space needed.
      DEFADR+3: Address of text.
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

HEX mode.
XM=1.

Available subroutine levels:

2

What registers/RAM may be used if handled?:

A-D, D0, D1, P, R0, R3.
SCRTCH RAM.

What registers/RAM may be used if not handled?:

A-C, D[15-5] D0, D1, P, R3.
SCRTCH RAM.

Special memory/pointer considerations (are pointers funny?):

May be in CALC mode.

Envisioned application(s):

Redefine keyboard.

One interesting application: Stuff funny key# in keybuffer (perhaps at pSREQ) and define it here.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/19/83 | NM | Added documentation |

## 17.30 pCLDST - Coldstart poll

Category: POLL File: SB&DVR::MS

Name:(S) pCLDST - Coldstart poll

Type: FPOLL

Purpose:
Allows module to gain control at Coldstart

Should poll be "Handled" (return with XM=0)?: No

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels: 5

What registers/RAM may be used if not handled?:
Nothing matters except D(A)

Envisioned application(s):
Operating system take overs.
Initialization of buffers, RAM, etc.

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | B.S. | Added documentation |
| 10/17/83 | B.S. | Updated documentation |

## 17.31 pMNLP - Poll on entry to main loop

Category: POLL File: SB&DVR::MS

Name:(S) pMNLP - Poll on entry to main loop

Type: FPOLL

Purpose:
Poll on entry to main loop.

Should poll be "Handled" (return with XM=0)?:
NO!! NEVER!! Take over, yes. Handle, no.

Meaning of "Handling" Poll (what does code do if handled?):
N/A

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number = pMNLP.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
N/A

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
5

NOTE:
Machine is entering an idle state. This is a good time to take over. This poll is one of the very first things done on entry to main loop. We have not done display scrolling, auto lineN, collapsing stmt buffer, checking for CALC mode, etc.

What registers/RAM may be used if handled?:
N/A

What registers/RAM may be used if not handled?:

All CPU registers except D[A].
All scratch RAM.

Special memory/pointer considerations (are pointers funny?):
May be in CALC mode. The routine fCALC? will RTNSC if we are in CALC mode without using D[A].

Envisioned application(s):
Taking over, maybe?

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/23/83 | NM | Added documentation |

## 17.32 pPWROF - Poll when powering off

Category: POLL  File: SB&DVR::MS

Name:(S) pPWROF - Poll when powering off

Type: FPOLL

Purpose:
Poll on entry to deep sleep.

Should poll be "Handled" (return with XM=0)?:
No.

Meaning of "Handling" Poll (what does code do if handled?):
N/A

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number = pPWROF
HEX mode.
P=0.
f1PWDN set iff deepsleep was called from PWROFF.

Normal exit conditions from handler if handled (ST, RAM,

registers, etc.):
N/A

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels: 3

NOTE:
The flag flPWDN indicates that the machine was called from PWROFF, as opposed to CALC mode, programmatic BYE, or somebody else.

This is a good time to take over.

What registers/RAM may be used if handled?:
N/A

What registers/RAM may be used if not handled?:
All CPU registers except D[A].
All scratch RAM.

Special memory/pointer considerations (are pointers funny?):
May be in CALC mode.

Envisioned application(s):
Some sort of takeover on shutdown.

Pocket secretary processing alarms at shutdown.
Suggested method if an alarm is due and you want to process it at power-off:
Schedule immediate wakeup through external alarm.
Create external command buffer at wakeup poll using the pocket secretary's handy ACKNOWLEDGE keyword.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 03/24/83 | NM | Added documentation |

## 17.33 pDSWNK - Poll to awake machine w/o key

Category: POLL File: SB&DVR::MS

Name:(S) pDSWNK - Poll to awake machine w/o key

Type: FPOLL

Purpose:
Poll if machine awoke without ATTN being hit or
ON TIMER going off.

Should poll be "Handled" (return with XM=0)?:
No. I don't think so.

Meaning of "Handling" Poll (what does code do if handled?):
N/A

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
N/A

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
If flTNOF is cleared during this poll, the machine will wake up AND will circumvent password processing (asking for password if one exists). If you wish to wake up the machine this way but not give control to the user, setting flMKOF will force machine back to sleep as soon as it hits the main loop. This is a way to wake up to process alarms and then return to sleep.
If ATNFLG is set during this poll, the machine will continue as though ATTN had been hit... wake up, perform password processing, etc.

Available subroutine levels: 3

NOTE:

The flag flPWDN indicates that the machine was called from PWROFF, as opposed to CALC mode, programmatic BYE, or somebody else. The importance of this is that on return from DSLEEP, PWROFF will recognize and process an external command buffer. Nobody else will. So if you wish to create a command buffer to be executed, flPWDN indicates whether or not it will be ignored.

The external command buffer was deallocated before the wakeup polls. If it currently exists, it means that a poll handler has created it. Think real hard about how badly you want to wipe out somebody else's command. On the other hand, some externally implemented sort of STARTUP may grab this buffer every time. Such are the dangers in this zoo. I guess this means not to assume that creating this buffer guarantees that it will be used.

What registers/RAM may be used if handled?:

N/A

What registers/RAM may be used if not handled?:

All registers except D[A].
All scratch RAM.

Special memory/pointer considerations (are pointers funny?):

May be in CALC mode.

Envisioned application(s):

Allowing non-ATTN, non-ON-TIMER to awake machine.

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/24/83 | NM | Added documentation |

## 17.34 pDSWKY - Poll if machine wants to wake up

Category: POLL File: SB&DVR::MS

Name:(S) pDSWKY - Poll if machine wants to wake up

Type: FPOLL

Purpose:

Poll if we are going to wake up because:
- ATTN key was hit.
- ON TIMER went off.
- Responder to pDSWNK told us to wake up.

Should poll be "Handled" (return with XM=0)?:

No.

Meaning of "Handling" Poll (what does code do if handled?):

N/A

Entry conditions for handler (registers, ST, RAM, etc.):

Carry set.
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

N/A

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

HEX mode.
XM=1.
If f1TNOF cleared, we will wake up without password processing.

Available subroutine levels: 3

NOTE:

At this point, we are committed to trying to wake up machine. If, however, f1TNOF is set on termination of this poll (it may or may not be set before poll), we will go through password processing... soliciting a password from the user if the machine has been locked.

The f1ALRM flag (ALARM annunciator) was cleared just before the poll. This is the time to set the flag if that annunciator should be on.

The flag f1PWDN indicates that the machine was called from PWROFF, as opposed to CALC mode, programmatic BYE, or somebody else. The importance of this is that on return from DSLEEP, PWROFF will recognize and process an external command buffer. Nobody else will. So if you wish to create a command buffer to be executed, flPWDN indicates whether or not it will be ignored.

The external command buffer was deallocated before the wakeup polls. If it currently exists, it means that a poll handler has created it. Think real hard about how badly you want to wipe out somebody else's command. On the other hand, some externally implemented sort of STARTUP may grab this buffer every time. Such are the dangers in this zoo. I guess this means not to assume that creating this buffer guarantees that it will be used.

What registers/RAM may be used if handled?:
N/A

What registers/RAM may be used if not handled?:
All CPU registers except D[A].
All scratch RAM.

Special memory/pointer considerations (are pointers funny?):
May be in CALC mode.

Envisioned application(s):
Takeover ROM at powerup.

Alarm processing.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/25/83 | MM | Updated documentation |

## 17.35 pSREQ - Service Request poll

Category: POLL File: SB&DVR::MS

Name:(S) pSREQ - Service Request poll

Type: FPOLL

Purpose:
Allow LEXFILE processing when a hardware service request is exerted.

Should poll be "Handled" (return with XM=0)?:
NO!! NEVER!!

Meaning of "Handling" Poll (what does code do if handled?):
N/A

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set.
B[A] = pSREQ.
HEX mode.
P=0.
f1DORM flag is set if machine is in main loop (dormant).

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
N/A

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels:
2

NOTE:
D[A], and R0-R4 must be preserved.

A copy of the user's status bits as they existed on entry to CKSREQ exists at DSPSTA (the 3 nibbles used by display routines to save status bits). Do not destroy this copy; it is needed so ST can be restored after the poll.

The available scratch RAM is, conveniently, just enough to use the clock system safely. You can save R0 and R1 at SCRTCH, D[A] at SCREX0, and subroutine levels in SCREX1, SCREX2, SCREX3.

This poll IS NOT a time to take over the machine. It may occur during display delay, program execution, character editing, wait, etc. This poll IS a time for handling service requests non-disruptively (such as scheduling an alarm, doing a beep, setting the exception flag, or anything else which does not disrupt the flow of whatever was going on when you generated your service request) and for setting up to take over the machine (such as setting a flag which tells you to grab the exception poll or the deepsleep poll).

What registers/RAM may be used if handled?:
N/A

What registers/RAM may be used if not handled?:
A-C, D[15-5], D0, D1, P, ST.
First 32 nibbles at SCRTCH.
SCREX0, SCREX1, SCREX2, SCREX3.

Special memory/pointer considerations (are pointers funny?):
We could be in CALC mode.

Envisioned application(s):
Scheduling external alarms though the clock system is one very important application. If a few simple rules are followed when dealing with the clock system, everything should work just fine:

Rule #1: If the current external alarm is past due (before current time), you may schedule an external alarm.
Rule #2: If the current external alarm is not past due, you may only schedule an external alarm if a) your alarm is not past due, and b) it occurs before the currently scheduled external alarm.
Rule #3: You can tell if one of your alarms is pending by comparing it to the current time. Do not count on the current value in the external alarm slot being yours... somebody may have followed rule #2 and jumped in ahead of you.

Another application: Remote Keyboard. Presumably your code is associated with some hardware (an HPIL mailbox, maybe) which has exerted a service request because of a remote keyboard. Take this poll as an opportunity to stuff a key# in the keybuffer. If it is not a key# which can be understood by the machine, you can define it by handling the key definition poll.

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/23/83 | NM | Added documentation |

## 17.36 pVER$ - VER$ Statement Extension Poll

Category: POLL File: SB&FCN::MS

Name:(S) pVER$ - VER$ Statement Extension Poll

Type: FPOLL

Purpose:
Allows a lex file to show its presence and revision code.

Should poll be "Handled" (return with XM=0)?:
No!!!

Meaning of "Handling" Poll (what does code do if handled?):
Not applicable

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
R2=(AVMEMS)
R3=Stack pointer
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Not applicable

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.
R2=(AVMEMS)
R3=New Stack pointer

Error exit conditions from handler (POLL only):
Not applicable

Available subroutine levels:
2

What registers/RAM may be used if handled?:
Not applicable.

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P
NOTE: D[A] is sacred in FPOLL!!
R1 and R4. Function scratch is available in the unlikely event it it needed.

What registers/RAM may be used if error exit (POLL only)?:
Not applicable

Special memory/pointer considerations (are pointers funny?):
This occurs during expression execute so keep in mind the rules of that game.

Envisioned application(s):
The poll handler is expected to add onto the string being built on the stack. The stack pointer is kept in R3 and must be decremented to point to the new end of the string. Available memory should be checked by comparing against the AVMEMS (which resides in R2).

The string added should have a leading blank followed by a short (~3-5 characters) name describing the lex file and optionally followed by a colon and a revision code. The revision code will usually be just a digit but a more complicated code may be required for a multi-chip ROM.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/08/83 | B.S. | Added documentation. |

## 17.37 pPRTIS - PRINTER IS handler poll

Category: POLL File: SB&IO::MS

Name:(S) pPRTIS - PRINTER IS handler poll

Type: POLL

Purpose:
Set up for the PRINT statement and return the address of a handler for the print items.

Should poll be "Handled" (return with XM=0)?:
YES

Meaning of "Handling" Poll (what does code do if handled?):
A handler for the PRINT statement has been provided and its address returned.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
A(A) is the address of the PRINT handler
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Not applicable

Available subroutine levels:
4

NOTE:
This poll is issued in the CKINFO routine which is in the process of setting up statement scratch to handle a PRINT/PLIST statements output.

What registers/RAM may be used if handled?:
Must not alter D1 or any status bits or any R registers
Function scratch is available

What registers/RAM may be used if not handled?:
A-D, D0, D1, P

What registers/RAM may be used if error exit (POLL only)?:

Not applicable

Special memory/pointer considerations (are pointers funny?):
Normal memory configuration

Envisioned application(s):
Extend PRINT/PLIST commands to handle unknown destination devices (specifically HPIL devices)

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/82 | N.Z. | Added documentation |
| 10/17/83 | B.S. | Updated documentation |

## 17.38 pPRTCL - PRINT class statement handler poll

Category: POLL File: SB&IO::MS

Name:(S) pPRTCL - PRINT class statement handler poll

Type: POLL

Purpose:
Set up a handler for a statement type not recognized by the mainframe.

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
The statement type has been recognized and statement scratch has been set up in accordance with CKINFO specifications for the specified type of statement.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
First nib of STMTRO is statement type

P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
STMTR0,STMTR1 set according to CKINFO specifications

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Not applicable

Available subroutine levels:
4

NOTE:
Function scratch is available
SCRTCH, SNAPBF, TRFMBF, LDCSPC

What registers/RAM may be used if handled?:
Statement scratch should be set by poll handler
A-D, D0, P

What registers/RAM may be used if not handled?:
A-D, D0, D1, P

What registers/RAM may be used if error exit?:
Not applicable

Special memory/pointer considerations (are pointers funny?):
No special conciderations

Envisioned application(s):
Allows adding new keywords in the same class as DISP and PRINT.

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/82 | N.Z. | Added documentation |
| 10/18/83 | B.S. | Updated documentation |

## 17.39 pRDNBF - Write Current Sector, Read Next Sector

Category: POLL File: SC&DAT::MS

Name:(S) pRDNBF - Write Current Sector, Read Next Sector

Type: FPOLL

Purpose:

Using the FIB, write current file I/O buffer to where it came from in a mass memory device, and read in next sector to the file I/O buffer.

There are total of 3 polls can be used to read/write a sector between a mass memory device and I/O buffer:

1. pRDNBF - Writes buffer out to current sector and read in next sector. If buffer content has not been altered, just read in next sector.
2. pRDCBF - Reads in current sector from mass memory device to the I/O buffer. This poll does not care about the content currently in the I/O buffer.
3. pWRCBF - Writes I/O buffer to current sector in the mass memory device.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):

As specified above.

Entry conditions for handler (registers, ST, RAM, etc.):

B[A] = Poll number.
HEX mode.
P=0.
STMTD1 contains the FIB entry address of the file

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

Carry clear.
HEX mode.
XM=0.
Current position in FIB is set to start of next sector.
File access nib in FIB is set to zero.

This poll handler always call the routine SNAPRS to restore A,D,D0,D1 from the snap save RAM before return. So if the polling routine calls the SNAPSV before issuing this poll, it can consider A,D,D0,D1 will not be change by this poll handler.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Error exit conditions from handler:
Won't return to calling routine if error occur, direct exit to BSERR routine.

Available subroutine levels: 3

What registers/RAM may be used if handled?:
A-D, D0, D1, P, ST[0-4]
(B,C,P,ST[0-4] if SNAPSV been called)

What registers/RAM may be used if not handled?:
C, D0

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.40 pREADN - READN on File of Copycode = 8

Category: POLL File: SC&DAT::MS

Name:(S) pREADN - READN on File of Copycode = 8

Type: POLL

Purpose:
Execution of READ # statement when the copy code of the

file is 8.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Complete the execution of READ # statement.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.

D[S] = Copy code of the file.
D(A) = # of bytes to end of file
R0(A)= Current position (absolute address)
R0(15:14) = Relative position in buffer if external
R1 = Record length in bytes
CHN#SV = Channel # specified in the statemnt.
STMTD1 = FIB entry address of the file.
(All the file related information can be found in the FIB entry of the file)
STMTD0 = Program counter points at the semmicolon of the statement.
S9 = 0 if serial access (record # not specified)
= 1 If random access
S10 = 0 if file is in mainframe RAM/ROM
= 1 if file is in external mass memory deivce
S11 = 0 if file is not in Independent RAM
= 1 if file is in Independent RAM

At the time when this poll is issued, the READ# already process the channel number and the record number -if specified.
If the record number has been specified, the pSREC# poll should been issued earlier so the file pointer (in the FIB) should already pointing at the start of the record.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
If the poll is handled, the handler should handle the statement completely. So the handler should directly exit to NXTSTM. The handler doesn't need to worry about the math stack used by the POLL routine, it will be taken cared by the run loop.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=1.

Error exit conditions from handler:

Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
All CPU registers, scratch RAM, S11-0

What registers/RAM may be used if not handled?:
A, C D0, D1

What registers/RAM may be used if error exit ?:
All CPU registers, scratch RAM, S11-0

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.41 pEOFIL - Poll at End-of-File

Category: POLL File: SC&DAT::MS

Name:(S) pEOFIL - Poll at End-of-File

Type: FPOLL

Purpose:
When end of file has been reached in a READ # statement, poll to give a LEX file a chance to act before the READ# statement would otherwise exit to error.
One possible thing an LEX file can do is to implement the "ON EOF GOTO/GOSUB <label>" mechanism.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
The end-of-file error has been intercepted.

Entry conditions for handler (registers, ST, RAM, etc.):
    Carry set.
    B[A] = Poll number.
    HEX mode.
    P=0.
    STMTD1 contains the FIB entry address of the file.
    The file pointer in FIB is pointing at :
    TEXT file : End-of-file mark (FFFF).
    SDATA file: Past the last data item of the file.
    DATA file : Pointing at an end-of-file mark or past the end of the file.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
    If handle, the handler should never return to the polling routine. If it ever returns to the polling routine, an "End of File" will be generated.
    This poll is just provide a hook for an LEX file to intercept the end-of-file error. The possible thing an LEX file can do to answer this poll is to implement a "ON EOF GOTO/GOSUB <label>" type of trap.

Normal exit conditions from handler if not handled:
    HEX mode.
    P = 0

Error exit conditions from handler (POLL only):
    HEX mode.
    P = 0

Available subroutine levels: 6

What registers/RAM may be used if handled?:
    All CPU registers, scratch RAM, ST11-0.

What registers/RAM may be used if not handled?:
    All CPU registers, scratch RAM, ST11-0.

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.42 pPRINN - PRINT# on File of Copycode = 8

Category: POLL File: SC&DAT::MS

Name:(S) pPRINN - PRINT# on File of Copycode = 8

Type: POLL

Purpose:
Execution of PRINT # statement when the copy code of the file is 8.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Complete the execution of PRINT # statement.

Entry conditions for handler (registers, ST, RAM, etc.):

```
B[A] = Poll number.
HEX mode.
P=0.

D[S] = Copy code of the file.
D(A) = # of bytes to end of file
R0(A)= Current position (absolute address)
R0(15:14) = Relative position in buffer if external
R1   = Record length in bytes
CHN#SV = Channel # specified in the statemnt.
STMTD1 = FIB entry address of the file.
 (All the file related information can be found in the
  FIB entry of the file)
STMTD0 = Program counter points at the semmicolon of the
         statement.
S9   = 0 if serial access (record # not specified)
     = 1 If randon access
S10 = 0 if file is in mainframe RAM/ROM
    = 1 if file is in external mass memory deivce
S11 = 0 if file is not in Independent RAM
    = 1 if file is in Independent RAM
```

At the time when this poll is issued, the READ# already process the channel number and the record number -if specified.
If the record number has been specified, the pSREC# poll should been issued earlier so the file pointer(in the

FIB) should already pointing at the start of the record.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
If handled, the handler should complete the PRINT# statement and directly exit to NXTSTM. The math stack will be cleared by the run loop automatically.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=1.

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
All CPU registers, scratch RAM, ST11-0

What registers/RAM may be used if not handled?:
B,C, D0, D1

What registers/RAM may be used if error exit ?:
All CPU registers, scratch RAM, ST11-0

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.43 pFTYPE - Search for file type table entry

Category: POLL File: SC&FIL::MS

Name:(S) pFTYPE - Search for file type table entry

Type: POLL

Purpose:
  Search file type table in LEX file for a given file type number.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
  Returns with D1 pointing to the file type table entry that contains the file type.

Entry conditions for handler (registers, ST, RAM, etc.):
  B[A] = Poll number.
  HEX mode.
  P=0.
  A[A] = file type

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
  Carry clear
  HEX mode.
  XM=0.
  D1 pts to start of the file type entry in the table
  A(S) = Position of file type number within entry
         (1 = first file type, etc.)
  A[A] = As entry condition

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
  Carry clear
  HEX mode.
  XM=1.

Error exit conditions from handler:
  Carry set.
  Hex mode.
  Error can only happen when there is not enough memory to do the poll at all.

Available subroutine levels: 4

What registers/RAM may be used if handled?:
  A-C, D1, P

What registers/RAM may be used if not handled?:
  A-C, D1, P

What registers/RAM may be used if error exit (POLL only)?:

A-C, D1, P

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.44 pFASCH - Search for File Type by Name

Category: POLL    File: SC&FIL::MS

Name:(S) pFASCH - Search for File Type by Name

Type: POLL

Purpose:
Search file type table in LEX file for a given file type name.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Returns the file type number for the unprotected form of the file type.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
A[9-0]= File type in ASCII, right justified with leading blanks(first character in A(B)).

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
A[3-0] = File type number

Normal exit conditions from handler if not handled (ST, RAM,

registers, etc.):
    Carry clear
    HEX mode.
    XM=1.

Error exit conditions from handler (POLL only):
    Carry set.
    HEX mode.
    Error can only happen when there is not enough memory to do the poll at all.

Available subroutine levels: 4

What registers/RAM may be used if handled?:
    A-C, D1, P

What registers/RAM may be used if not handled?:
    A-C, D1, P

What registers/RAM may be used if error exit:
    A-C, D1, P

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.45 pSREC# - Position to Rec# of File w/Copycode 8

Category: POLL    File: SC&FIL::MS

Name:(S) pSREC# - Position to Rec# of File w/Copycode 8

Type: POLL

Purpose:
    Set file pointer to a given record # of a file whose copy code is >= 8.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
 Set the file pointer in FIB to a given record # in the file.

Entry conditions for handler (registers, ST, RAM, etc.):
 B[A] = Poll number.
 HEX mode.
 P=0.
 A[s] = copy code of the file
 A[4-0] = FIB entry address of the file
 SIMTD1 = FIB entry address of the file
 R1 = Record # (first record of the file is record 0)

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
 Carry clear.
 HEX mode.
 XM=0.
 Following field in FIB of the file is updated:
 .Current position set to start of the given record.
 .# of bytes left in current is set to equal to record length.
 .If the file is in external device, the file I/O buffer should contain the current sector.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
 Carry clear (POLL only).
 HEX mode.
 XM=1.

Error exit conditions from handler (POLL only):
 Carry set.
 HEX mode.

Available subroutine levels:

What registers/RAM may be used if handled?:
 All CPU registers
 Don't use SIMTD0 & SIMTD1

What registers/RAM may be used if not handled?:
 All CPU registers
 Don't use SIMTD0, SIMTD1, R1

What registers/RAM may be used if error exit ?:
 All CPU registers

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.46 pRDCBF - Read Current Sector From Mass Memory

Category: POLL File: SC&FIL::MS

Name:(S) pRDCBF - Read Current Sector From Mass Memory

Type: FPOLL

Purpose:
Using the FIB, read the current sector of a file in the mass memory device into the I/O buffer that is associated with the file.

There are total of 3 polls can be used to read/write a sector between a mass memory device and I/O buffer:
1. pRDNBF - Write buffer out to current sector and read in next sector. If buffer content has not been altered, just read in next sector.
2. pRDCBF - Read in current sector from mass memory device to the I/O buffer. This poll does not care about the content currently in the I/O buffer.
3. pWRCBF - Write I/O buffer to current sector in the mass memory device.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Read the current sector from the mass memory device into the I/O buffer of the file.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
STMTD1 contains the FIB entry address of the file
(SNAPBF contains A, D, D0 and D1 to restore on exit)

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
- Carry clear
- HEX mode.
- XM=0.
- File access nib in FIB is set to zero.
- A,D,D0,D1 restored to values from SNAPBF.

This poll handler must always call the routine SNAPRS to restore A,D,D0,D1 from the snap save RAM before return. So if the polling routine calls the SNAPSV before issuing this poll, it can consider A,D,D0,D1 will not be changed by this poll handler.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
- HEX mode.
- XM=1.

Error exit conditions from handler (POLL only):
Won't return to calling routine if error occur, direct exit to BSERR routine.

Available subroutine levels:
3

What registers/RAM may be used if handled?:
A-D, D0, D1, P, ST[0-4]
(B,C,P,ST[0-4] if SNAPSV was called)

What registers/RAM may be used if not handled?:
B, C, D0
(SNAPRS is not called)

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.47 pWRCBF - Write I/O Buffer to Mass Memory Device

Category: POLL File: SC&FIL::MS

Name:(S) pWRCBF - Write I/O Buffer to Mass Memory Device

Type: FPOLL

Purpose:
Using the FIB, write the file I/O buffer to the sector it came from in a mass memory device. Buffer content, current position and record address are not changed by this operation.

There are total of 3 polls can be used to read/write a sector between a mass memory device and I/O buffer:
1. pRDNBF - Write buffer out to current sector and read in next sector. If buffer content has not been altered, just read in next sector.
2. pRDCBF - Read in current sector from mass memory device to the I/O buffer. This poll does not care about the content currently in the I/O buffer.
3. pWRCBF - Write I/O buffer to current sector in the mass memory device.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Write the file I/O buffer to the sector it came from in a mass memory device.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
STMTD1 contains the FIB entry address of the file
(SNAPBF contains A,D1,D0 and D1 to restore on exit)

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=0.
File access nib in FIB is set to zero.
A,D,D0,D1 restored to value from SANPBF.

This poll handler must always call the routine SNAPRS to restore A,D,D0,D1 from the snap save RAM before return. So if the polling routine calls the SNAPSV before issuing this poll, it can consider A,D,D0,D1 will not be changed by this poll handler.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Error exit conditions from handler
Won't return to calling routine if error occurs, direct exit to BSERR routine.

Available subroutine levels:
3

What registers/RAM may be used if handled?:
A-D, D0, D1, P, ST[0-4]
(B,C,P,ST[0-4] if SNAPSV been called)

What registers/RAM may be used if not handled?:
B, C, D0
(SNAPRS is not called)

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.48 pCREAT - Create File in External Device

Category: POLL    File: SC&FIL::MS

Name:(S) pCREAT - Create File in External Device

Type: POLL

Purpose:
Create a file in an external device
This poll handles files of all copy codes except copy code 8.

Should poll be "Handled" (return with XM=0)?: Yes

Meaning of "Handling" Poll (what does code do if handled?):
Create a file in an external mass memory device

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

D(X) = Device address
D(S) = Device type
STMTR0 = First 8 chars of the file name
STMTR1(3,0) = Last 2 chars of the file name
STMTR1(6,5) = Offset to data (from file type table)
STMTR1(9,7) = Device address
STMTR1(13,10) = File type
STMTR1(14) = Create code (can not be 8)

R2(A) = First parameter for CREATE:

| Create code | Format Implied | Meaning of this parameter |
|---|---|---|
| 0 | Executable | Data length in nibs |
| 1 | DATA(fix length) | Number of records |
| 2 | SDATA(41C data) | Number of registers |
| 4 | LIF1 type (vbl len record) | File length in bytes |

R3(A) = Second parameter for CREATE:

| Create code | Format Implied | Meaning of this parameter |
|---|---|---|
| 0 | Executable | (ignored) |
| 1 | DATA(fix length) | Record length in bytes |
| 2 | SDATA(41C data) | (ignored) |
| 4 | LIF1 type(vbl len) | (ignored) |

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
  Carry clear (POLL only).
  HEX mode.
  XM=1.

Error exit conditions from handler (POLL only):
  Carry set.
  HEX mode.
  C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
  A-D, D0, D1, P ,R0-R4, S0-S11, SCRTCH RAM

What registers/RAM may be used if not handled?:
  A-D, D0, D1, P

What registers/RAM may be used if error exit (POLL only)?:
  Anything

NOTE:
  No future changes to this interface should cause the handler to alter statement scratch!!!

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/19/83 | SC | Document |

## 17.49 pCRT=8 - Create File w/Create Code = 8

Category: POLL    File: SC&FIL::MS

Name:(S) pCRT=8 - Create File w/Create Code = 8

Type: POLL

Purpose:
Create a file whose create code is 8. The file can be in internal memory or external mass memory device. The poll handler must handle all HPIL access.

Should poll be "Handled" (return with XM=0)?: Yes

Meaning of "Handling" Poll (what does code do if handled?):
Create the file.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

D(X) = Device address
D(S) = Device type
STMTR0 = First 8 chars of the file name
STMTR1(3,0) = Last 2 chars of the file name
STMTR1(6,5) = Offset to data (from file type table)
STMTR1(9,7) = Device address
STMTR1(13,10) = File type
STMTR1(14) = Create code (can not be 8)

R2(A) = First parameter for CREATE:

| Create code | Format Implied | Meaning of this parameter |
|---|---|---|
| 0 | Executable | Data length in nibs |
| 1 | DATA(fix length) | Number of records |
| 2 | SDATA(41C data) | Number of registers |
| 4 | LIF1 type (vbl len record) | File length in bytes |

R3(A) = Second parameter for CREATE:

| Create code | Format Implied | Meaning of this parameter |
|---|---|---|
| 0 | Executable | (ignored) |
| 1 | DATA(fix length) | Record length in bytes |
| 2 | SDATA(41C data) | (ignored) |
| 4 | LIF1 type(vbl len) | (ignored) |

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    Carry clear
    HEX mode.
    XM=1.

Error exit conditions from handler (POLL only):
    Carry set.
    HEX mode.
    C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
    A-D, D0, D1, P ,R0-R4, ST, SCRTCH RAM

What registers/RAM may be used if not handled?:
    A-D, D0, D1, P

What registers/RAM may be used if error exit (POLL only)?:
    Anything

NOTE:
    No future changes to this interface should cause the poll handler to alter Statement Scratch!

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/19/83 | SC | Document |

## 17.50 pFINDF - Find External File

Category: POLL    File: SC&FIL::MS

Name:(S) pFINDF - Find External File

Type: POLL

Purpose:

Find a given file in a given mass memory device

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Return file information about the file.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
R0 = First 8 chars of file name
R1 = Last 2 chars of file name
D(X) = Device address
D(S) = Device type

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.

R0(0,3) = Starting record #
R0(4,6) = Device address
R0(7,10)= 0000
R0(11,14)= File type
R0(15) = 8
R1(0) = Entry # in the record containing directory
R1(1,4) = Record # of directory entry
R1(5) = 0
R1(6,9) = # of sectors of file length

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
A,B,C,D(15,5),D1,R0,R1, P

What registers/RAM may be used if not handled?:
A,B,C,D[15-5],D1,R0,R1,P

What registers/RAM may be used if error exit:
A,B,C,D[15-5], P

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.51 pDIDST - Poll for Device ID Storage

Category: POLL File: SC&FIL::MS

Name:(S) pDIDST - Poll for Device ID Storage

Type: FPOLL

Purpose:
Handler for device ID storage (D1 @ destination point)

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Save the device ID in FIB for the file

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

R2 contains C[W] from SETUP
(R2[14] is the device code from FILSPx)
R3 contains the device ID/volume label

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.

XM=0.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels: 4

What registers/RAM may be used if handled?:
A-D, D0, D1, P R2-R3

What registers/RAM may be used if not handled?:
A-D, D0, D1, P, R2, R3

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.52 pDATLN - Compute File Len w/Create Code = 8

Category: POLL File: SC&FIL::MS

Name:(S) pDATLN - Compute File Len w/Create Code = 8

Type: POLL

Purpose:
Compute the file length of an external file whose create code is 8.

Should poll be "Handled" (return with XM=0)?: Yes

Meaning of "Handling" Poll (what does code do if handled?):
Return the file length of the external file

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.

P=0.
D[S] = copy code of the file, but already been shifted left once(top bit lost).
The directory entery of the file is copied from the mass memory into SCRTCH RAM (64 nibs)

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=0.
A(A) = File length of the file in nibbles.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels: 6

What registers/RAM may be used if handled?:
A-D, D0, P

What registers/RAM may be used if not handled?:
A-D, D0, D1, P

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 04/20/83 | SC | Document |

## 17.53 pREN - Renumber an XWORD line# reference

Category: POLL File: SC&REN::MS

Name:(S) pREN - Renumber an XWORD line# reference

Type: POLL

Purpose:
Renumber a XWORD statement if it has line number as its arguments.

Should poll be "Handled" (return with XM=0)?:Yes

Meaning of "Handling" Poll (what does code do if handled?):
Return D1 points to where the line number is.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
A[4-0] = LEX file ID and fcn #
D1 past the XWORD tokens.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=0.
D1 @ the line number token(tLINE# or tLITRL)
S3 = 1, if there are more than one line numbers followed.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Carry set.
HEX mode.
Will exit to MEMERR(Insufficient Memory).

Available subroutine levels: 5

What registers/RAM may be used if handled?:

A-D, D0, P

What registers/RAM may be used if not handled?:
A-D, D0, P

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, P

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/20/83 | SC | Document |

## 17.54 pCALSV - Poll to save local environment on CALL

Category: POLL File: SC&SUB::MS

Name:(S) pCALSV - Poll to save local environment on CALL

Type: POLL

Purpose:
Give any LEX file a chance to save its local environment when CALL is executed.

Should poll be "Handled" (return with XM=0)?:
Since this poll is intended to reach every LEX file, so XM should always set to 1 on return.

Meaning of "Handling" Poll (what does code do if handled?):
A LEX file can put a block of its local environment on top of the stack. When the ENDSUB is executed later on, the LEX file can use this block to restore its local environment.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
AVMEME(available memory end) is pointing at current top

of stack.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
- Carry clear (POLL only).
- HEX mode.
- XM=1.
- Update the AVMEME to point at the top of the block which just been put on to the top of the satck.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
- Carry clear (POLL only).
- HEX mode.
- XM=1.

Error exit conditions from handler (POLL only):
- Carry set.
- HEX mode.
- C[3,0] = Error code

Available subroutine levels:
5

NOTE:
Definition of the save block: (starting from lower addr.)

| Nibs | Meaning |
|---|---|
| 1-2 | LEX file ID |
| 3-5 | Block length in nibs(not include the 1st 5 nibs) |
| 6 | # of update addresses following |
| 7-n | Update addresses 5 nibs each |
| n-m | Anything else |

What registers/RAM may be used if handled?:
A-D, D0, D1, P ,R0-R3, ST, scratch RAM

What registers/RAM may be used if not handled?:
A-D, D0, D1, P ,R0-R3, ST, scratch RAM

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P, R0-R3, ST, scratch RAM

Special memory/pointer considerations (are pointers funny?):
None.

Envisioned application(s):
Allows a LEX file to stack and unstack local data that is not stored in a system buffer. This may be useful to applications which can be called recursively, since

system buffers are global and are not allocated recursively.

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/18/83 | SC | Document |

## 17.55 pCALRS - Poll to restore local environment

Category: POLL    File: SC&SUB::MS

Name:(S) pCALRS - Poll to restore local environment

Type: POLL

Purpose:
Give any LEX file a chance to restore its local environment when ENDSUB is executed.

Should poll be "Handled" (return with XM=0)?:
Since this poll is intended to reach every LEX file, so XM should always set to 1 on return.

Meaning of "Handling" Poll (what does code do if handled?):
A LEX file can restore its local environment saved at CALL time (by respond to pCALSV poll)

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
CALSTK is pointing at the first of all the save blpcks.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=1.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

Carry clear (POLL only).
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):

Carry set.
HEX mode.
C[3,0] = Error code

Available subroutine levels:

3

NOTE:

How to find the save block of your own :

Starting from the CALSTK, look for first 2 nibbles of each block for your LEX ID. All the save blocks are link listed. When your block is found, just use the information to restore your local environment, don't collapse the block. All the update addresses in the block are justified if memory had been moved.

What registers/RAM may be used if handled?:

A-D, D0, D1, P ,R0-R3, ST, scratch RAM

What registers/RAM may be used if not handled?:

A-D, D0, D1, P ,R0-R3, ST, scratch RAM

What registers/RAM may be used if error exit (POLL only)?:

A-D, D0, D1, P, R0-R3, ST, scratch RAM

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/18/83 | SC | Document |

## 17.56 pFNIN - Poll at start of multiline U.D.F.

Category: POLL File: SC&SUB::MS

Name:(S) pFNIN - Poll at start of multiline U.D.F.

Type: FPOLL

Purpose:
Poll before start execution of a multiline user-defined function.

Should poll be "Handled" (return with XM=0)?:
If handled set XM=1 on return.

Meaning of "Handling" Poll (what does code do if handled?):
This poll give everybody a chance to do something, so the poller doesn't care it will be handled or not.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels: 4

What registers/RAM may be used if handled?:
Everything but the R1

What registers/RAM may be used if not handled?:
A-C, D[15-5] D0, D1, P
--NOTE: D[A] is sacred
R1-R4, ST, scratch RAM

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/10/83 | SC | Document |

## 17.57 pFNOUT - Poll at end of multiline U.D.F.

Category: POLL File: SC&SUB::MS

Name:(S) pFNOUT - Poll at end of multiline U.D.F.

Type: FPOLL

Purpose:
Poll before exiting a multiline user-defined function.

Should poll be "Handled" (return with XM=0)?:
If handled set XM=1 on return.

Meaning of "Handling" Poll (what does code do if handled?):
This poll give everybody a chance to do something, so the poller doesn't care it will be handled or not.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry set on entry.
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
HEX mode.
XM=1.

Available subroutine levels: 4

What registers/RAM may be used if handled?:
    Everything but the R0

What registers/RAM may be used if not handled?:
    A-C, D[15-5] D0, D1, P
    --NOTE: D[A] is sacred
    R1-R4, ST, scratch RAM

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/10/83 | SC | Document |

## 17.58 pRTNTp - Poll on Special Return type

Category: POLL    File: SG&EXC::MS

Name:(S) pRTNTp - Poll on Special Return type

Type: FPOLL

Purpose:
    Poll for Special Return type
    Allow for future extension of Special Return types on
      the GOSUB stack. When the RETURN is encountered,
      a LEX file may handled to do something before the
      RETURN (ex: Reactivate a Timer)

    Return types: 9-E are reserved for future implementation
    The GOTO+ entry point allows the special Return type to
      be passed on entry in R3(S)

Should poll be "Handled" (return with XM=0)?:
    No - if this poll is handled, it is a take over pollxxx

Meaning of "Handling" Poll (what does code do if handled?):
    Do the appropriate "special" return processing

```
     Pop address of GOSUB stack
     Perform the RETURN or POP
```

Entry conditions for handler (registers, ST, RAM, etc.):

```
     --Carry set on entry iff fastpoll--
     B[A] = Poll number (pRTNTp)
     HEX mode.
     P=0.
     R2(S) = Return type  (Range = 9-E)
     R2(A) = Return address
     sRETRN  (S0) = 1 if RETURN
                  = 0 if POP
     The address is NOT popped off the stack

     DO NOT destroy S0 or R2 while determining if handling
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

```
     HEX mode.
     Perform the "special" processing
     Pop the address off stack  (GOSBVL =POPGSB)
     If POP
        GOVLNG NXTSTM
     If RETURN
        If Return to Program (type must indicate this)
           Set PgmRun
           Save return address in R2
           Set D0 @ return address
           If TRACE needed                        (TRFLCK)
              TRACE TO                            (TRTO+)
           Set D0 @ Return address                  (R2)
           go execute "Return stmt"     (govlng RUNRT1)
        If Return to Keyboard
           If tracing                             (TRFLCK)
              Send CR/LF                          (CRLFSD)
           If Keyboard buffer to return to        (KBRTCK)
              Set D0 @ Return address             (R2)
              go execute "Return stmt" (govlng RUNRT1)
```

Sample code:

```
          GOSBVL   =POPGSB      Pop addr off stack
          C=D      A
          R2=A                  Save Return addres
          ?ST=0    sRETRN       POP?
          GOYES    RTN40
          ?ST=1    sRTNKY       Return to Keyboard ?
          GOYES    RTN20
          ST=1     PgmRun       Set Pgm Running flag
          A=R2                  Return address
```

```
        D0=A
        GOSBVL  =TRFLCK     Tracing ?
        GOC     RTN10       No
        GOSBVL  =TRTO+      TRACE TO
RTN10   A=R2
        D0=A                D0 @ Return address
        GOVLNG  =RUNRT1     Execute Return stmt

* Return to keyboard

RTN20   GOSBVL  =TRFLCK     Tracing ?
        GOC     RTN30
        GOSBVL  =CRLFSD     Send CR/LF
RTN30   GOSBVL  =KBRTCK     Keyboard buffer?
        GOTO    RTN10       Yes, go execute
*
* POP
*
RTN40   GOVLNG  =NXTSTM
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    HEX mode.
    XM=1.
    sRETRN (S0) and R2 must be be preserved--

Available subroutine levels: 7
    --FPOLL handler is two levels deeper than caller--
    RETURN/POP is statement execute: all levels available

NOTE:
    See GOTO+ entry for pushing special return type on
        GOSUB stack
    The return type must NOT conflict with other
        GOSUB/RETURN extended statements
    The return type or somewhere else --- must reflect
        if return to PROGRAM or KEYBOARD. This is
        determined at GOSUB time from PgmRun flag

What registers/RAM may be used if handled?:
    --A-D, D0, D1, P always available--
    --Statement execute usage

What registers/RAM may be used if not handled?:
    --A-C, D[15-5] D0, D1, P always available (FPOLL only)--
    --NOTE: D[A] is sacred in FPOLL!!--
    --R0,R1,R3,R4

Envisioned application(s):
    Special GOSUB statement:

```
ON INTERUPT GOSUB....
ON ALARM GOSUB....
ON ... GOSUB....
```

where "something" must be done before the actual return is exeucte. For example, schedule an ALARM

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/02/83 | J.P. | Changed to Fast Poll |

## 17.59 pFILXQ - Poll for device to return device ID

Category: POLL File: SG&FXQ::MS

Name:(S) pFILXQ - Poll for device to return device ID

Type: POLL

Purpose:
Polls for dedicated device to intervene to return its id

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Reads device specifier (either as an executed string expression off stack or as a literal) and if their device is referenced, return device ID in D(S) & D(X)

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
R0 contains file name (if any) - <=8 characters
D0 may be restored prior to filespec, using STMTD0
(this is generally not useful)
IF S7=0

```
          Device specifier is a literal
          D0 points past tCOLON (Poll handler must check
           to ensure that D0 points to tLITRL - if it doesn't
           poll should NOT be handled.
        IF S7=1
          Device specifier is a string on the stack
           (string header pointed to by AVMEME)
          D0 points past the entire file specifier
          A colon was found on the stack, in the appropriate
          position.
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):

```
        Carry clear
        HEX mode.
        XM=0.
        File Name in A (Retrieve from R0 before exit)
        D(S),D(X) set appropriately with device id
        D0 past file specifier
        ADDITIONALLY:
          IF S7=1 on entry, then D1 must point past the string
           on the stack and AVMEME must reflect this.
```

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

```
        Carry clear
        HEX mode.
        XM=1.
        R0 must be unaltered from entry.
```

Error exit conditions from handler (POLL only):

NO ERROR - Instead DON'T HANDLE

Available subroutine levels:

6

NOTE:

What registers/RAM may be used if handled?:

A-D,D1,D0,R1,STMTR1 (all of it), S1,S2

What registers/RAM may be used if not handled?:

A-D, D0, D1, P
R1,STMTR1 (all of it), S1,S2

What registers/RAM may be used if error exit (POLL only)?:

NO error exit

Envisioned application(s):

So a dedicated device may be referenced analogous to an

HPIL device, eg > INITIALIZE :HP145XX

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/21/83 | S.W. | Added POLL & documentation |

## 17.60 pFSPCx - File Spec Execution poll

Category: POLL     File: SG&FXQ::MS

Name:(S) pFSPCx - File Spec Execution poll

Type: POLL

Purpose:
POLL for file specification execution

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Returns 1st 8 chars of file name in A
and last two characters in R0
D(S)=Device type; D(X)= Device address

Entry conditions for handler (registers, ST, RAM, etc.):
```
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
Low 2 bytes of R0 are blank-filled
S7=1 => String expression on stack
        Top of stack pointed to by AVMEME
        D0 past string expression
  =0 => Literal
        D0 may be restored from STMTD0 to interpret
         file specifier
```

Normal exit conditions from handler if handled (ST, RAM,

registers, etc.):
    Carry clear
    HEX mode.
    XM=0.
    A contains file name (blank-filled)
     If > 8 characters, last 9 & 10 in R0
    If no file name specified, A=0
    D(S) = Device type
    D(X) = Device address
    D0 past file specifier
    S7 intact from entry
    If S7 set on entry (string), D1 must point past file
     specifier on stack; eg D1 must reflect new top of stk

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    Carry clear
    HEX mode.
    XM=1.

Error exit conditions from handler:
    Carry set.
    HEX mode.
    C[0-3] = Error number.

Available subroutine levels:
    7

NOTE:
    If not handled, error generated is eFSPEC

What registers/RAM may be used if handled?:
    A-D, D0, D1, P
    R0,R1, S-R1-0 thru S-R1-3, STMTD0, STMTD1,
    S1,S2

What registers/RAM may be used if not handled?:
    A-D, D0, D1, P
    Same as if handled (See above)
    S7 must remain intact

What registers/RAM may be used if error exit:
    A-D, D0, D1, P
    Same as if handled (See above)

Special memory/pointer considerations (are pointers funny?):
    No

Envisioned application(s):
    PURGE A:TAPE

Note:
If not handled, error generated is eFSPEC.

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/04/83 | S.W. | Added documentation |

## 17.61 pPURGE - Poll to PURGE file on external device

Category: POLL File: SG&FXQ::MS

Name:(S) pPURGE - Poll to PURGE file on external device

Type: POLL

Purpose:
Polls to PURGE a file on non-mainframe device

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Purges the file
The mainframe will handle purging any associated FIB

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
D0 past file specifier
D(S) & D(X) contains device info
Blank-filled file name in A(W) & R0; R0 contains chars 9 & 10; If no file name given, A=0

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.

XM=0.
S8=0 (indicates current file was not purged)

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
7

NOTE:
If not handled, error generated is eFSPEC

--SCRATCH RAM TO CONSIDER BELOW:--
--STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
--LEXPTR.--

What registers/RAM may be used if handled?:
A-D, D0, D1, P
Anything available to statements
STMT/FN scratch, R0-R4, S0-S11

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
Same as if handled, except don't use R0 !

What registers/RAM may be used if error exit:
A-D, D0, D1, P
Same as if handled

Special memory/pointer considerations (are pointers funny?):
No

Envisioned application(s):
PURGE A:TAPE
PURGE :PORT(2) ! PURGE ALL on a plug-in EPROM perhaps

Note:
If not handled, error generated is eFSPEC.

History:

| Date | Programmer | Modification |
| --- | --- | --- |

06/29/82 S.W. Added documentation

## 17.62 pPRGPR - Poll to PURGE file on non-RAM device

Category: POLL File: SG&FXQ::MS

Name:(S) pPRGPR - Poll to PURGE file on non-RAM device

Type: POLL

Purpose:
Polls for PURGE of file on non-RAM memory device

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Checks File Protection.
If current file, ensure there's a workfile in mainframe
or room to create one - See Note below.
If not secure, purge the file.
Call RFAD-I with begin source in R0, offset in B(A)
and D1 pointing to S-R0-1 (which contains old enf of
file chain)
Have S10 set on exit iff a LEX file was purged.
S9 should be set on return iff current file purged

Mainframe will handle:
Deleting any associated FIB.
If current file purged (S9=1), SUSP annun. will be
cleared & new workfile created.
If LEX file purged (S10=1), will call LEXBF+
If current running file purged, S7 will be set.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
D1 at file header of file to purge
D(S) contains device type

```
      2 => ROM
      3 => EPROM
     D(B) contains port info

Normal exit conditions from handler if handled (ST, RAM,
registers, etc.):
     Carry clear.
     HEX mode.
     XM=0.
     PCADDR intact
     S9 set iff current file purged
     S10 set if LEX file purged

Normal exit conditions from handler if not handled (ST, RAM,
registers, etc.):
     Carry clear
     HEX mode.
     XM=1.

Error exit conditions from handler
     Carry set.
     HEX mode.
     C[0-3] = Error number.
     Possible errors are:
       eFPROT (file is SECURE)
       eMEM   (file is current, there's no workfile, and no
                room to create one)

Available subroutine levels:
     6

NOTE:
     If file to purge is current file, consult mainframe code
      between PRGF25 & PRGF35 to verify there's a workfile
      or room to create one.

     --SCRATCH RAM TO CONSIDER BELOW:--
     --STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
     --LEXPTR.--

What registers/RAM may be used if handled?:
     A-D, D0, D1, P
     R0,R1,R2,R3,S-R0-0,S-R0-1,S0-S7,S9-S11

What registers/RAM may be used if not handled?:
     A-D, D0, D1, P
     Same as if handled (see above)

What registers/RAM may be used if error exit:
     A-D, D0, D1, P
     Same as if handled (see above)
```

Special memory/pointer considerations (are pointers funny?):
No
However, it is important to consider that PURGE of current file CAN and SHOULD generate an insufficient memory error if there's no workfile in the mainframe & no room to create one.

Envisioned application(s):
PURGE A:PORT(2) ! where PORT#2 is an EPROM

Note:
If no one responds, error generated is eFACCS

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/29/82 | S.W. | Added documentation |
| 12/16/82 | S.W. | Eliminated check to distinguish ROM from other non-RAM devices |
| 12/16/82 | S.W. | Poll handler no longer requires entry point to PRGF40 |
| 06/03/83 | S.W. | Replaced call to CLSUSP with ZERPGM |

## 17.63 pRNAME - Poll to RENAME file on unknown device

Category: POLL File: SG&FXQ::MS

Name:(S) pRNAME - Poll to RENAME file on unknown device

Type: POLL

Purpose:
Polls to RENAME file on external device or on non-RAM memory device.

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Writes out new name to file header (or directory)

Should be ready to go on to NXTSTM

Entry conditions for handler (registers, ST, RAM, etc.):
- Carry clear
- B[A] = Poll number.
- HEX mode.
- P=0.
- D0 is past the file specifier
- Proposed new file name is in the SAVSTK area (or at least what WAS the SAVSTK area before poll) The 10 character blank-filled new file name is 112 nibbles LOWER in memory than where SAVSTK points (70 HEX).

D(S) >= 7 =>
- RENAME file on external device
- Name of file to rename is blank-filled in A(W); Characters 9 & 10 in R0
- D(S),D(X) contain device id
- In higher memory, adjacent to proposed file name given above, is its corresponding 5 nibble device id (Do a shift right circular to restore to original form).
- If poll isn't handled, default error is eFSPEC

D(S) < 7 =>
- RENAME file on non-RAM memory device
- D1 is at the file header
- D(S) contains memory type info
  - 1 => ROM
  - 2 => EPROM
- D(B) contains port number/extender
- If poll isn't handled, default error is eFACCS

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
- Carry clear
- HEX mode.
- XM=0.
- Ready to go on to NXTSTM

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
- Carry clear
- HEX mode.
- XM=1.
- R0 intact from entry.

Error exit conditions from handler:
- Carry set.
- HEX mode.

C[0-3] = Error number.

Available subroutine levels:
7

NOTE:
Error if:
1) No file name is specified, ie RENAME A TO :TAPE
(eFSPEC)
(This is default error given if D(S)>=7, else the handler MUST EXPLICITLY error)
2) Proposed file name is 'keys'
(eFSPEC)
3) File by that name already exists on the medium
(eFEXST)

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0-R4, S0-S11, STMT/FN scratch

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
R1-R3, S0-S11, STMT/FN scratch
Don't alter SAVSTK !

What registers/RAM may be used if error exit:
A-D, D0, D1, P
R0-R4, S0-S11, STMT/FN scratch

Special memory/pointer considerations (are pointers funny?):
No

Envisioned application(s):
RENAME A:<external device> TO B
RENAME A:PORT(3) TO B (where A is on EPROM)

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/16/82 | S.W. | Combined polls |
| 05/17/83 | S.W. | Added documentation |

## 17.64 pFPROT - [UN]SECURE or PRIVATE in non-RAM device

Category: POLL File: SG&FXQ::MS

Name:(S) pFPROT - [UN]SECURE or PRIVATE in non-RAM device

Type: POLL

Purpose:
Poll to SECURE/UNSECURE/PRIVATE file on external device or in non-RAM memory device

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Change file protection; ready to go on to NXTSTM

Entry conditions for handler (registers, ST, RAM, etc.):

```
Carry clear.
B[A] = Poll number.
HEX mode.
P=0.
D0 past file specification
D(S) >= 7 =>
            File on external device
            File name blank-filled in A(W);
             characters 9 & 10 in low nibbles of R0
            D(S),D(X) contains device identifier
            If poll not handled, default error is eFSPEC

D(S) <  7 =>
            File in non-RAM memory device
            D1 at file header
            D(S) contains memory type info
            D(B) contains port extender/number
            If poll not handled, default error is eFACCS

S11=1 => PRIVATE
       else
          S10=1 => UNSECURE
              0 => SECURE
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear.

HEX mode.
XM=0.
PCADDR intact (ready to go on to NXTSTM)

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):

Carry clear.
HEX mode.
XM=1.
S10,S11 intact from entry
If D(S)>=7, R0 must be intact from entry

Error exit conditions from handler:

Carry set.
HEX mode.
C[0-3] = Error number.
Only foreseen errors are for PRIVATE on a SECURE or non-executable file, which generates eFPROT, eFTYPE respectively.

Available subroutine levels:

7

NOTE:

For no file name specified, ie SECURE :<device>
if D(S)>=7, the default error for 'not handled' will be eFSPEC. But if D(S)<7, the handler MUST EXPLICITLY error on this.

What registers/RAM may be used if handled?:

A-D, D0, D1, P,R0-R4
STMT/FN Scratch, S0-S11

What registers/RAM may be used if not handled?:

A-D, D0, D1, P
R1-R3, S0-S9, STMT/FN Scratch
R0 if D(S)<7

What registers/RAM may be used if error exit :

A-D, D0, D1, P
R0-R4, S0-S11, STMT/FN scratch

Envisioned application(s):

SECURE A:TAPE
PRIVATE A:PORT(3)        where PORT#3 is EPROM

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/30/82 | S.W. | Added documentation |

12/16/82 S.W. Combined polls

## 17.65 pEDIT - Poll to position at non-BASIC file

Category: POLL File: SG&FXQ::MS

Name:(S) pEDIT - Poll to position at non-BASIC file

Type: POLL

Purpose:
Just gives the 'OK' to position at non-BASIC file

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Clears XM

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
D1 points at file header
A(A) contains file type#

Normal exit conditions from handler if handled
Carry clear (POLL only).
HEX mode.
XM=0.
D1 at file header
S11 preserved from entry (flags whether to CATalog)
P=0

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear (POLL only).
HEX mode.
XM=1.
S11 preserved from entry
P=0

Error exit conditions from handler (POLL only):
    Carry set=> Must be MEMERR
    HEX mode.
    C[0-3] = Error number.

Available subroutine levels:
    7

NOTE:
    If handled or not, S11 & D1 must be preserved

What registers/RAM may be used if handled?:
    A-D,D0,R0-R3,S6

What registers/RAM may be used if not handled?:
    B,C,D,D0,R0-R3,S6

What registers/RAM may be used if error exit (POLL only)?:
    N/A

Special memory/pointer considerations (are pointers funny?):
    N/A

Envisioned application(s):
    To designate non-BASIC file as current, so cursor keys could be used to 'scroll' through the file contents.

    Also possibly to be used in conjunction with the parse take-over poll, ie position at a non-BASIC file and enter lines.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 03/04/83 | S.W. | Added poll |

## 17.66 pFILDC - Polls for File Decompile

Category: POLL        File: SG&LDC::MS

Name:(S) pFILDC - Polls for File Decompile

Type: POLL

Purpose:
Polls for handler for device decompile

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Decompiled device specifier output & D0 updated.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
D1 at tCOLON
A(B) contains tCOLON
D0 past last decompiled character
D(A) contains the end of available memory

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
P=0
Carry clear
HEX mode.
XM=0.
D1 past the file/device specifier
File specifier output & D0 updated
D(A) preserved

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
P=0
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler:
(Only happens with insufficient memory)
P=0
Carry set.
HEX mode.

Available subroutine levels:
6

NOTE:
When D(A) is passed to the poll handler, it reflects what

the end of available memory WILL be once we get to the handler.

What registers/RAM may be used if handled?:
A-D, D0, D1, P always available
S8,S9
+Anything EXPRDC uses (R0,R1,R2,S0,S3,S10,S11)

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
Same as if handled (see above)

What registers/RAM may be used if error exit:
A-D, D0, D1, P
Same as if handled (see above)

Special memory/pointer considerations (are pointers funny?):
No

Envisioned application(s):
Decompile non-mainframe device

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 07/08/82 | S.W. | Added documentation |

17.67 pCAT - Poll for CAT on external device

Category: POLL File: SG&SYS::MS

Name:(S) pCAT - Poll for CAT on external device

Type: POLL

Purpose:
Handles CAT for files not in MAIN, plug-in memory, Independent RAM, or CARD

Should poll be "Handled" (return with XM=0)?:

Yes

Meaning of "Handling" Poll (what does code do if handled?):
Takes over command; exits ready to go to next statement

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
File name (if any) in A(W) & R0
If no file name, then A(W)=0
Device Specifier in D(S),D(X)

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
PCADDR intact

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
7

NOTE:

--SNAPBF, TRFMBF, LDCSPC
--LEXPTR.--

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0-R4, All of STMT/FN Scratch
Anything is available which is normally available to statements.
S0-S11

What registers/RAM may be used if not handled?:
Same as if handled, except can't use R0 (see above)

What registers/RAM may be used if error exit

A-D, DO, D1, P
Same as if handled (See above)

Special memory/pointer considerations (are pointers funny?):
None

Envisioned application(s):
CAT on TAPE, etc

Note:
If no one responds to POLL, error given is eFSPEC.

See pCAT$ for related poll

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/10/83 | S.W. | Added new documentation header |

17.68 pCAT$ - Poll for CAT$ on external device

Category: POLL File: SG&SYS::MS

Name:(S) pCAT$ - Poll for CAT$ on external device

Type: POLL

Purpose:
Creates buffer to execute CAT$ for external device (related to pCAT)

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Pushes string on stack; AVMEME points to string header

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.

HEX mode.
P=0.
S0 set
AVMEME points to string header on stack (string contains device name)
If the string is not null, it has already been reversed via REV$ (Characters in mem in ascending order)
PC (D0) saved in F-R0-0

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
String on stack, with AVMEME pointing to string header
F-R0-0 preserved from entry

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
5

NOTE:
If poll not handled, eFSPEC generated

--SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
--LEXPTR.--

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0-R4, S7-S11, FN Scratch except F-R0-0

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
Same as if handled (see above)

What registers/RAM may be used if error exit
A-D, D0, D1, P
Same as if handled (see above)

Special memory/pointer considerations (are pointers funny?):
No

Envisioned application(s):
To handle: CAT$(n,":TAPE")

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/17/82 | S.W. | Improved documentation |
| 07/07/82 | S.W. | Modified code before calling BF2STK to reference AVMEME instead of TFORM |
| 07/19/82 | S.W. | Push null string on stack when positive numeric argument too large -- used to error. |
| 10/20/82 | S.W. | Replaced call to DDOSET (DO<=AVMEMS) with call to LDCSET (DO<=OUTBS) |
| 12/06/82 | S.W. | Changed exit conditions for CAT poll as per N. Zelle |
| 12/13/82 | S.W. | Polls on unrecognized file spec Polls on file name (may be device name without preceding colon) |

## 17.69 pLIST - Poll for LIST on an external device

Category: POLL File: SG&SYS::MS

Name:(S) pLIST - Poll for LIST on an external device

Type: POLL

Purpose:
LISTS a file on an external device

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Checks protection
If file not PRIVATE, LISTS the file, ready to go on to NXTSTM

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
Blank-filed file name in A(W); R0 contains chars 9 & 10
If no file name specified, A=0
D(S) contains device id; D(X) contains device address

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
PCADDR intact

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
7

NOTE:
For no file name specified, the default error message for 'not handled' will be eFSPEC.

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0-R4, All Statuses except S13
Scratch RAM?

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
R1,R2,R3
Scratch RAM?
Statuses except S13
NOTE: R0 MAY NOT BE USED IF NOT HANDLED !!!

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P
R0-R4, Statuses except S13, Scratch Ram

Envisioned application(s):

Listing a file that resides on an external device.

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/01/83 | S.W. | Added documentation header to poll |

## 17.70 pLIST2 - POLL to LIST non-BASIC/non-KEY file type

Category: POLL File: SG&SYS::MS

Name:(S) pLIST2 - POLL to LIST non-BASIC/non-KEY file type

Type: POLL

Purpose:
POLLS to LIST a mainframe file that isn't BASIC or KEY

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
LISTs the file on the display device
Clears XM
Ready to go to NXTSTM

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
D1 at file header start
A(A) contains file type#
D0 past file specifier

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
Ready to go on to NXTSTM - PCADDR intact

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    Carry clear
    HEX mode.
    XM=1.

Error exit conditions from handler
    Carry set.
    HEX mode.
    C[0-3] = Error number.

Available subroutine levels:
    7

    STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC, LEXPTR.

What registers/RAM may be used if handled?:
    A-D, D0, D1, P always available
    R0-R4, ST, scratch RAM

What registers/RAM may be used if not handled?:
    A-D, D0, D1, P always available
    R0-R4, ST, scratch RAM

What registers/RAM may be used if error exit (POLL only)?:
    A-D, D0, D1, P always available
    R0-R4, ST, scratch RAM

Envisioned application(s):
    LISTing files of types other than BASIC and KEY, eg perhaps TEXT or DATA files.

Default:
    If POLL not handled, error is Invalid File Type

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/04/83 | S.W. | Documented poll |

## 17.71 pMERGE - Polls to MERGE non-mainframe file

Category: POLL     File: SG&SYS::MS

Name:(S) pMERGE - Polls to MERGE non-mainframe file

Type: POLL

Purpose:
Polls to MERGE a non-mainframe file

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Merges designated file into current file (if BASIC), into keys file (if KEY), or other if some other file type and the command has been extended to allow this.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
A(W) contains first 8 characters of file name
R0(3-0) contains characters 9 & 10
D0 past file specifier

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=0.
PCADDR intact, ready to go on to NXTSTM.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.
R0 MUST be preserved from entry.

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
7

NOTE:
For no file name specified (MERGE :<device>)
the default error message for 'not handled' will
be eFSPEC.

What registers/RAM may be used if handled?:
A-D, D0, D1, P
R0-R4, all statuses except S13
All of STMT and FN scratch

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
R1,R2,R3; All statuses except S13
R0 can NOT be altered!
All of STMT and FN scratch

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P
R0-R4, All statuses except S13
All of STMT and FN scratch

Envisioned application(s):
Note that poll handler must check the following:
1) file type of specified file
2) Protection of source (can't be PRIVATE), and
of destination (can't be SECURE or PRIVATE).
3) Destination must be in RAM
4) Sufficient memory?

History:

| Date | Programmer | Modification |
|---|---|---|
| 04/18/83 | S.W. | Updated documentation |

17.72 pMRGE2 - Polls to MERGE non-BASIC,non-KEY file

Category: POLL File: SG&SYS::MS

Name:(S) pMRGE2 - Polls to MERGE non-BASIC,non-KEY file

Type: POLL

Purpose:
Polls for handling of MERGE of non-BASIC,non-KEY

Should poll be "Handled" (return with XM=0)?:
Yes

Meaning of "Handling" Poll (what does code do if handled?):
Does appropriate MERGE, checking file protection, and memory requirements, exits ready to go on to NXTSTM.

Entry conditions for handler (registers, ST, RAM, etc.):
Carry clear
B[A] = Poll number.
HEX mode.
P=0.
D1 at start of mainframe (source) file header
A(A)=File type#
D0 past file specifier

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=0.
RFADJ has been called to update necessary pointers,etc
Ready to go on to NXTSTM.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear
HEX mode.
XM=1.

Error exit conditions from handler (POLL only):
Carry set.
HEX mode.
C[0-3] = Error number.

Available subroutine levels:
7

NOTE:

--STMT/FN Scratch, SCRTCH, SNAPBF, TRFMBF, LDCSPC,--
--LEXPTR.--

What registers/RAM may be used if handled?:

A-D, D0, D1, P
R0-R4, All statuses except S13
All of STMT and FN scratch.

What registers/RAM may be used if not handled?:
A-D, D0, D1, P
R0-R4, All statuses except S13
All of STMT and FN scratch.

What registers/RAM may be used if error exit (POLL only)?:
A-D, D0, D1, P
R0-R4, All statuses except S13
All of STMT and FN scratch.

Envisioned application(s):
Perhaps merging TEXT or LEX files.
Could implement by using the EDIT poll to position
at the file, thereby making it the current file.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 04/18/83 | S.W. | Added documentation |

## 17.73 pWARN - Warning poll

Category: POLL File: TI&ERD::MS

Name:(S) pWARN - Warning poll

Type: POLL

Purpose:
Alert LEX files that a warning is about to go out.

Should poll be "Handled" (return with XM=0)?:
Only if you want the message to be entirely suppressed.
Most applications will "handle" the poll without
setting XM=0 (see below).

Meaning of "Handling" Poll (what does code do if XM=0?):
It's up to you. For instance, a LEX file might want to intercept all warnings and errors to write them to a file; in this case, do your thing and return with XM=0 so that the message is suppressed.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.
A(A)=0 if Quiet (flag -1) is to be checked,
else A(A)=FFFFF if Quiet is not to be checked.

```
         F  E  D  C  B  A  9  8  7  6  5  4  3  2  1  0
    RO: |  |     |                          |     |     |
         ^     ^                               ^     ^
         |     |                               |     |
         |  control codes for text insertion   |     |
         + Entry P value (see MFWRN)           |     |
                                  LEX ID of message  |
                                          message number
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=0.
P=0.
no other requirements -- the message will be suppressed

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=1.
P=0.
RO can be changed as needed to adjust msg (see MFWRN)

Error exit conditions from handler:
Carry set.
HEX mode.
C[0-3] = Error number.
P= value to select options in MFERR*

Available subroutine levels:
6

What registers/RAM may be used if handled?:
A,B,C,D,D0,D1,P,RO
(Not available: R1,R2,R3,R4,ST, scratch RAM)

What registers/RAM may be used if not handled?:
A,B,C,D,D0,D1,P,R0 (change R0 only to affect msg)
(Not available: R1,R3,R4,ST, scratch RAM.
R2 unavailable except for rare cases when insertion text is being passed to the msg routines.)

What registers/RAM may be used if error exit (POLL only)?:
A,B,C,D,D0,D1,P,R0
(Not available: R1,R2,R3,R4,ST, scratch RAM)

NOTE:
The pWARN poll (and other message polls) are usually "handled" without setting XM=0. This is to allow all LEX files to get a chance to intercept the poll.

A LEX file which intercepts the poll has essentially four choices:
1) Abort the warning message, continue executing or whatever else it wants to do (including jumping instead to the error routine).
2) Change the values in R0 to cause a different warning to be reported, or to cause different entry conditions as selected by the value in R0(S), or to cause different text insertion by changing the values in R2 (text insertion applies only to certain rare messages). Then allow the poll to return to the warning routine with XM=1.
3) Simply clear XM ("poll handled"). This causes the message to be suppressed; message driver returns immediately (without setting ERRN, etc.)
4) If error is generated by poll handler, set carry and load error number in C(3-0). This will cause a jump to BSERR with the new error number.

Envisioned application(s):
A) Foreign Language Translators: if the warning message number is from the appropriate LEX file, the message number in R0 is adjusted to generate the translator's message. (If a type {5} building block is included in the message, this will have to be adjusted through a nested pTRANS poll, too. See IDS volume I, chapter "Message Handling".) Set XM=1 and return.
B) Say a LEX file intercepts all warnings, writes the message number (ERRN) and line number (ERRL) to a file, and suppresses the display of the warning. When intercepting this poll, it would do the necessary processing and return with XM=0.
C) Say another operating system will not allow any warnings to be issued, only errors. It could

intercept the pWARN poll and jump directly to BSERR so that the warning is converted into an error.

D) An automated card puller (!?!?) might trap the appropriate card reader messages and use them as prompts when to insert and pull cards.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/05/82 | MB | Documentation |
| 01/27/83 | MB | Added "poll handled" suppress |

## 17.74 pERROR - Error poll

Category: POLL     File: TI&ERD::MS

Name:(S) pERROR - Error poll

Type: POLL

Purpose:
Alert LEX files that an error is about to go out.

Should poll be "Handled" (return with XM=0)?:
Only if you want the message to be entirely suppressed. Most applications will "handle" the poll without setting XM=0 (see below).

Meaning of "Handling" Poll (what does code do if XM=0?):
It's up to you. For instance, a LEX file might want to intercept all errors and warnings to write them to a file; in this case, do your thing and return with XM=0 so that the message is suppressed.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

```
     F E D C B A 9 8 7 6 5 4 3 2 1 0
RO: | |   |                 |   |   |
     ^   ^                    ^   ^
     |   |                    |   |
     | control codes for text insertion |   |
     + Entry P value (see MFERR*)   |   |
                     LEX ID of message  |
                             message number
```

If parse error (identified by bit3 in RO(S)=1xxx):
   Address in INBS points to input stream
   A(A)= address of error within input stream

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
- Carry clear.
- HEX mode.
- XM=0.
- no other requirements -- the message will be suppressed

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
- Carry clear.
- HEX mode.
- XM=1.
- RO can be changed as needed to adjust msg (see MFERR*)

Error exit conditions from handler:
- Carry set.
- HEX mode.
- C[0-3] = Error number.
- P= value to select options in MFERR* (caution: do not select a parse error in this manner -- A(A) cannot pass information back through the poll. i.e., do not set bit3 in P. If such a thing is necessary, the appropriate action is to abort the poll and jump directly to BSERR, MFERR or MFERR*.)

Available subroutine levels:
   5

What registers/RAM may be used if handled?:
   A,B,C,D,DO,D1,P,RO
   (Not available: R1,R2,R3,R4,ST, scratch RAM)

What registers/RAM may be used if not handled?:
   A,B,C,D,DO,D1,P,RO (change RO only to affect msg)
   (Not available: R1,R3,R4,ST, scratch RAM.
   R2 unavailable except for rare cases when insertion text is being passed to the msg routines.)

NOTE: If a parse error (bit3 in R0(S)=1xxx), then DO NOT call I/OALL, IODALL -- DO NOT allocate, deallocate or adjust the length of any I/O buffer! DO NOT change the value in AVMEMS or INBS! (I/O buffer routines move I/O buffer memory and change AvMemSt.) These pointers may be changed if the error is NOT a parse error.

What registers/RAM may be used if error exit (POLL only)?:
A,B,C,D,D0,D1,P,R0
(Not available: R1,R2,R4,ST, scratch RAM.
R3 is unavailable unless the error is a parse error; i.e., if R0(S)=1xxx.)

NOTE:
The pERROR poll (and other message polls) are usually "handled" without setting XM=0. This is to allow all LEX files to get a chance to intercept the poll.

Remember, if a parse error, do NOT change the values in AVMEMS or INBS! This prohibits any adjustment (or allocation/deallocation) of I/O buffer length.

A LEX file which intercepts the poll has essentially four choices:

1) Abort the error message, continue executing or whatever else it wants to do (including jumping instead to the warning routine).
2) Change the values in R0 to change the format of the message:
   i) change R0(4-0) to generate a different message
   ii) change R0(S) to select different options (see MFERR*). However, bit3 in R0(S) CANNOT be changed! Bit3 in R0(S) indicates a parse error; if you need to change this, the appropriate way is to jump directly to BSERR, MFERR or MFERR* with your own entry conditions.
   iii) change the values in R2 to change text insertion (text insertion applies only to certain rare messages).

   Then allow the poll to return to the error routine with XM=1.
3) Simply clear XM ("poll handled"). This causes the message to be suppressed; message driver returns immediately (without setting ERRN or ERRL, without checking ON ERROR!)
4) If error is generated by poll handler, set carry and load error number in C(3-0). This will

cause the new error to be displayed.
In addition, the poll handler may perform any housekeeping type functions, such as cleaning up pending operations.

Envisioned application(s):

A) Foreign Language Translators: if the error message number is from the appropriate LEX file, the message number in R0 is adjusted to generate the translator's message. (If a type {5} building block is included in the message, this will have to be adjusted through a nested pTRANS poll, too. See IDS volume I, chapter "Message Handling".) Set XM=1 and return.
B) Say a LEX file intercepts all errors, writes the message number (ERRN) and line number (ERRL) to a file, and suppresses the display of the error. When intercepting this poll, it would do the necessary processing and return with XM=0.
C) Say another operating system prevents any error from halting execution; instead it issues warnings and recovers without user intervention. It could intercept the pERROR poll and jump directly to MFWRN so that the error is converted into a warning. MFWRN is a subroutine, so processing would return to this operating system.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/05/82 | MB | Documentation |
| 01/27/83 | MB | Added "poll handled" suppress |

## 17.75 pMEM - Memory error poll

Category: POLL File: TI&ERD::MS

Name:(S) pMEM - Memory error poll

Type: FPOLL

Purpose:
Alert LEX files that an "Insufficient Memory" error is about to be reported.

Should poll be "Handled" (return with XM=0)?:
Only if you want the message to be entirely suppressed. Most applications will "handle" the poll without setting XM=0 (see below).

Meaning of "Handling" Poll (what does code do if XM=0?):
It's up to you. For instance, a LEX file might want to intercept all errors and warnings to write them to a file; in this case, do your thing and return with XM=0 so that the message is suppressed.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

```
     F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
RO: |   |0   0  |F  |                                |       |       |
     ^       ^   ^                                     ^       ^
     |       |   |                                     |       |
     |       | error flag                              |       |
     |   insertion codes                               |       |
     + Entry P value (see MFWRN)                       |       |
                                        LEX ID of message      |
                                                     message number
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=0.
P=0.
no other requirements -- the message will be suppressed

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=1.
P=0.
RO can be changed as needed to adjust msg (see MEMER*)

Available subroutine levels:
3

What registers/RAM may be used if handled?:

A,B,C,D,D0,D1,P,R0
(Not available: R1,R2,R3,R4,ST, scratch RAM)

What registers/RAM may be used if not handled?:
A,B,C,D,D0,D1,P
R0: change R0(3-0) to modify message
R0: change R0(14-13) to allow text insertion
(only if you're the LEX file that originated the message, and know what you're doing).
(Not available: R1,R2,R3,R4,ST, scratch RAM.)

NOTE:

The pMEM poll (and other message polls) are usually "handled" without setting XM=0. This is to allow all LEX files to get a chance to intercept the poll.

The message number is usually eMEM (18 hex, 24 dec). But any LEX file can call the MEMER* routine with its own message number; the fact that it called MEMER* means that it is reporting insufficient memory.

The MEMERR routine uses the leeway area in available memory as a building buffer; there is only enough room for about 80 characters, plus prefix. If a poll handler substitutes another message number, it cannot exceed an 80 character limit (a message should never be longer than about 25 characters, anyway). If it does, the computer would enter an infinite MEMERR loop.

A LEX file which intercepts the poll has essentially four choices:

1) Abort the error message, continue executing or whatever else it wants to do (including jumping instead to the warning routine).
2) Change the values in R0 to change the format of the message:
   i) change R0(4-0) to generate a different message
   ii) change R0(S) to select different options (see MFERR*). However, bit3 in R0(S) CANNOT be changed! Bit3 in R0(S) indicates a parse error; if you need to change this, the appropriate way is to jump directly to BSERR, MFERR or MFERR* with your own entry conditions.

   Then allow the poll to return to the error routine with XM=1.
3) Simply clear XM ("poll handled"). This causes the message to be suppressed; message driver returns immediately (without setting ERRN or

ERRL, without checking ON ERROR!)

4) Replace the address in level 1 of the RSTK (counting from 0) with its own address, so that after the message is displayed, processing returns to itself.

In addition, the poll handler can perform any housekeeping type functions (such as cleaning up pending operations).

One other option deserving mention is that of generating a memory error which calls for text insertion. For instance, say an external system has 6 different files open, and is writing to them randomly; it reaches insufficient memory while writing to FILE4, so wants to report:

    Write Limit: FILE4

using a text insertion point to pass "FILE4". Before calling MEMER*, set up R2 for insertions. When handling the pMEM poll, verify that this is indeed your message, adjust R0(14-13) to contain the insertion codes, and return with XM=1.

Envisioned application(s):

A) Foreign Language Translators: if the error message number is from the appropriate LEX file, the message number in R0 is adjusted to generate the translator's message. Set XM=1 and return.

B) Say a LEX file intercepts all errors, writes the message number (ERRN) and line number (ERRL) to a file, and suppresses the display of the error. When intercepting this poll, it would do the necessary processing and return with XM=0.

C) Say another operating system prevents any error from halting execution; instead it issues warnings and recovers without user intervention. It could intercept the pERROR poll and jump directly to MFWRN so that the error is converted into a warning. MFWRN is a subroutine, so processing would return to this operating system.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/05/82 | MB | Documentation |
| 01/27/83 | MB | Added "poll handled" suppress |

## 17.76 pENTER - Poll to ENTER Data From HPIL Device

Category: POLL File: TI&XTD::MS

Name:(S) pENTER - Poll to ENTER Data From HPIL Device

NOTE:
THIS POLL IS NOT ISSUED BY THE OPERATING SYSTEM. It is issued by the HP-71 HPIL Module and is fully documented in the HP-71 HPIL Module Internal Design Specification.

## 17.77 pTEST - Test Poll for Timing Polls

Category: POLL File: TI&XTD::MS

Name:(S) pTEST - Test Poll for Timing Polls

Type: POLL or FPOLL

Purpose:
THIS POLL IS NOT ISSUED BY THE OPERATING SYSTEM. It is a dummy poll which is used for timing the system overhead in issuing a poll. It should NEVER be handled.

Should poll be "Handled" (return with XM=0)?:
NO.

Meaning of "Handling" Poll (what does code do if handled?):
None.

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
    None.

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    Carry clear (POLL only).
    HEX mode.
    XM=1.

Error exit conditions from handler (POLL only):
    Carry set.
    HEX mode.
    C[0-3] = Error number (only Insufficient Memory)

Available subroutine levels:
    1

NOTE:
    This poll is for timing purposes only, and should never be handled.

What registers/RAM may be used if handled?:
    None.

What registers/RAM may be used if not handled?:
    A-C, D[15-5], D0, D1, P

What registers/RAM may be used if error exit (POLL only)?:
    A-D, D0, D1, P

Special memory/pointer considerations (are pointers funny?):
    None.

Envisioned application(s):
    None.

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/12/83 | FH | Created documentation |

## 17.78 pTRANS - Poll to Translate a Message

Category: POLL File: TI&XTD::MS

Name:(S) pTRANS - Poll to Translate a Message

Type: FPOLL and POLL

Purpose:
THIS POLL IS NOT ISSUED BY THE OPERATING SYSTEM. It is only issued by the MSG$ keyword (in LEX file #82), and by language translator LEX files. It alerts a language translator LEX file that a message needs to be translated.

Should poll be "Handled" (return with XM=0)?:
Yes.

Meaning of "Handling" Poll (what does code do if XM=0?):
A language translator has substituted a foreign language message for the English one (the message number has been changed to generate a translated equivalent to the English message).

Entry conditions for handler (registers, ST, RAM, etc.):
B[A] = Poll number.
HEX mode.
P=0.

```
         F  E  D  C  B  A  9  8  7  6  5  4  3  2  1  0
    RO:  |                                   |  ^  |  ^  |
                      ^                         |     |
                      |                         |     |
            these 12 nibbles may contain needed |     |
            information for message handler     |     |
                                 LEX ID of message    |
                                            message number
```

Normal exit conditions from handler if handled (ST, RAM, registers, etc.):
Carry clear.
HEX mode.
XM=0.
RO(3-0)= new message number (see below)

Normal exit conditions from handler if not handled (ST, RAM, registers, etc.):
    Carry clear.
    HEX mode.
    XM=1.
    R0 untouched.

Available subroutine levels:
    3

What registers/RAM may be used if handled?:
    A,B,C,D(15-5),D0,D1,P
    The message number in R0(3-0) may be changed.
    (Not available: R1,R2,R3,R4,ST, scratch RAM)

What registers/RAM may be used if not handled?:
    A,B,C,D(15-5),D0,D1,P
    (Not available: R1,R2,R3,R4,ST, scratch RAM.

NOTE:

```
!!!  Because the pTRANS poll may be issued      !!!
!!!  as a nested poll from a pERROR poll,       !!!
!!!  you CANNOT change the values in AVMEMS     !!!
!!!  or INBS!  This prohibits any change in     !!!
!!!  length of an I/O buffer (including I/OALL, !!!
!!!  IODALL), since I/O buffer routines move    !!!
!!!  I/O buffer memory and adjust AvMemSt.      !!!
```

Since the pTRANS poll is usually issued as a fast poll, the poll handler cannot do an error exit ("carry set"). However, the mainframe poll routine can error out with Insufficient Memory while trying to issue a slow pTRANS poll.

Language translators for message tables are the only LEX files which should handle the pTRANS poll. The scheme behind message translation is as follows:

- -- mainframe message numbers (LEX ID 00) are replaced with (message number)+100hex. E.g., message number 002D (decimal 45 as expressed by ERRN) has the foreign language equivalent numbered 012D (decimal 1045 as expressed by ERRN).
- -- other message numbers (for LEX files numbered above 01) are replaced with (msg number)+80hex. E.g., message number FF1F (decimal 255031 as expressed by ERRN) has the foreign language equivalent numbered FF9F (decimal 255159 as expressed by ERRN).

See IDS volume I, chapter "Message Handling" for more details.

A language translator should not handle the pTRANS poll unless the LEX ID number of the message (found in R0(3-2)) is the appropriate one for translating.

The pTRANS poll is issued from two locations:

1) The MSG$ function (LEX file #82) issues a fast pTRANS poll to translate the desired message number. For example,
   MSG$(45)
   issues a pTRANS poll which, if intercepted by a language translator for mainframe messages, causes message number 1045 to be returned.
2) Language translators, in certain rare cases, may issue a slow pTRANS poll to translate a type{5} indirect message number. This is a nested poll, issued during a pWARN poll (for instance, mainframe message #88 , "IFM WRN L:", contains a type{5}, and causes a nested pTRANS poll). A nested pTRANS poll may also be issued during a pERROR poll, although no applications have yet been envisioned which might do this. A nested pTRANS poll should NEVER be issued from a pMEM poll; this means that any local equivalent to "Insufficient Memory" should NEVER have a type{5} cell.

   A pTRANS poll should never be nested within another pTRANS poll.

   When handling a pTRANS poll, don't change the contents of R0(15-4); these nibbles may contain information from a nested pWARN or pERROR poll.

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/22/83 | MB | MSG$ written for LEX file #82 |
| 10/23/83 | MB | Added pTRANS poll handling to translators |

```
+-----------------------------------------+-------------------+
|                                         |                   |
|  PTRUTL  -  Pointer Utilities           |  CHAPTER  18      |
|                                         |                   |
+-----------------------------------------+-------------------+
```

## 18.1 D=AVMS - Set D(A) to AVMEMS or AVMEME

Category: PTRUTL File: SB&EXC::MS

```
Name:(S) D=AVMS  -  Set D(A) to AVMEMS or AVMEME
Name:(S) D=AVME  -  Set D(A) to AVMEMS or AVMEME

Purpose:
     D=AVMS  :  Read AVMEMS into D(A)
     D=AVME  :  Read AVMEME into D(A)

Entry:

Exit:
     D(A)=memory location specified.
     C(A)=a copy of value in D1 at time of call

Calls:      None

Uses.......
 Inclusive: C(A),D(A)

Stk lvls:   0

History:

   Date     Programmer              Modification
 --------   ----------   -----------------------------------
 10/19/82   B.S.         Added documentation
```

## 18.2 GETAVM - Get Available memory limits

Category: PTRUTL File: SB&IO::MS

```
Name:(S) GETAVM  -  Get Available memory limits

Purpose:
     Reads (AVMEME) into C & D1 and (AVMEMS) into D(A)

Entry:

Exit:
     D(A)     =  (AVMEMS)
     C(A),D1  =  (AVMEME)

Calls:      D=AVMS

Uses.......
 Inclusive: C(A),D(A),D1

Stk lvls:   1

History:

   Date     Programmer               Modification
 --------   ----------   ------------------------------------
 10/18/83   B.S.         Updated documentation
```

## 18.3 D1=AVE - Set D1 to (AVMEME)

Category: PTRUTL File: SB&IO::MS

```
Name:(S) D1=AVE  -  Set D1 to (AVMEME)

Purpose:
     Reads (AVMEME) into D1 (and C(A))
```

Entry:

Exit:
    D1,C(A) = (AVMEME)

Calls: None

Uses.......
 Inclusive: C(A),D1

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/18/83 | B.S. | Added Documentation |

## 18.4 AVE=D1 - Update AVMEME From D1 or C

Category: PTRUTL File: SB&IO::MS

Name:(S) AVE=D1 - Update AVMEME From D1 or C
Name:(S) AVE=C - Update AVMEME From D1 or C

Purpose:
    Update AVMEME pointer to the value in D1 or C

Entry:
    AVE=D1 : D1 = new value for AVMEME
    AVE=C : C(A) = new value for AVMEME

Exit:
    C(A)=D1= Value stored into AVMEME

Calls: None

Uses.......
 Inclusive: C(A)

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/12/82 | B.S. | Added documentation |

## 18.5 D0=FIB - Set D0,C(A) to value at STMTD1

Category: PTRUTL File: SC&DAT::MS

Name:(S) D0=FIB - Set D0,C(A) to value at STMTD1

Purpose:
Set D0,C(A) to the value stored in STMTD1

Entry:

Exit:
D0,C(A) = (STMTD1)

Calls: None

Uses.......
Inclusive: D0,C(A)

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/06/83 | BS | Added documentation |

## 18.6 RFAD-I - Adjust Refs when mem moves to lower addr

Category: PTRUTL File: SG&FXQ::MS

```
Name:(S) RFAD-I  -  Adjust Refs when mem moves to lower addr
Name:(S) RFAD--  -  Adjust Refs when mem moves to lower addr

Purpose:  Adjusts address references on the FOR/NEXT & GOSUB
          stacks, in FIBs, as well as RAM pointers
          (PCADDR -> TMRAD3) & (CURRST -> AVMEMS), when
          appropriate; this is to be used when part of
          program memory moves to lower address space
          (hence a negative offset will be added to the
          references)

          RFAD-- entry is used to adjust pointers when the
          file chain in MAIN has moved.

          RFAD-I entry is used to adjust pointers when a
          file chain in an IRAM has moved.

Entry:
          B(A) = Bgn destination - Bgn source  (offset)
          R0 contains Begin Source
          2 entry points:
              1) RFAD-- - End Source assumed to be (AVMEMS).
              2) RFAD-I - D1 points to a 5-nibble location
                          containing the address of the file
                          chain end.

Exit:     B(A)=offset
          R0=Bgn Source
          R1=Bgn Destination
          Carry Clear
          RFAD-I entry point  - D1 preserved
          All other entry pts - D1 pts to AVMEMS ram loc.

Calls:    RFUPD-, RFAD58
          LXFND, CSRC10, CSLC5, FORUPD, RFAD97, BUFFIB,
          PRVADR, I/OFND, RFUPD+, RFAD86

Uses:     A, C, D, R0, R1, D0, D1

Stack lvls: 2 (PCUPD1)
```

**Detail:** Zeroes out references on the GOSUB & FOR-NEXT stacks which point into purged address space.

**Note:** Memory must be moved BEFORE calling this routine!

**History:**

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 12/29/82 | S.W. | Updates CURRST -> AVMEMS |

## 18.7 RFUPD+ - Updates a ptr when mem moves

Category: PTRUTL File: SG&FXQ::MS

Name:(S) RFUPD+ - Updates a ptr when mem moves

Purpose:
Adds offset to given address reference, if memory movement to lower address space calls for such adjustment. Indicates if reference points to a part of memory that has just been purged.

Entry:
```
D(S)=0 => memory expansion, else memory contraction
R0=Bgn Source for MOVEUM
R1=Bgn Destination for MOVEUM
D0 points to RAM location containing address to
    check/update
D1 points to Ram location containing ptr to end source
B(A)=offset  (bgn destination)-(bgn source)
       This number will be negative!
```

Exit:
```
B, D, R0-R3, D0 & D1 are as they were upon entry
Carry set=> Reference into purged address space.
            A(A)=Bgn Destination
      clr=> Reference has been updated if needed.
            Correct reference in C(A) & in RAM pointed
            to by D0.
```

Calls: none

Uses.......
 Inclusive: A(A), C(A)

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |

## 18.8 FORUPD - FOR Stack Update

Category: PTRUTL File: SG&FXQ::MS

Name:(S) FORUPD - FOR Stack Update

Purpose:
Updates references on FOR-NEXT stack

Entry:
P = 0
R0 contains Begin Source
D1 points to location, containing End Source
If want appropriate references zeroed
have D(S)#0 and R1 containing Begin Destination
B(A) containing offset (Bgn Source)-(Bgn Dest)

Exit:
P = 0

Calls: RFUP++

Uses.......
 Inclusive: A(A), C(A), D(A), D0

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/28/83 | S.W. | Added routine |

## 18.9 RFADJ+ - Adjusts Refs When Mem Moves=>Higher Addr

Category: PTRUTL File: SG&FXQ::MS

Name: RFADJ+ - Adjusts Refs When Mem Moves=>Higher Addr
Name:(S) RFAD++ - Adjusts Refs When Mem Moves=>Higher Addr
Name:(S) RFAD+I - Adjusts Refs When Mem Moves=>Higher Addr

Purpose: Adjust address references on the FOR/NEXT & GOSUB stacks, in the FIBs, as well as the RAM locations PCADDR -> TMRAD3 & CURRST -> AVMEMS, to reflect instances of program memory expanding into higher address space.

Entry:

```
        B(A)=   Offset  (End Dest.)-(End Source)
                    This number will be positive!
        3 entry points:
             1) RFADJ+  - Bgn source in A(A).
             2) RFAD++  - Bgn source already in R0.
             3) RFAD+I  - D1 pointing to RAM location
                          containing pointer to end of
                          file chain - entry pt for IRAMS.
                          Bgn source already in R0.
```

Exit:

```
        B(A)=OFFSET; R0=BGN SOURCE; CARRY CLEAR
        C(S )=0 => Some address on GOSUB or FOR-NEXT
                   referenced block that moved
```

Calls: RFAD58, RFUPD+, RFAD85
RFAD97, BUFFIB, LXFND, CSRC10, CSLC5, FORUPD,
I/OFND, PRVADR, RFAD86

Uses: A, C, D, D0, D1, R0

Stk lvls: 2 (PCUPDT)

Detail: Needed when program mem moves to higher address space

Note: Memory must be moved BEFORE calling this routine!

History:

| Date | Programmer | Modifications |
|---|---|---|
| 07/01/82 | S.W. | Added documentation |
| 12/29/82 | S.W. | Updates CURRST -> AVMEMS |

## 18.10 LDCSET - Set D=AVMEME; D0=OUTBS

Category: PTRUTL File: SG&LDC::MS

Name:(S) LDCSET - Set D=AVMEME; D0=OUTBS
Name: D0=OBS - Set D=AVMEME; D0=OUTBS

Purpose:
Set D @ AVMEME, D0 @ OUTBS

Entry:
2 entry points:
1) LDCSET - Sets D(A) to AVMEME. Sets D0 to OUTBS.
2) D0=OBS - Sets D0 to OUTBS.

Exit:
All entry points:
C(A) = (OUTBS)
D0 @ (OUTBS)
Carry = Entry state
LDCSET only:
D(A) = (AVMEME)

Calls: D=AVME

Uses.......
Exclusive: C(A), D0, D(A) (LDCSET only)

Stk lvls: 1 (LDCSET), 0 (D0=OBS and D0=OUTB)

Detail:
    The carry must be PRESERVED due to call from AUTO

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | J.P. | Modified documentation |

## 18.11 D0=AVS - Set D0=address in AVMEMS

Category: PTRUTL    File: TI&ERD::MS

Name:(S) D0=AVS - Set D0=address in AVMEMS
Name:(S) D0=PCA - Set D0=address in PCADDR

Purpose:
    D0=AVS : Set D0=<AVMEMS> (also set A(A)=<AVMEMS>)
    D0=PCA : Set D0=<PCADDR> (also set A(A)=<PCADDR>)

Entry:
    No necessary conditions

Exit:
    D0=AVS : D0=A(A)=<AVMEMS>
    D0=PCA : D0=A(A)=<PCADDR>
    Carry not affected.

Calls: None

Uses:...... D0, A(A)

Stk lvls: none

Detail:

```
=D0=AVS D0=(5) =AVMEMS    Set D0= start of avail mem.
        GOTO   D0=DTO
=D0=PCA D0=(5) =PCADDR    Set D0= addr of xqtn line.
 D0=DTO A=DATO A          Set D0= address in DATO.
```

```
                D0=A
                RTN
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/05/83 | MB | Documentation |

## 18.12 MEMCKL - Check Avail Memory With, Without Leeway

Category: PTRUTL      File: TI&UTL::MS

```
Name:(S) MEMCKL  -  Check Avail Memory With, Without Leeway
Name:    MEMCK+  -  Check Avail Memory With, Without Leeway
Name:    CHKSPC  -  Check Available Memory With Leeway
Name:    CHKSPF  -  Check Available Memory Without Leeway
Name:(S) CHKmem  -  Check Available Memory Without Leeway
```

Purpose:
See if requested memory amount [+ Leeway] is less than or equal to available memory. Nonzero value of P on entry determines whether leeway will be included in check for some entry points. "Insufficient Memory" error code is returned with carry set if requested amount exceeds the available memory.

Entry:

```
   MEMCKL:
     C(A)     = Absolute amount memory to check
     P        = 0 iff LEEWAY to be added to amt being checked
   MEMCL+:
     B(A)     = Absolute amount memory to check
     P        = 0 iff LEEWAY to be added to amt being checked
   CHKSPC:    (LEEWAY ALWAYS added; B(A) not used)
     C(A)     = Absolute amount memory to check
     P        = 0
   CHKSPF:    (LEEWAY NEVER added; B(A) not used)
     C(A)     = Absolute amount memory to check
     D1       @ Available memory end pointer
   CHKmem:    (LEEWAY NEVER added; B(A) not used)
```

```
      A(A)    = Available memory end
      C(A)    = Absolute amount memory to check
      P       = 0

  Exit:
    Carry Clear:   Enough memory
      B(A)    = Amount to check  (MEMCKL, MEMCL+ only)
      A(A)    = Available Memory start
      D1      @ AVMEMS
      C(A)    = Available memory MINUS requested amount
                (MINUS Leeway if also checked)
      P       = 0
    Carry set:     Not enough memory
      B(A)    = Amount to check  (MEMCKL, MEMCL+ only)
      C(A)    = eMEM
      P       = 0

  Calls:       None

  Uses.......
   Inclusive: A(A),C(A),D1,B(A) (MEMCKL, MEMCL+ only)

  Stk lvls:   0

  Algorithm:

  MEMCKL: B <-- Requested Amount
  MEMCL+: C <-- B
          If P=0
  CHKSPC:    C <-- Leeway
             Amount = Req Amount + Leeway
             If overflow ---> Error Return
          D1 <-- AVMEME
  CHKSPF: A <--  Available Memory End
  Chkmem: Subtract Req Amount from Available Memory End
          If negative  -->     Error Return
          D1 <-- AVMEMS
          A <--  Available Memory Start
          Subtract Avail Memory start from subtracted amount
          If negative, then
             Error Return   [ C <-- eMEM ]
          else
             Return carry clear
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |
| 09/11/82 | JP | Added Leeway check code |
| 10/24/83 | FH | Updated documentation |

## 18.13 CLCOLL - Collapse Buffer Pointers to CLCSTK

Category: PTRUTL File: TI&UTL::MS

```
Name:     CLCOLL  - Collapse Buffer Pointers to CLCSTK
Name:     SYCOLL  - Collapse Buffer Pointers to SYSEN
Name:(S)  OBCOLL  - Collapse Output Buffer
Name:     BBCOLL  - Collapse Input, Output Buffer Pointers
Name:     OBPRD   - Read Output Buffer Pointers
Name:     OBLCMP  - Compute Output Buffer Length
Name:     INBS=C  - Set INBS to the Value in C
Name:     D1=IBS  - Set D1 to Start of Input Buffer
Name:(S)  D1@AVS  - Set D1 to Available Memory Start
Name:     AVS=D0  - Set AVMEMS to Value in D0
Name:     AVS=C   - Set AVMEMS to Value in C
```

Purpose:
Manipulate buffer pointers.

CLCOLL:
Collapse SYSEN, OUTBS, and AVMEMS to CLCSTK.

SYCOLL:
Collapse OUTBS and AVMEMS to SYSEN.

OBCOLL:
Collapse AVMEMS to OUTBS (collapse output buffer).

BBCOLL:
Collapse INBS, OUTBS, and AVMEMS to SYSEN (collapse both input and output buffers).

OBPRD:
Read output buffer pointers OUTBS and AVMEMS into C(A), A(A).

OBLCMP:
Compute length of output buffer = (AVMEMS) - (OUTBS).

INBS=C:
Set INBS to the value in C.

D1=IBS:
Set D1 to start of input buffer.

D1@AVS:
Set D1 to AVMEMS, A(A) to (AVMEMS).

AVS=D0:
Set AVMEMS to value of D0.

AVS=C:
Set AVMEMS to the value in C(A).

Entry:
No entry conditions assumed unless explicitly stated below.

INBS=C:
C(A) = Value to store in INBS.

AVS=C:
C(A) = Value to store in AVMEMS.

Exit:

CLCOLL:
C(A) = (CLKSTK)
D1 = 5 beyond AVMEMS
Carry = Clear

SYCOLL:
C(A) = (SYSEN)
D1 = 5 beyond AVMEMS
Carry = Clear

OBCOLL:
C(A) = (OUTBS)
D1 = 5 beyond AVMEMS
Carry = Clear

BBCOLL:
C(A) = (SYSEN)
D1 = INBS
Carry = Clear

OBPRD:
A(A) = (AVMEMS)
C(A) = (OUTBS)
D1 @ AVMEMS
Carry = Clear

```
    OBLCMP:
      A(A)   =  Length of output buffer -- (AVMEMS) - (OUTBS)
      C(A)   =  (OUTBS)
      D1     @  AVMEMS
      Carry  =  Clear

    INBS=C:
      C(A)   =  Entry state
      D1     =  INBS
      Carry  =  Clear

    D1=IBS:
      D1     @  Start of input buffer
      C(A)   =  INBS
      Carry preserved

    AVS=DO:
      C(A)   =  AVMEMS
      Carry  =  Clear

    AVS=C:
      C(A)   =  AVMEMS
      DO     @  C(A) entry value
      Carry  =  Clear
```

**Calls:** INITPT (CLCOLL,SYCOLL,OBCOLL only)

```
Uses.......
 Inclusive: C(A),D1          (CLCOLL,SYCOLL,OBCOLL,BBCOLL,
                               INBS=C,D1=IBS)
            A(A),D1          (D1@AVS)
            A(A),C(A),D1     (OBPRD,OBLCMP)
            C(A)             (AVS=DO)
            C(A),DO          (AVS=C)

Stk lvls:   0 (CLCOLL,SYCOLL,OBCOLL,OBPRD,INBS=C,
               D1=IBS,D1@AVS,AVS=DO,AVS=C)
            1 (OBLCMP)
            2 (BBCOLL)
```

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 09/16/82 | FH | Designed and coded. |
| 10/12/82 | FH | Added CLCOLL,SYCOLL,BBCOLL,INBS=C, D1=IBS,AVS=DO,AVS=C |
| 02/10/83 | FH | Removed IBPRD,OBSKIP,OBBACK |

```
+---------------------------------------------+------------------+
|                                             |                  |
|  SAVSTK  -  Save Stack Utilities            |  CHAPTER  19     |
|                                             |                  |
+---------------------------------------------+------------------+
```

## 19.1 SVINFO - Save/Read File Information

Category: SAVSTK     File: JP&EXC::MS

```
Name:(S) SVINFO  -  Save/Read File Information
Name:(S) SVINF+  -  Save/Read File Information
Name:(S) RDINFO  -  Read Source/Dest File Information
Name:    RDINFS  -  Read Source File Info
Name:    RDINFD  -  Read Dest File Info
```

Purpose:

These entry points are used by COPY, TRANSFORM, RUN, and CHAIN to save and access information on their source/destination files. The info is stored in an area on the SAVSTK, which must be allocated using ALINFO beforehand. SVINFO and SVINF+ write the file info; RDINFO, RDINFS, and RDINFD read the info back.

Entry:

```
All:    File Info save area allocated on SAVSTK

SVINFO:
 A       =  Filename   (first 8 chars)
 R0(3-0)=  Last 2 chars of filename
 D(A)    =  Device information
   D(0)    =  Device code
   D(4-1) =  Device spec (Port, extender#, etc)
              If PORT:
                 D(1) = Extender#
                 D(2) = PORT#
              If HPIL:
                 D(3-1) = Device address
                 D(4)   = Device characterization
 S3      =  0 => Save info in source file position
         =  1 => Save info in dest file position

SVINF+:  Same as SVINFO, except:
```

```
        D(S)   = Device code (position returned by FSPECx)
        D(3-0) = Device spec shifted right (in position
                 returned by FPECx)

       RDINFO:
        S3     = 0  if Source file info to be read
               = 1  if Dest file info to be read

       RDINFS, RDINFD:
        None.

   Exit:
        S4     = 0  (SVINFO, SVINF+)
               = 1  (RDINFO, RDINFS, RDINFD)
        S3     = 0  (RDINFS)
               = 1  (RDINFD)
               = Entry condition (RDINFO)

       SVINFO, SVINF+:  Information saved in appropriate spot
        A      =  Entry Condition
        RO(3-0)=  Entry Condition
        D(A)   =  Device information (see SVINFO entry)

       RDINFO:  Info on selected file
        A      =  Filename  (first 8 chars)
        RO     =  Last 2 chars of filename
        D(A)   =  Device information (see SVINFO entry)
        C(A)   =  D(A)

        RDINFS:  Same as RDINFO; Source information
        RDINFD:  Same as RDINFO; Destination information

  Calls:      None.

  Uses.......
   Inclusive: sDEST(S3),sREADI(S4),A,C,RO,D1,
              D(A) (RDINFO, RDINFD, RDINFS),
              D    (SVINF+)

  Stk lvls:   0

  Detail:
        Start addr  Size   Information
        ----------  ----  ------------------------------
        SAVSTK-50    20   Destination Filename
        SAVSTK-30     5   Destination Device Information
        SAVSTK-25    20   Source Filename
        SAVSTK- 5     5   Source Device Information

  History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |

## 19.2 SALLOC - Allocate Arbitrary Save Stack Block

Category: SAVSTK File: TI&UTL::MS

Name:(S) SALLOC - Allocate Arbitrary Save Stack Block
Name: ALINFO - Allocate File Info Save Stack Block

Purpose:
Allocates a block of the specified size on the Save Stack (SAVSTK). SALLOC allocates an arbitrary size, and ALINFO allocates the amount for the filespec info area used by COPY and TRANSFORM. Available memory is checked with or without LEEWAY, depending on the entry conditions.

Entry:
P = 0 if memory check to be performed with LEEWAY
# 0 if memory check to be performed without LEEWAY
SALLOC:
C(A) = Number of nibs to allocate

Exit:
P = 0
B(A) = Number of nibs allocated

Carry clear:
Allocation was successful
AVMEME updated
D1 @ Start of newly created Save Area
C=D0 on entry.

Carry set:
Allocation failed due to insufficient memory
C(3-0) = Error code (=eMEM)

Calls: MEMCKL,MOVEU3

```
Uses.......
 Exclusive: A(A),B(A),C(A),D1
 Inclusive: A,   B(A),C(A),D1

Stk lvls:   2

Detail:    If sufficient memory to allocate
              Save D0 on stack
              Move memory between SAVSTK --> AVMEME
              Update AVMEME
              Restore D0
```

```
+---------------------------------------+------------------+
|                                       |                  |
|  SAVUTL  -  Save Utilities            |  CHAPTER  20     |
|                                       |                  |
+---------------------------------------+------------------+
```

## 20.1 STATRS - Restore Status

Category: SAVUTL    File: FH&TFM::MS

```
Name:(S) STATRS  -  Restore Status
Name:    STATR+  -  Restore Status

Purpose:
     Restore status flags S11 - S0 and S13 from area saved
     by STATSV.  STATR+ merges specified bits from current
     status setting with restored S11 - S0.

Entry:
     D1      @  Save area written by STATSV
    STATR+:
     C(X)    =  Bits corresponding to status flags to be pre-
                served from current status setting during
                restore.

Exit:
     S13, S11 - S0 restored (merged w/input bits if STATR+)
     C(X)  =  Old S11 - S0
     Carry clear

Calls:       STATR+ calls STATRS which has no calls

Uses.......
 Inclusive: C(A), S13, S11-S0, A(A) for STATR+ only

Stk lvls:   0 (STATRS), 1 (STATR+)

History:

   Date     Programmer             Modification
 --------   ----------   ----------------------------------
 06/15/82       FH       Designed and coded.
```

## 20.2 STATSV - Save Status S13, S11 - S0

Category: SAVUTL File: FH&TFM::MS

Name:(S) STATSV - Save Status S13, S11 - S0

Purpose:
Save status flags S13, S11 - S0 in designated spot.

Entry:
D1 @ Start of 4-nib save area

Exit:
Save area written (see detail below)
Carry clear

Calls: None

Uses.......
Exclusive: C(A)
Inclusive: C(A)

Stk lvls: 0

Detail:
Save area:

| Nibs | Contents |
|---|---|
| 2-0 | Status S11 - S0 |
| 3 | 0 is S13 clear, 1 if set |

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/15/82 | FH | Designed and coded. |

## 20.3 RSTK<R - Restore RSTK Level(s) From RSTKBF Buffer

Category: SAVUTIL File: TI&UTL::MS

Name:(S) RSTK<R - Restore RSTK Level(s) From RSTKBF Buffer

Purpose:
Restore Return Stack Level(s) from circular buffer. Levels are saved and restored on a last-in-first-out (LIFO) basis (see R<RSTK for save routine). The buffer holds up to 16 levels. No more than 6 levels should be saved or retored in one call, however, since the return to the caller of RSTK<R requires one level.

Entry:
P = n - 1, where n is number of levels to restore (not counting return to caller of R<RSTK)

Exit:
Carry = Clear
P = 0
D0 @ RSTKBp RAM location

Calls: RSTK>1

Uses.......
Inclusive: C(A), C(S), B(A), D0

Stk lvls: n (n levels are ADDED to the stack on return)

NOTE:
The addresses stored in the buffer are NOT updated by RFADJ.

Detail:
The position in the circular buffer is indicated by the nibble =RSTKBp in System RAM, which points to the last position written.

During the routine:
C(S) = Level counter (from P on entry)
P = Circular buffer position (from =RSTKBp)
These counters are set up by routine RSTK>1, which is shared by RSTK<R and R<RSTK.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/14/82 | FH | Designed and coded |
| 02/24/83 | FH | Expanded buffer from 8 to 16 levels |

## 20.4 R<RSTK - Save RSTK Level(s) Into RSTKBF Buffer

Category: SAVUTL File: TI&UTL::MS

Name:(S) R<RSTK - Save RSTK Level(s) Into RSTKBF Buffer

Purpose:
Save Return Stack Level(s) in circular buffer. Levels are saved and restored on a last-in-first-out (LIFO) basis (see RSTK<R for restore routine). The buffer may hold up to 16 levels. No more than 6 levels should be saved or retored in one call, however, since the return to the caller of R<RSTK requires one level.

Entry:
P = n - 1, where n is number of levels to save (not counting return to caller of R<RSTK, which is not saved)

Exit:
Carry = Clear
P = 0
D0 @ RSTKBp RAM location

Calls: RSTK>1

Uses.......
Inclusive: B(A), C(A), C(S), D0 (R<RSTK)

Stk lvls: -n (n levels are REMOVED from stack on return)

NOTE:
The addresses stored in the buffer are NOT updated by RFADJ.

Detail:

The position in the circular buffer is indicated by the nibble =RSTKBp in System RAM, which points to the last position written.

During the routine:

```
     C(S)   =  Level counter (from P on entry)
     P      =  Circular buffer position (from =RSTKBp)
```

These counters are set up by routine RSTK>1, which is shared by RSTK<R and R<RSTK.

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/14/82 | FH | Designed and coded. |
| 02/24/83 | FH | Expanded to 16 use levels |

## 20.5 SNAPRS - Restore CPU Snapshot From SNAPSV Buffer

Category: SAVUTL    File: TI&UTL::MS

Name:(S) SNAPRS - Restore CPU Snapshot From SNAPSV Buffer
Name:(S) SNAPR* - Restore CPU Snapshot From Any Buffer

Purpose:
Restore registers saved by SNAPSV (A, D, D0, D1) and return saved stack level for caller to push onto stack.

Entry:

```
    SNAPRS:
     None.
    SNAPR*:
     D1     @  Starting address of save buffer + 42 decimal
```

Exit:

```
     D0     =  Value saved by last SNAPSV call.
     D1     =  Value saved by last SNAPSV call.
     A      =  Value saved by last SNAPSV call.
     B(A)   =  Stack level saved by last SNAPSV call.
     C(A)   =  Stack level saved by last SNAPSV call.
     D      =  Value saved by last SNAPSV call.
```

Carry = Clear.

Calls: None.

Uses.......
Inclusive: A, B(A), C(A), D, D0, D1

Stk lvls: 0

Detail:

SNAPSHOT SAVE BUFFER LAYOUT

| Offset into Buffer | Nibs | Register |
|---|---|---|
| 0 | 16 | A |
| 16 | 16 | D |
| 32 | 5 | D1 |
| 37 | 5 | D0 |
| 42 | 5 | Stack level |

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/10/82 | FH | Designed and coded. |

## 20.6 SNAPSV - Save Snapshot of CPU in SNAPSV Buffer

Category: SAVUTL File: TI&UTL::MS

Name:(S) SNAPSV - Save Snapshot of CPU in SNAPSV Buffer
Name: SNAPLC - Save Snapshot of CPU in Any Buffer

Purpose:
Save limited snapshot of CPU (1 stack level, A,D,D0,D1) to allow a routine to function without disturbing the registers of its caller. Useful for tight situations. Snapshot is saved in system RAM, and is restored by the

routine SNAPRS.

SNAPSV uses dedicated RAM locations for storage.
SNAPLC uses a "local" RAM location for storage.

Entry:
SNAPSV
C(A) = Stack level to be saved; popped by caller of SNAPSV.

SNAPLC
D1 @ Starting address of save buffer + 42 decimal
C(A) = Anything you want to save.

Exit:
B(A) = C(A) on entry
C(A) @ Save area start address + 42 decimal
Carry = Clear.

Calls: None.

Uses.......
Inclusive: B(A), C(A)

Stk lvls: 0

Detail:

SNAPSHOT SAVE BUFFER LAYOUT

| Offset into Buffer | Nibs | Register |
|---|---|---|
| 0 | 16 | A |
| 16 | 16 | D |
| 32 | 5 | D1 |
| 37 | 5 | D0 |
| 42 | 5 | Stack level |

History:

| Date | Programmer | Modification |
|---|---|---|
| 09/10/82 | FH | Designed and coded. |
| 11/15/82 | MB | Added SNAPLC entry |

## 20.7 SRLEAS - Release Arbitrary Block From Save Stack

Category: SAVUTL    File: TI&UTL::MS

```
Name:(S) SRLEAS  -  Release Arbitrary Block From Save Stack
Name:    RLINFO  -  Release File Info Block From Save Stack

Purpose:
     Release block of specified size from the Save Stack.
     SRLEAS releases a block of arbitrary size, while RLINFO
     releases a block the size of the filespec info area
     used by COPY and TRANSFORM.

Entry:
    SRLEAS:
     C(A)    =  Number of nibs to release.
    RLINFO:
     P       =  0

Exit:
     P       =  0
     D0      @  Old Av mem end
     D1      @  New Av mem end
     Carry   =  Clear

Calls:     MOVED3  (RLINFO falls into SRLEAS)

Uses.......
 Exclusive: A(A),B(A),C(A),D0,D1
 Inclusive: A,   B(A),C(A),D0,D1


Stk lvls:   0

Detail:
           Move Memory Down parameters:

           End Dest    = (SAVSTK)                     (D1)
           End Source  = (SAVSTK) - release           (D0)
           Length      = ((SAVSTK) - release) - (AVMEME) (C)
```

```
+------------------------------------------+-------------------+
|                                          |                   |
|   STDCMP  -  Statement Deconpile         |  CHAPTER  21      |
|                                          |                   |
+------------------------------------------+-------------------+
```

## 21.1 DSTRDC - Deconpiles Variable Declarations

Category: STDCMP  File: SG&LDC::MS

```
Name:(S) DSTRDC  -  Deconpiles Variable Declarations
Name:    DECDC   -  Deconpiles Variable Declarations

Purpose:  Deconpiles the following statements:
          INTEGER, SHORT, REAL, DIM, DESTROY, NEXT

Entry:    2 entry points:
          D(A) contains end of available memory
          P= 0
          D1 points into token stream
          D0 points into ascii output buffer
          1) DSTRDC - for statements with a possible
              keyword, eg TRACE and DESTROY.
          2) NXTDC
             DECDC  - For variable list, eg
              INTEGER, SHORT, REAL, DIM, NEXT

Exit:       A(B)=EOL TOKEN
            via OUTELA

Calls:    VARDC, ARYDC, OUTBYT, GTEXT+, EOLXC*

Uses:     A, C, S5,S6,S9, D1,D0
          A-C, R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Stk lvls: 6

History:

   Date      Programmer   Modifications
 --------    ----------   -------------
 08/18/82    S.W.         Added documentation
```

## 21.2 PRNTDC - Expression List Decompile

Category: STDCMP File: SG&LDC::MS

```
Name:(S) PRNTDC  -  Expression List Decompile
Name:(S) DISPDC  -  Expression List Decompile
Name:(S) FIXDC   -  Expression List Decompile
Name:(S) DROPDC  -  Expression List Decompile

Purpose:  Decompiles PRINT, DISP, POKE, FIX, SCI, ENG, FLAG,
           DELAY, WAIT, INPUT, READ, statements

Entry:    P=0
          A(B) contains token pointed to by D1
          D(A) contains available memory end (AVMEME)
          D1   input pointer
          D0   output pointer
          PRNTDC - Entry FOR PRINT, DISP
            Allows USING to precede expression list
          FIXDC  - Entry FOR FIX, SCI, & ENG
            Must be at least 1 expression in list
          DROPDC - Entry for DROP, ADD
            Optional expression list (none necessary)
          INPTDC - Entry for INPUT
          READDC - Entry for READ, READ#
          SFLGDC - Entry for SFLAG, CFLAG
            Decompiles ALL, MATH, or expression list

Calls:    OUT1TK, EXPRDC, GTEXT+, EOLXC*, LIN#DC, -EXPR-,
          CONTST

Uses:     A-C, D1,D0, S9
          A-C, D1,D0, R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Detail:   WILL WORK FOR ANY STATEMENT WHICH COMPILES TO A LIST
          OF EXPRESSIONS DELIMITED BY COMMA OR SEMI-COLON
          TOKENS.
          2 ENTRY POINTS:
            1) PRNTDC - FOR STATEMENTS WHICH OPTIONALLY ALLOW
                        A NULL LIST.
            2) DLAYDC - OTHERWISE

NOTE: tEND, tTAB, or \#\ MAY NOT BE USED AS A 'KLUDGE' TOKEN
```

BY ANY ROUTINES THAT USE THIS ROUTINE.

Stk lvls: 6

History:

| Date | Programmer | Modifications |
|---|---|---|
| 08/18/82 | S.W. | Added documentation |

## 21.3 ONDC - ON..GOTO,..GOSUB,..RESTORE Decompile

Category: STDCMP File: SG&LDC::MS

Name: ONDC - ON..GOTO,..GOSUB,..RESTORE Decompile
Name:(S) GOTODC - GOTO Decompile
Name:(S) ONDC20 - Keyword and Opt Line#/Label Decompile

Purpose:
ONDC decompiles ON..GOTO,.. GOSUB,..RESTORE statements

GOTODC entry decompiles an optional list of line numbers/ labels. It is used by GOTO, GOSUB, and RESTORE decompile in the mainframe.

ONDC20 entry decompiles a keyword within leading and trailing blanks, then decompiles an optional list of line numbers/labels.

Entry:
D(A) contains available memory end (AVMEME)
D1 points into token stream
D0 output pointer
P= 0
Entry points:
ONDC - D1 points to tERROR, tTIMER, or <expr>
GOTODC - D1 points to start of optional list of line numbers/labels.
ONDC20 - D1 points at keyword token preceding optional list of line numbers/labels

```
Exit:       Through PRNTDC

Calls:      EXPRDC,LIN#DC,LABLDC,OUTBYT,GTXT++,ETMRDC

Uses.......
 Exclusive: A-C, D1,D0,    S5,S9 (ONDC only)
 Inclusive: A-C, R0-R2, D1,D0, S0,S3,S8,S10,S11 - EXPRDC

Stk lvls:   6

Detail:
     ON ERROR (GOTO|GOSUB) (<lineno> | <label> )
     ON TIMER #<timer no>,  <#secs> (GOTO|GOSUB)
              (<lineno> | <label>)
     ON <exp> GOTO    <lineno>|<label> [,<lineno>|<label>]
              GOSUB
              RESTORE
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/13/82 | J.P. | Modified documentation |
| 08/29/83 | S.W. | Updated documentation |

## 21.4 RENMDC - PURGE, COPY Decompile

Category: STDCMP     File: SG&LDC::MS

```
Name:     RENMDC  -  PURGE, COPY Decompile
Name:(S)  PURGDC  -  PURGE, COPY Decompile
Name:     COPYDC  -  PURGE, COPY Decompile

Purpose:  Decompiles RENAME, PURGE, PRIVATE, COPY

Entry:    P= 0
          D1 past begin BASIC token
          D0 output file
          D(A) contains available memory end (AVMEME)

Exit:     via OUTEL1
```

Calls: FILDC, GTEXT+, EOLXC*, BLNKCK

Uses: A-C, D1,D0, R1,R2, S8,S9
A-C, D1,D0, R0-R2, S0,S3,S8,S10,S11 -- EXPRDC

Stk lvls: 6

Detail: NAME <file name>
PURGE <file specifier>|ALL|keys
RENAME [<file specifier>|keys] TO <file name>|keys
COPY [ <file spec> | KEYS | CARD | PCRD ] [ TO
[ <file spec> | KEYS | CARD | PCRD ] ]

Note: The TO clause is OPTIONAL in a COPY statement
In RENAME, the TO is ALWAYS there

<destination file> is optional in COPY
This requires an EOL Check to be done after <file2>
This does not affect RENAME Decompile

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 08/18/82 | S.W. | Added documentation |

```
+-----------------------------------------+------------------+
|                                         |                  |
|  STEXEC  -  Statement Execute           |  CHAPTER  22     |
|                                         |                  |
+-----------------------------------------+------------------+
```

## 22.1 ASNMNT - Perform Variable Assignment

Category: STEXEC    File: AB&ASN::MS

```
Name:(S) ASNMNT  -  Perform Variable Assignment
Name:    ASNSTO  -  Perform Variable Assignment

Purpose:
     Evaluate expression and assign it to a variable.
     ASNMNT evaluates (i.e., locates) destination variable.
     ASNSTO does not (and requires proper entry conditions
       for DEST subroutine).

Entry:
            ASNMNT - D0 @ Destination Variable token.
            ASNSTO - D0 1 byte before start of expression,
                     Entry conditions for DEST.

            S15 set if trace is desired.

Exit:       Top 16 nibbles of Mathstack in A,
            D0 @ end of Statement,
            D1 @ top of Mathstack.

Calls:      DEST, EXPEX-, SVTRC.  All STORE calls (below)

Uses:       Everything.

Stk lvls:   6

History:    SA and SC

   Date     Programmer             Modification
 --------   ----------   ---------------------------------
 05/26/82   SA           Personnel change
```

## 22.2 STORE - Store From Stack To Variable

Category: STEXEC    File: AB&ASN::MS

**Name:(S) STORE - Store From Stack To Variable**

**Purpose:** Store number or string in known register.

**Entry:** Exit conditions of DEST
D1 = (MTHSTK) = true top of Mathstack,
Top 16 nibbles of Mathstack in A.
Statement scratch has information set up by DEST.
S-R1-2= Address points at the variable name This address is for TRACE to decompile the variable name. If the content of S-R1-2 is zero, the assignment will not be traced.

**Exit:** Preserves D0,
D1 @ top of Mathstack,
R3 contains value stored in variable location (as opposed to the value in the RES register)

**Calls:** CPOLL, Create, INTGR, RESTOR, SHRT, STRASN.

**Uses:** Everything.

**Stk lvls:** 5 (TRACEA and CREATE)

**History:**

| Date | Programmer | Modifications |
|---|---|---|
| 03/14/83 | SW | R3 contains value stored |

## 22.3 ONERR - Execute branch of ON TIMER/ERROR

Category: STEXEC File: JP&EXC::MS

```
Name:     ONERR   -  Execute branch of ON TIMER/ERROR
Name:(S) ONTIMR   -  Execute branch of ON TIMER/ERROR

Purpose:
     Process ON TIMER execution
     Process ON ERROR execution

     Indicates code needed to process any statement with
     GOTO/GOSUB that interrupts program execution and wants
     TRACE.  This code must be duplicated

     The main difference is sXWORD should be set before the
     call to GOTO+.  This guarantees that all line# references
     will be searched for, incase the reference was never cleare??
     due to the LEX file being missing when clearing references.

     Example statement: ON INTR GOTO|GOSUB <stmt id>

Entry:
     C  @ GOTO | GOSUB of statement
     For ONTIMR: RSTK = Next Stmt Address
                 sONTMR = 1  (S6)
                 sONERR = 0  (S4)
                 A(S) = Timer #
     Duplicate this code for
     External Statement w/GOTO or GOSUB with interupt
     Make sure sXWORD is set before jumping to GOTO+
     This code will TRACE properly

Exit:
     Through GOTO+ to execute GOTO | GOSUB
     RSTK = Next Stmt address
     sEXTGS = 1
     If ONTIMER: sONTMR = 1
                 R3(S)  = Timer#
     If ONERROR: sONERR = 1

Calls:      TRFCK-,TRFROM,UPDPC,TRTO*,RACTMI,LNSKP-

Uses.......
 Exclusive: sGOSUB(S3),sEXTGS(S5),sONERR(S4),S6,S9,R1,R2,R3
```

```
 sGOSUB    S3 = GOSUB flag
 sONTMR    S6   = ON TIMER statement
 sEXTGS    S5   = External Entry flag for GOTO+
 sONERR    S4   = ON ERROR statement
 sXWORD    S9   = XWORD flag for searching for GOTOs
           RSTK = Return Address if GOSUB & ON TIMER
           R2   = Position @ <lineno> | <label> in stmt
                  Saved D0
           R3(S)= Timer#  (if ON TIMER)
           A(S) = Timer#  (if ON TIMER)

 RACTMI    uses R0,R1,R3,

Stk lvls:   <= 7  (statement execute)

Detail:

ONERR:   Clear status
         Set ON ERROR flag
         Compute next statement return addr(LNSKP-)
         Save on stack
ONTIMR:  Set External Entry flag          (sEXTGS)
         Set D0 = C  (position within ON statement)
         Read and skip over token
         Save D0 in R2
         Save Timer# in R3(S)
         Set GOSUB flag
         If GOTO
            Clear GOSUB flag
            If ON TIMER
               Reactivate timer          (RACTMI)
               Resave timer#             (R3(S))
         If trace needed                 (TRFCK-)
            Trace FROM line#             (TRFROM)
            Restore D0
         Update PC address to point to ON stmt
            If trace needed              (TRFCK-)
               Trace TO line#            (TRTO*)
         Restore D0
ONGTGB:  Clear XWORD flag                (sXWORD)
         go execute GOTO | GOSUB of statement
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/04/82 | JP | Modified documentation |
| 09/28/82 | JP | Changed ON TIMER implementation |
| 11/28/82 | JP | Changed interface to GOTO/GOSUB |
| 12/08/82 | JP | Fixed Timer# destroy by TRACE |
| 02/11/83 | JP | Clear sXWORD before GOTO+ jump |

| | | |
|---|---|---|
| 03/08/83 | JP | Removed sEXTGS set, clear @ ONERR |
| 03/31/83 | JP | Compute Rtnadr for ON ERROR |
| 03/31/83 | JP | Always update PCADRR @ ON stmt |

## 22.4 END - END, END ALL, END SUB, END DEF Statements

Category: STEXEC File: JP&SYS::MS

```
Name:    END     - END, END ALL, END SUB, END DEF Statements
Name:(S) ENDALL  - External Stmt entry to perform END ALL
Name:(S) ENDBIN  - End Binary Program or Subprogram
Name:    END10   - STOP Statement Execute
Name:    END20   - END SUB reentry
Name:    STOP    - STOP Statement Execute
Name:    EXITRN  - Clear status, return to BASIC loop
```

Purpose:

These entry points deal with terminating execution of in the current environment due to an explicit command such as END or STOP, or a SST past the last statement in the program. The running program may be BASIC or Binary.

END checks for ALL token
    checks for ENDSUB/ENDDEF
    Returns to BASIC loop allowing exceptions to be
    checked

S2 set will cause ending of execution so that:
   Exceptions not checked
   Program not suspended, CNTADR not updated

All entries but ENDALL collapse ONLY ONE level
ENDALL collapses to one level

Entry:

```
END:      DO past END token
          Checks for ALL token
STOP:
ENDBIN:
END10:    (Checks if END SUB or END DEF)
```

```
              (BASIC Loop entry if @ program scope end)
              (SST @ Program End entry)

              sSST (S2=1) if non-exception/nonprogram exit
                   (Clears PgmRun (S13=0), Clears S0-S11)
                   (Returns to BASIC loop without checking
                    exceptions)
                   (Prevents update of Cont Addr [CNTADR]
                    and SUSPend of program)
              Collapse stacks one level ONLY

     END20:   END SUB reentry
              sSST assumed cleared  (S2=0)
              If S2=1 acts like END10 entry

     ENDALL:  External Statement entry
              Sets sSST (S2=1) to avoid CNTADR update and
                   program suspension
              Avoids checking of exceptions in BASIC loop
              Clean-up for TRANSFORM current file
              Clean-up for PURGE current file
              Collapse stacks down to ONE level

     All entries, but ENDALL, collapse ONLY ONE level

  Exit:
     If END ---> sENDx (S1=1) for BASIC loop return
                   Prevents SUSPend of program
                 Through NXTST1 to avoid sENDx clearing
                   Returns to BASIC loop so exceptions
                     are checked
                 NoCont (S14=1) if within program
                   Causes BASIC loop execution to stop
     If END DEF or implied END DEF
            ---> Through ENDDEF
     If END SUB or implied END SUB
            ---> Through ENDSB-
     If SST @ PRGMEN or non exception check END desired
            ---> Through BSCEXT with PgmRun (S13) clear
                 Exceptions are not checked
                 Prevents CNTADR update and prgm SUSPension

     If non BASIC program
            ---> Through EXITRN
                 Clears S0-S11; exit BASIC loop (BSCEXT)
                 Exceptions are not checked
                 CNTADR not updated, program not SUSPended

  Calls:      CLRSTK, CLOSEA, CLPSTK, GETSTC, SUBCHK

  Uses:     A-D,P, D1, D0, CNTADR, sENDx (S1), sSST (S2),
```

```
          R0,R2,ALRM (+36), PNDALM (+1),STMID1,F1SUSP,PgmRun

Stk lvls:   6

Algorithm:
       If END ALL
          goto ENDAL1:
ENDBIN:
END10: If END DEF | END SUB
          go process appropriate statement
END20: Clear addresses, one level of stacks    (CLRSTK)
END30: Close all open files                     (CLOSEA)
       If non BASIC file                        (GETSTC)
          go Clear status and Exit BASIC loop   (BSCEXT)
       If non programmatic END desired          (sSST)
          Clear PgmRun to prevent SUSPend
          go Clear status and Exit BASIC loop   (BSCEXT)
       else
          If program running
             Set Don't Continue flag            (NoCont)
          Set END Execute flag                  (sENDx)
          Golong to end of BASIC loop through
                 NXTST1 to avoid sENDx clearing

ENDALL:Set sSST flag
ENDAL1:Collaspe stacks to one level             (CLPSTK)
       goto END30
```

Note:

The sENDx flag was originally used to distinquished END from all other statements/conditions that stop the BASIC loop exec. If a program had been running, this flag allowed CURRL to be updated to the END statement, but prevented the SUSP annunciator from lighting and the CONTINUE address from being updated.

This sSST flag was used to avoid any checking of a program running by returning to a different place in the BASIC loop, since CURRL could not be updated in situations like SST past the program end.

When the decision was made to update CURRL only when SUSPending the use of two flags is not that different. A "normal" END statement returns through NXTSTM to the BASIC loop. This causes exceptions to be checked before execution is stopped if a program was running. If from the keyboard, execution continues. If sSST is set (from SST past the end of the program or for TRANSFORMing the current file...) then the BASIC loop is reentered below the exception checking.

In either case, neither CNTADR is updated, nor SUSP lit.

History:

| Date | Programmer | Modification |
|---|---|---|
| 03/08/83 | JP | STOP === END SUB, END DEF |
| 03/17/83 | JP | Packed D1=(5) CALSTK |
| 04/25/83 | JP | CLRST thru EXITRN if sSST |
| 05/09/83 | JP | Clear PgmRun before EXITRN |
| 05/17/83 | JP | Check ENDSUB/DEF if SST at end of program (PRGMEN) |
| 06/05/83 | JP | END10 is Binary program return |
| 06/05/83 | JP | If nonBASIC prgm---> EXITRN |
| 06/05/83 | JP | ENDBIN entry point added |

## 22.5 GOTO - Statement Execution

Category: STEXEC File: JP&SYS::MS

Name:(S) GOTO - Statement Execution
Name:(S) GOSUB - Statement Execution
Name: RESTOR - Statement Execution

Purpose:
- Execution of GOTO | GOSUB
- Partial execution of ON, ON ERROR, ON TIMER
- Partial execution of RESTORE
- Partial execution of XWORD with GOTO/GOSUB within

Entry:

```
GOSUB:  D0 past GOSUB token       (Sets sGOSUB S3=1)
GOTO:   D0 past GOTO token        (Sets sGOSUB S3=0)
RESTOR: D0 past RESTORE token     (Sets S10=1)
        All status must be clear
GOTO+ : Entry for statement containing:
        GOTO | GOSUB <lineno> | <label>
        D0 @ <lineno> | <label> token past GOTO | GOSUB
        sEXTGS = 1 if External statement entry
        If GOSUB within statement
```

```
    sGOSUB = 1      (S3)
ON TIMER:       sONTMR = 1   (S6)
                sEXTGS = 1
                R3(S)  = Timer#
                RSTK   = Return address
ON ERROR:       sONERR = 1   (S4)
                sEXTGS = 1
                RSTK   = Return address
ON RESTORE:     sRESTR = 1   (S10)
External Entry:      (like ON INTRPT)
                sEXTGS = 1
                RSTK   = Return address
                sGOSUB = 1 if GOSUB
All other status MUST be clear!!!!


sXWORD = 1      (S9)
   If XWORD with GOTO | GOSUB
      Statement performing GOTO/GOSUB in  a
      "sequential" fashion. EX: ON <exp> GOTO

   Guarantees always search for Line# referen
   Eliminates problem of Line# reference address
   that is invalid because it was not cleared
   during PEDIT because the Lex File was missing.

 External statements with GOTO/GOSUB that
 interupt program execute (ex:ON TIMER,ON INTR)
 must duplicate ONTIMR code (see JP&EXC) to
 guarantee proper TRACE of program execution.
 sXWORD must be set before jumping to GOTO+

   R3(S) = Return type
     If "normal" GOSUB
        R3(S) = 0
     If GOSUB from Keyboard (PgmRun=0)
        R3(S) = 1
     If "special" GOSUB/RETURN
        See pRTNTp Poll in RETURN
        R3(S) = 9 through 15

        This allows special processing when
        RETURN of GOSUB is encountered

Assumes External Entry statements execute from a
Program, i.e. PgmRun (S13) is set.
```

Exit:
 to BSCX60
  Avoids exception checking until AFTER the branch

```
          Cleans up TRACE

       If RESTORE | ON RESTORE (sRESTR (S10))
          Jump to execute RESTORE
          Return to Run Loop thru NXTSTM

       If RESTORE #
          Jump to execute RESTORE #

       If GOTO from Keyboard
          Through NXTSTM after Setting CNTADR,CURRL

       If Error (Label | Line# not found)
          If ON ERROR stmt          (sONERR (S7))
             Zero out ON ERROR address
          If ON TIMER, ON ERROR or External Entry
             PCADDR has been updated to ON statement
          If ON TIMER
             Appropriate Timer# has be OFFed
          Set up Error Message
          goto MFERR

   Calls:      PFNDZL,FILXQT,FINDLB, PSHGSB,PRSCKB,
               TRFCK-,TRFROM,SNcrlf,TRTO+,CNTCUR,LNSKP-,
               SFGPGM,POPGSB,OFFTMR,CNTCK2,PSHUPD

   Uses.......
    Exclusive: A,C,S0,S3,S4,S5,S6,S7,S8,S9,S10,S13,S14,R0-R2,D0,
               S-R0-0 (1 nib)
    Inclusive: A-D,S0-S10,S13,S14,R0-R4,D0,D1,all FUNCTION scrtch
               S-R1-0 thru S-R1-3,STMTD0,S-R0-0 (1nib)

    PRSCKB uses R2; but its called only when NOT running
                    ON TIMER only active WHEN running

    RSTK    = Return address          (If ON TIMER)
    R1      = Saved D0
    R3(S)   = Timer #                 (If sONTMR)
    R3(S)   = Return type             (If sXWORD)
    sGOSUB  = GOSUB                   (S3)
    sONERR  = ON ERROR entry          (S4)
    sEXTGS  = External statement entry (S5)
    sONTMR  = ON TIMER entry          (S6)
    sXWORD  = XWORD entry for PFNDZL  (S9)
    sRESTR  = RESTORE statement entry (S10)
    PgmRun  = Program running         (S13)
    NoCont  = Don't Continue Run Loop (S14)
    S-R0-0  = Timer#

   Stk lvls:   7
```

```
Detail: (GOTO | GOSUB) (<lineno> | <label>)
        RESTORE [ <lineno> | <label> ]
        RESTORE # <assign#> [ <lineno> | <label> ]
        ON <exp> GOTO | GOSUB (<lineno>|<label>....)
        ON ERROR GOTO | GOSUB <lineno> | <label>
        ON TIMER # <exp>, <exp> GOTO | GOSUB <lineno>|<label>

 RESTOR:  If next token = #
             go Execute RESTORE#   (RESTR#)
 RESTOR:  Set RESTORE flag         (sRESTR)
          If (no <line#> | <label>)
             Set C=0 (Indicates start of file for DATPTR)
             go Execute RESTORE    (RESTRX)
          goto GOTO+
 GOSUB:   Set GOSUB flag           (sGOSUB)
          goto GOTO-
 GOTO:    Clear GOSUB flag
 GOTO-:   Clear RESTORE, ON ERROR, External Entry flag
 GOTO+:   Save D0                           (R1)
          If not running
             Set program scope              (PRSCKB)
          Check if trace needed             (TRFCK,TRFROM)
          Restore Timer# to A(S)            (R3(S))
          If GOSUB
             Pop Return Address of Stack incase ON TIMER
             If not ON TIMER
                Calculate Return Address   (LNSKP-)
             If XWORD
                go Push Return type/addr   (goto 0)
             Set Return Type = 0
             If ON ERROR                    (sONERR)
                Save Return Address in ERRSUB to detect
                nesting of ON ERROR GOSUB statements
             If Return to keyboard (not PgmRun)
                Push CNTADR on GOSUB stack (PSHUPD)
                Return type = 1
             If ON TIMER
                Shift Timer# to C(S)
                Return type = Timer# + 1
    0:       Push Return type/addr on stack(PSHGSB)
          Save Timer# incase of Error       (S-R0-0)
          Restore D0                        (R1)
          If Line#
             Find line# address                     (PFNDZL)
             If found
                Position to EOF before Line#
                Move Run address D1 -> C

    1:          If RESTORE statement                (sRESTR)
  RESTRX:          Set DATPTR to C
                   golong NXTSTM
```

```
                If GOTO from Keyboard
                   Update CNTADR @ stmt jump& (CNTCUR)
                   Compute Line# of stmt jump &
                   Update CURRL @ Line#
                   golong to next stmt in Stmt Buffer
                Set DO @ Run/execution address (C)
                Check if Trace Flow              (TRFCK-,TRTO+)
                Restore DO @ Execution address (R1)
                Set PRGM annunc, PgmRun flag     (SFGPGM)
                Goto to Run Loop
          If Label
             Move label into A                   (FILXQT)
             If Illegal Label or not in Current file
                Error Exit                       (ERROR)
             else
                Find label                       (FINDLB)
                If label not found ---> Error exit
             Move Label stmt start (Run address)  DO -> C
             goto 1;
 ERROR:   If line# or label not found
             If GOSUB
                Pop Return address off stack     (POPGSB)
             If ON ERROR statement               (sONERR)
                Clear ERRSUB address
                Clear ON ERROR address
             If ON TIMER                        (sONTMR)
                OFF appropriate Timer           (OFFTMR)
             If Trace mode --> Send CR/LF       (SNcrlf)
             Error Exit --> eSTMNF              (MFERR)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/04/83 | JP | Saving Timer# in scratch |
| 02/07/83 | JP | Add sXWORD status, PFNDLZ call |
| 03/08/83 | JP | Checking sEXTGS instead of sONTMR |
| 03/31/83 | JP | Remove UPDPC if External Entry |
| 04/29/83 | JP | If sXWORD, R3(S) = Return type |
| 05/27/83 | JP | If GOSUB from keyboard save CNTADR on GOSUB stack |
| 06/17/83 | JP | If GOTO from keyboard; set SUSP |
| 06/29/83 | JP | Check TRACE to before set PgmRun Set PgmRun ALWAYS |

## 22.6 USING - Interpret IMAGE String

Category: STEXEC     File: MB&IMG::MS

Name:(S) USING - Interpret IMAGE String

Purpose:
Parse IMAGE stmt for formatted input/output (DISP USING,
PRINT USING, ENTER USING, etc.)

Entry:
P = 0
D0= program PC (points to IMAGE string or line #)
D1 points to next item on stack.

Exit:
If error (IMAGE parse or USING xqt), to MFERR.
Otherwise, to NXTSTM, unless picked up by poll handler.

Calls: EXPEXC... Need I say more?

Uses: EXPEXC can use all CPU registers.

Stk lvls: 4 (all stack levels are lost, since the
IMAGE parse routines use the stack
for storage)

NOTE: All RSTK levels are lost. Never call USING expecting
any RSTK levels to be saved.

Detail:
Register usage:

D0= pointer into IMAGE string.
D1= pointer into BldIMG (expanded string where execution
code is built)

D(A)= address of available memory start.

R0(A) = D1 where backward search was started.
R0(9-5)= address to start execution.

R1(A) = stores D0.
R1(9-5)= length of IMAGE string (nibs).
R1(S) = counter for 2 complex numeric fields.

```
R2(A)  = counter for digits in front of radix

R3(A)  = Program Counter (D0 at entry or re-entry).
R3(9-5)= Address of start of IMAGE string.

Image tokens for building expanded IMAGE.
1) Tokens not identifying the end of a numeric field.
  1a) Tokens not used in backwards search.
  uSTRPT =#D0    String pointer
  uSTRPT =#D0    String pointer
  uMULT  =#D1    Multiplier
  uLOOPB =#D2    Loop on byte
  uLOOPS =#D3    Loop on string (12 nibs)
  uIMXCH =#D4    Strange execution character.

  1b) Tokens used in backwards search.
  uOPNNM =#D8    Open loop without multiplier
  uJMP{} =#D9    Jump over paren loop ptr (9 nibs)
  uJMPst =#DA    Jump over string pointer (14 nibs)
  uJMPdl =#DB    Jump over unfilled delimiter (8nibs)
  uIMbck =#DC    Poll for backward search handler
  uIMsta =#DE    IMAGE string start (|Dx| - see IMentr)
  uOPNM- =#DF    Open loop with mult, decremented
  uOPNUM =#E0    Open loop with mult  (ends in 0!)

****************************************************************
    EndNum =#E6      Any value >= this identifies the    *
                     end of a numeric field (used        *
                     in execution).                      *
****************************************************************

  2) Tokens identifying the end of a numeric field.
   2a) Tokens not used in backwards search.
   uCPLXC =#EE      Complex field closed
   uLOOPP =#EF      Loop on parentheses (variable #bytes)
   uIMend =#F0      IMAGE string end

   2b) Tokens used in backwards search.
   uRESTP =#F1      Restart parse
   uDELIM =#F4      Delimiter
       Tokens delimiting an output/input field.
   uHKB^  =#F6      H,K,B or ^ field
   uALit  =#F7      "A" literal field
   uNUMNn =#F8      Numeric, no float chars, no sign*
   uNUMNs =#F9      Numeric, no float chars, w/sign*
   uNUMFn =#FA      Numeric, w/float chars, no sign*
   uNUMFs =#FB      Numeric, w/float chars, w/sign*
   uNUMEn =#FC      Numeric, w/Exponent, no sign*
   uNUMEs =#FD      Numeric, w/Exponent, w/sign*

*Note: these numeric delimiters have values that
```

determine the status bit setting in USING execute.

Status bits

```
These status bits must be preserved during execution!
sMULT  =8        Multiplier pending.
sSIGN  =9        Sign already specified.
sFOUND =10       Output field found (at least one).
sRDX   =11       Radix already specified.

These status bits can be changed during execution.
(status bits 0,1,2 are used for numeric flags in xqt)
sXQT   =0        Start executing.
sC/P   =1        C/P pending.
sCntg  =2        Counting digits.
sInit  =3        Field already initialized.
InhEOL =4        Same as SB&IO !!! (always=0)
sSTOP  =5        Stop backward search.
sSpecl =6        Special handling (used in xqtn)
sCplxP =7        Complex field pending.
```

Bits for character masks used in parsing (CkLoop)

```
X-chr  =2^15     X: "blank"
D-chr  =2^14     D: digit
A-chr  =2^13     A: string char
Pt-chr =2^12     Decimal point
Dblqt  =2^11     Dbl quote: literal delim
Sglqt  =2^10     Single quote: literal delim
S-chr  =2^9      S: sign
M-chr  =2^8      M: sign
Z-chr  =2^7      Z: digit
E-chr  =2^6      E: exponent
C-chr  =2^5      C: separator
astrsk =2^4      *: digit
Z1-chr =2^3      Z: unit's digit A
P-chr  =2^2      P: separator
R-chr  =2^1      R: radix
Nf     =2^0      Nxtfld flag: return to Nxtf15.
ed     =(X-chr)+(Dblqt)+(Sglqt)   Edit chars
CP     =(C-chr)+(P-chr)           Separators
SM     =(S-chr)+(M-chr)           Sign chars
Rx     =(Pt-chr)+(R-chr)          Radix chars
edSMRx =(ed)+(SM)+(Rx)
CPE    =(CP)+(E-chr)
```

Algorithm:
 1:Statement set-up:
   If IMAGE is referred to by line no.,

establish program scope (in case keyboard xqt)
point D1 to line# (PFINDL)
skip over any line labels, find start of IMAGE string
calculate IMAGE string length
write uIMend token ("end of IMAGE string") to AvMemEnd
move IMAGE string to AvMemEnd
goto 2

If IMAGE is referred to by a string expression,
write uIMend token to AvMemEnd,
call EXPEXC (EXPR) to put string on stack at AvMemEnd
reverse string so it's in "normal" direction (REVPOP)
store D0(=PC) and D1(=start of IMAGE string) in R3.

2:IMAGE parse:
Follow the parse tree laid out in individual parse routines.

History:

| Date | Programmer | Modification |
|---|---|---|
| 08/10/82 | MB | Started writing code |
| 11/10/82 | MB | Finished writing code |
| 01/14/83 | MB | Updated documentation |

## 22.7 BEEP - BEEP Keyboard Execute

Category: STEXEC File: MN&BP::MS

Name:(S) BEEP - BEEP Keyboard Execute

Purpose:
BEEP, BEEP ON and BEEP OFF commands from BASIC.

Entry:
Jumped on BEEP token.

Exit:
If normal exit, NXTSTM.
eDATTY if provided complex argument(s).

Calls: BEEP: EXPEXC, POP1N, SFLAG?, BP.
BEEP ON: SFLAGC.
BEEP OFF: SFLAGS.

Detail:

BEEP ON
BEEP OFF
BEEP [ frequency [ , duration ] ]

Algorithm:

If PC points at ON token, clear BEEP disable flag.
If PC points at OFF token, set BEEP disable flag.
Else call EXPEXC;
  If parameters not supplied, use default frequency
  of 500 hz and default duration of 0.25 sec.
  Call BP to perform beep.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 05/20/82 | NM | Added documentation |

## 22.8 PRINT* - PRINT class statement execution

Category: STEXEC File: SB&IO::MS

Name:(S) PRINT* - PRINT class statement execution

Purpose:

Implements PRINT class statement execution. This includes DISP and PRINT.

Entry:

```
P      =  0
C(0) = PRINT class statement class number
        0 --> DISP
        1 --> PRINT
        2 --> OUTPUT
        3 --> PLOT
```

Exit:
Exits through NXTSTM

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/01/83 | B.S. | Added documentation |

## 22.9 PART3 - Finishes up a PRINT class statement

Category: STEXEC File: SB&IO::MS

Name:(S) PART3 - Finishes up a PRINT class statement

Purpose:
This is the 3rd part of PRINT class statements. It calls the appropriate routine to finish up the current line.

Entry:
P = 0
STMTRO set up by CKINFO

Exit:
Exits through NXTSTM

Calls: xPART3

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 11/09/83 | B.S. | Added documentation |

## 22.10 ZERBUF - Looks Like a Zero Length Buffer

Category: STEXEC    File: SB&IO::MS

Name:(S) ZERBUF - Looks Like a Zero Length Buffer

Purpose:
This looks like a zero length buffer.

Entry:
Do not enter

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | B.S. | Added documentation |

## 22.11 CREATE - Statement to Create Data File

Category: STEXEC    File: SC&FIL::MS

Name:(S) CREATE - Statement to Create Data File

Purpose:
The CREATE statement creates files of type DATA, TEXT, or SDATA. The syntax is:

CREATE <file type> <file spec> , <size> , <# recs>

Entry:

P = 0
D0 @ 4-nib file type in tokenized CREATE statement. (The file type is immediately followed by file specification)

Exit:
    P = 0
    To NXTSTM if successful
    To BSERR if error

Calls: FSPECx,SVFSP+,SNAPSV,EXTCHK,FINDF,SNAPRS,D0=PCA,
       SVFTYP,CRTF-

Uses.......
 Inclusive: A-D,R0-R4,D0,D1,S11-S0,Statement and Function
            scratch RAM, SCRTCH ram, SNAPBF

Stk lvls: 7

History:

| Date | Programmer | Modification |
|---|---|---|
| | SC | Designed and coded |
| 11/18/83 | FH | Added documentation |

## 22.12 CALL - Sub-program call execution

Category: STEXEC File: SC&SUB::MS

Name:(S) CALL - Sub-program call execution

Purpose: Call a sub-program

Enry: D0 pts past the tCALL token

Exit:
    To NXTSTM if successful
    To BSERR if error

Calls: I/OALL, GETCH-, FDCHN, EXPEXC, DEST, NEWVAR, SCHSUB
       LNSKP-, TRFLCK, TRCLIN, TRTOEN, I/OFND, EXPCHN, FNDMK-
       POPCHN, CR-VAR, POLL, STRASN, MOVEMD, FSPECx, FINDF
       SFLAGC, SFLAGS, SFLAG?, GETSTC, PRSCOO, CHKSPC

Uses: Everything

Stk lvls: All

Detail:

1. Search the subprogram and save the name on stack.
2. Start process the actuall parameters:
 a. Go down the parameters list, call expression to get every parameters.
 b. Save the value or the address of each parameter on the stack. Put a cap on top of each parameter to indicate it is a vaule or an address. (Parse routine already figured out each parameter is passed by value or by reference).
 c. If find an "#" sign preceding an expression, it must be a channel number. Then make sure the channel is open, also put a cap to indicate this is a channel number.
 d. Call the routine DEST right after returning from the expression execution routine. If the parameter is a non-existent variable, call the routine NEWVAR to create the variable. THEN collapse the stack(except the subprogram name), process the actual parameters all over again starting from the beginning. The reason for starting from the beginning is that some of the references that already been processed may need to be adjusted due to the creation of new variable. In order to save code, I choose to re-evaluate the all the expression rather than only to adjust those references.

3. Save the calling environment on the stack(on top of the actual parameters information).
(lowest address):

```
0004F  (5 nibs):  ID & length
A      (1)     :  Update pointers count
               :  Following 10 pointers are absolute
CURRST (5)         addresses, they will be adjusted
PRGMST (5)         when memory moved.
PRGMEN (5)
CURREN (5)
PCADDR (5)
CNTADR (5)
ERRSUB (5)
ERRADR (5)
ONINTR (5)
DATPTR (5)

Offset to previous FORSTK (5)
           //      GSBSTK (5)
           //      ACTIVE (5)
           //      CALSTK (5)
```

4. A LEX file can save its local environment on the call stack too. At this point, a poll(pCALSV) will be issued. An LEX file when answering to this poll can put a save block on top of the current stack pointer(pointed by D1). The format of the save block is as follow :

| nibs | meaning |
|---|---|
| 1-2 | LEX file ID. |
| 3-5 | Save block length(exclude the first 5 nibs). |
| 6 | Number of addresses follow that need to be adjusted when memory moved. |
| 7-11 | First address if any. |
| ..... | |
| ..... | to end of the block. |

5. Search for the subprogram.

6. Set CALSTK, ACTIVE, GSBSTK, FORSTK to the current stack pointer(MTHSTK)

7. Clear the variable chain head table

8. Put a level mark in the channel number assign buffer.

9. Process the formal parameters:
   a. If the parameter is a channel, open the channel.
   b. Call expression execution to get each variable and call the routine DEST right after that. Then call the routine CR-VAR to create the dope vector of each variable.
   c. Dig out the actual parameter from the stack one at a time and compare its type with the corresponding formal parameter.
   d. Assign value or indirect address to the formal parameter.

10. Pull all the actual parameter information from stack and adjust all the offset values in the call save block.

11. Clear ERRSUB, ERRADR, ONINTR, DATPTR

12. Execute the subprogram.

## 22.13 CALBIN - Binary program call BASIC subprogram

Category: STEXEC File: SC&SUB::MS

Name:(S) CALBIN - Binary program call BASIC subprogram

Purpose: To allow a binary program to call a BASIC subprogram.

Entry: This GOSBVL has to precede right before the CALL statement. The binary file has to construct the CALL statement exactly as it is in a BASIC file. The first two nibs are the statement length and the last two nibs are the EOL.

Exit: The execution of the binary program will be resumed after CALL statement.

Uses: Everything

Stk lvls: Only one RSTK will be saved, the one calls CALBIN.

Note: When CALBIN is called, the PCADDR will be set to @ the line length of the CALL statement.
When ENDSUB is executed, if it is returning to binary code, the PCADDR will be set to @ the end of the CALL statement.

## 22.14 ENDSUB - ENDSUB execution

Category: STEXEC File: SC&SUB::MS

Name:(S) ENDSUB - ENDSUB execution

Purpose: End a subprogram, restore the calling program environment.

Entry: Don't care

Exit: Exit to NXTSTM

Calls: STMBUF, TRFLCK, TRCLIN, TRTOEN, POPSTK, LINSKP
SCOPCK, CLPSTK, CLOSEA, KBRTCK, SFLAGC, SFLAGS

## 22.15 CAT - Executes CAT Command

Category: STEXEC File: SG&SYS::MS

Name: CAT - Executes CAT Command
Name: CAT100 - Buffer of Nonreadable Chars to Display
Name:(S) CATEDT - Display CATalog Info on the Current File

Purpose: CAT entry point executes CAT Statement

CAT100 sends a buffer of nonreadable characters to the display. It turns off the delay and the cursor. It assumes the buffer is pointed to by AVMEMS.

CATEDT displays the catalog for current file.

Entry: 2 ENTRY POINTS:
1) CAT - Execution of CAT command. Expect D0 is past tCAT
2) CATEDT - Displays CAT info on current file
3) CAT100 - Buffer pointed to by AVMEMS

Exit: via NXTSTM

Calls: FINDA, FINDF, BF2DSP, FSPECx, POLL, NOSCRL, RPTKY, SCRLLR, POPBUF, EDIT80, ROMCHK, ROMFND, WSR0-3, EOFLCH, tKYSck, D1=CRS, DSPDLY, EOLXC*, C=MAIN, CAT95, ROMF-1

Uses: A-D, D1, D0, R0-R3, STMTR0 (All 16 nibbles), S0
+ all of function scratch, S0-S11 - EXPEXC

Detail: CAT [file name][:<dev id>]|ALL|CARD|keys

```
Stack lvls: CAT    - 7
            CATEDT - 6
            CA1100 - 5
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/28/82 | S.W. | Added documentation |
| 12/07/82 | S.W. | All keys popped out of buffer |

```
+---------------------------------------------------+------------------+
|                                                   |                  |
|   STPARS  -  Statement Parse                      |  CHAPTER  23     |
|                                                   |                  |
+---------------------------------------------------+------------------+
```

## 23.1 GOTOp - GOTO Statement Parse

Category: STPARS    File: JP&PR1::MS

```
Name:(S) GOTOp   -  GOTO Statement Parse
Name:(S) GOSUBp  -  GOSUB Statement Parse

Purpose:
     Parse GOTO | GOSUB statement

Entry:
     D1 past GOTO | GOSUB token

Exit:
     Carry Clear - If lineno | label is output

      else error exit to PARERR:
               Illegal first character: Syntax Error

Calls:      LBLINP

Uses.......
 Inclusive: A-C,D(S), S0-S3,S7,S9-S11, R0,R1,R3, P, D0,D1

Stk lvls:   6

Detail:

GOTOp:
GOSUBp:  Parse lineno | label   (LBLINP)
         If carry set --> Error exit - "Syntax"
            else      --> RTNCC

History:

   Date     Programmer              Modification
 --------   ----------   ---------------------------------
 07/08/82   JP           Modified documentation
```

## 23.2 RESTRP - RESTORE Statement Parse

Category: STPARS File: JP&PR1::MS

Name: RESTRP - RESTORE Statement Parse
Name:(S) FIXP - FIX and WAIT Statement Parse

Purpose:
RESTRP parses RESTORE statement

FIXP parses FIX and WAIT statements. It also parses a single numeric expression.

Entry:
D(A) = (AVMEME)
D0 points into the output buffer
RESTRP entry:
 D1 past RESTORE keyword
 D0 past RESTORE token
FIXP entry:
 D1 points at alleged numeric expression

Exit:
RESTRP entry:
Legal statement syntax =>
 Return with carry clear
 Statement parsed and tokenized
 D1 past legally parsed statement
 D0 past token stream for RESTORE statement
 P=0
Else take error exit

FIXP entry:
Valid numeric expression found =>
 Return with carry clear
 Tokenized expression written to output buffer
 D0 points past token stream
 D1 points immediately past the expression
Else take error exit

Calls: LBLINP,PILP+,WRDSCN, OUT1TK, RESPTR, NUMCK

```
            D1C=R3

Uses........
 Exclusive: R3,S8
 Inclusive: R3,S8,S0-S3,S7,SA11,A-C,D(S),D0,D1,R0-R3,

Stk lvls:   6

Detail:
     RESTORE [ <lineno> | <label> ]
     RESTORE [# <num expr> [, <num expr> ] ]

Algorithm:

     Parse for lineno or label
     If lineno | label not found
        If channel # not found
           Return to main line parse to check for EOL
        else
           If comma follows Channel #
              Parse for <numeric expression>
     else
        RTNCC

History:
```

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | JP | Modified documentation |
| 10/20/82 | S.W. | No more RESTORE #<num expr>, END |

## 23.3 BEEPP - BEEP Statement Parse

Category: S1PARS    File: JP&PR1::MS

```
Name:    BEEPP   -  BEEP Statement Parse
Name:(S) DELAYp  -  DELAY and WINDOW Statement Parse

Purpose:
     Parses BEEP, WINDOW and DELAY statements
```

Entry:
D1 past BEEP, WINDOW, or DELAY keyword
D0 past tBEEP, tWINDW, or tDELAY
D(A) = (AVMEME)

Exit:
Return with carry clear =>
Accepted statement

Else error exit to PARERR

Calls: NUMCK, COMCK1, RESPTR, OUT1TK, EOLCK

Uses.......
Exclusive: A,C,D1,D0
Inclusive: A-C,D(15-5),R0-R2,S0-S3,S7,S11,FUNCD0
D1,D0

Stk lvls: 5

Detail:
```
BEEP [ ON | OFF ]
BEEP [ <frequency> [ , <duration> ]

DELAY <delayt> [,<scrollt>]
WINDOW <start> [,<end>]
```

frequency, duration, delayt, scrollt, start, and end are all specified using numeric expressions.

Algorithm:
```
If Next Token = End of Line Terminator
   Restore Pointer
   Return CC
If Next Token = ON | OFF
   Output Token
   Return CC
else
   Restore Input pointer
   Verify first parameter
   If next token = comma
      Verify second parameter
   else
      Go Restore pointer & Return
RTNCC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/08/82 | JP | Modified documentation |

```
08/18/82   S.W.        Combined WINDOW and DELAY parse with
                        BEEP parse
11/01/83   S.W.        Modified documentation header.
```

## 23.4 ONP - ON Statement Parse

Category: STPARS     File: JP&PR1::MS

```
Name:     ONP    - ON Statement Parse
Name:(S)  ONP40  - GOTO,GOSUB,RESTORE in middle of stmt Parse

Purpose:
     Parse ON statement

     Possible syntax is:
       ON ERROR (GOTO | GOSUB) (lineno | label)
       ON TIMER  # <timer no>,  <#secs> ( GOTO | GOSUB )
                ( <lineno> | <label> )
       ON <exp> GOTO    <lineno>|<label> [, <lineno>|<label>]
             "  GOSUB           "
             "  RESTORE         "

Entry:
     D1 past ON keyword
     D0 past tON in output buffer
     D(A) = (AVMEME)

Exit:
     If accepted
       Return with carry clear
       P=0
       D1 past valid statement
       D0 past tokenized statement in output buffer
       S8=1 => ON ERROR | ON TIMER statement

     If unaccepted
       Error exit through PARERR

Calls:      NUMCK,LBLINP,COMCK+,RESPTR,WRDSCN,NTOKEN
            #CK, NUMC++, RESPTR, CONCKO
```

```
Uses.......
 Exclusive: A,C,S8
 Inclusive: A,C,S8,B,D(15-5),S0-S3,S7,S11,R0-R3,FUNCD0

Stk lvls:   6

Detail:

     ON <exp> ...         allowed from keyboard
     ON TIMER | ERROR     not allowed from keyboard

Algorithm:

     If Next Token = ERROR
        If Keyboard execute --> Error exit
        Set ON ERROR statement flag
        goto 1;
     If Next Token = TIMER
        If Keyboard execute --> Error exit
        Set ON TIMER statement flag
        If next char # "#"
           Error Exit with No restore of input pointer
        Skip "#" and
        Verify <timer no> expression  (NUMC++)
        If A(B) # Comma  (F1)
           then Error ---> Syntax
        Output Comma token  (COMCK0)
        Verify <# secs> expression  (NUMCK)
1:   If Next Token # GOSUB | GOTO
        If ON-ERROR| TIMER stmt    ---> Error Exit
        If Token # RESTORE     ---> Error Exit
2:   Check for label | lineno  (LBLINP)
     If not label | lineno ---> Error Exit
     If ON-ERROR statement ---> RTNCC
     Check for comma and output (COMCK0)
       Continue Label/Lineno parse  (goto 2)
     else goto RESPTR (Position before non-comma & RTN)

History:

  Date       Programmer              Modification
 --------    ----------    ------------------------------------
 07/08/82    JP            Modified documentation
 11/01/83    S.W.          Updated documentation header
```

## 23.5 READP - READ, READ# Statement Parse

Category: STPARS File: JP&PR2::MS

```
Name:     READP   -  READ, READ# Statement Parse
Name:     INPUTP  -  INPUT Statement Parse
Name:     LINPTP  -  LINPUT Statement Parse
Name:     DSTp    -  Single Destination Variable Parse
Name:(S)  READP5  -  Destination Variable List Parse

Purpose:  Parses READ, READ#, INPUT, LINPUT statements.

          DSTp entry expects a 'destination' variable, ie
          one that is suitable for storing a value.

          READP5 entry will parse a list of destination
          variables, delimited by commas.  Depending on
          status bits S8 and S9 on entry, it allows or
          disallows dummy arrays, allows a list of any
          number of destination variables, or demands
          that the first variable in the list is a
          string destination and then returns to leave
          the rest of the parse (if any) to the caller.

Entry:    D(A) = (AVMEME)
          5 entry points:
          1) LINPTP - D1 past LINPUT
                      D0 past tLINPT
          2) INPUTP - S9=0
                      D1 past INPUT
                      D0 past tINPUT
          3) READP  - S8=0, S9=0
                      D1 past READ
                      D0 past tREAD
          4) READP5 - S8=0 iff Dummy arrays are valid
                      S9=1 iff single string var parse
          5) DSTp   - D1 pts to alleged destination var.

Exit:     Valid parse =>
            P=0
            LINPTP,INPUTP,READP entry:
             D1 past syntactically correct stmt
             D0 past tokenized statement
             Return with carry clear

            READP5 entry:
```

```
                D1 past the parsed variable or var list
                D0 past tokenized destination variable(s)
                Return with carry clear
                If S9=1 on entry
                 Single string destination variable parsed
                 D1 past the string variable
                 D0 past the tokenized string variable

               DSTp entry:
                D1 past destination variable
                D0 past tokenized destination variable
                Carry set on return iff dummy array

             Invalid parse =>
               LINPTP,INPUTP,READP entry:
                Error exit to PARERR

               READP entry:
                Error exit to PARERR
                If S8=0,S9=0 on entry
                 Something in list was not a destination
                 variable, or a delimiter was missing
                If S8=1,S9=0 on entry
                 Something in list was either not a
                 destination variable, or was a dummy
                 array, or a delimiter was missing
                If S8=0,S9=1
                 First item in list was not a string
                 destination variable.
                If S8=1,S9=1
                 First item in list was either a dummy
                 array or was not a string destination
                 variable.

               DSTp entry:
                 Input either was an invalid expression
                 or was inappropriate as a destination.

  Calls:     OUT1TK, NTOKEN, DSTp, COMCK, PILP, WRDSCN
             DATACK, STRGCK, COMCK1, OUT1TK, EXPPR+

  Uses:      A-C,D(15-5), D1,D0, R0-R2, S0-S3,S7-S9,S11
             FUNCD0, P

  Detail:    Doesn't allow for INPUT/READ/LINPUT without at
             least one variable in the list
             Allows for READ#, but not INPUT#.
             READ# compiled as:
             # num expr [tCOMMA <num expr>] [SEMIC <var list>]
               Even if there's no record# specified, there must
               be a variable list.
```

INPUTP and LINPTP allow an optional prompt
and initial string for default values

Tokenized destination variables in READ, READ#,
INPUT and LINPUT are delimited by tCOMMA.

Stk lvls: 5

History:

| Date | Programmer | Modifications |
|---|---|---|
| 12/06/82 | S.W. | READ,READ# allows dummy arrays |
| 03/11/83 | S.W. | Tokenize INPUT with prompt with preceding zero byte |
| 05/18/83 | S.W. | Calls new subroutine: DSTp |

## 23.6 DECP - Parse of Variable Declaration Statements

Category: STPARS File: JP&PR2::MS

Name:(S) DECP - Parse of Variable Declaration Statements

Purpose: Parses REAL, SHORT, INTEGER statements

Entry: D1 past REAL, SHORT, or INTEGER keywords
D0 past tREAL, tSHORT, or tINTEG
D(A) = (AVMEME)

Exit: If valid statement syntax:
via RESPTR (Carry clear)
D1 past syntactically correct statement
D0 past tokenized statement in output buffer

If error in syntax:
Exit to PARERR

Calls: COMCK0, ARRYCK, VARP

Uses: A-C,D(15-5), D0,D1, S0-S3,S11, R0,R1, FUNCD0

Stk lvls: 6

History:

| Date | Programmer | Modifications |
|---|---|---|
| 03/06/83 | S.W. | New documentation header added |
| 05/10/83 | S.W. | Added call to COMCKO |

## 23.7 PRTP - PRINT Statement Parse

Category: STPARS File: JP&PR3::MS

```
Name:     PRTP    - PRINT Statement Parse
Name:(S) DISPP   - DISP Statement Parse
Name:     DSPPO2  - Implied DISP Statement Parse
Name:(S) USINGp  - USING statement Parse

Purpose:  PRTP parses the PRINT statement.

          DISPP parses the DISP statement.
          It is also used to parse an implied DISP
          when implied LET parse has failed.

          DSPPO2 parses implied DISP. The distinction
          between DSPPO2 and DISPP is that with DSPPO2
          entry, parse errors result in a return to the
          caller; this entry is used on an alleged
          implied DISP that cannot be an implied LET,
          ie one that doesn't start with a variable or
          user-defined function name.

          USINGp parses USING part of PRINT USING stmt
          This entry point used by HPIL for ENTER USING

Entry:    D(A) = (AVMEME)
          D1 points at input stream
          D0 points into output buffer
          3 entry points:
          1) PRTP   - D1 past PRINT keyword
                      D0 past tPRINT
```

```
        2) DISPP  - D0 is past tDISP.
                    Either D1 is past the DISP keyword
                     OR
                    D1 is at the beginning of a statement
                    that failed implied LET parse and
        3) DSPP02 - D1 at alleged expression list that
                     doesn't start with a variable or
                     user-defined function name.
                    tDISP has been output and D0 points
                    past it.
                    S8=1
                    If needed, D1/D0 have been saved
                    somewhere so that in case of error
                    they can be recovered.

        4) USINGp - D1 at USING keyword

Exit:
        Carry clear =>
         P=0
         D1 past syntactically correct statement
         D0 past tokenized statement in output buffer

        Carry set (DSPP02 entry only) =>
         Not a valid implied DISP statement

        Else error exit of some kind:
         To PARERR (PRTP, DISPP entry only)
         or to MEMERR (possible for all entry points)

Calls:  EXPPAR, NTOKEN, OUT1TK, NUMCK, PILP, COMCK, WRDSCN,
        LBLINP, EOLCKR, RESPTR, R3=D10, D1C=R3

Uses:   A-C,D(15-5),S0-S3,S7,S8,S9,S11,R0-R3,FUNCD0

NOTE: No routines called may use S8 (except PILP), S9
      No routines below DISPP entry point may use R3 -
       See LNP00 utility

Detail: The PRINT statement is tokenized identical to the
        DISP statement, except for tPRINT instead of tDISP.
        PRINT# is tokenized very differently from PRINT.

        Compiled DISP statement looks like:
       tDISP [tUSING <tLINE# line#> | <string expr>]
         [tSEMIC <display list>]

        Compiled PRINT# statement looks like:
  tPRINT #<channel no.>[tCOMMA <rec no.>]tSEMIC<exprs>
  tPRINT #<channel no.> tCOMMA <record no.>
```

Stk lvls: 5 (if PRINT# then 6)

History:

| Date | Programmer | Modifications |
| --- | --- | --- |
| 10/21/82 | S.W. | Eliminated capability for DISP USING <lbl> |
| 04/29/83 | S.W. | Disallow TAB in PRINT/DISP USING |
| 05/02/83 | S.W. | Create USING subroutine for use by PRINT/DISP, ENTER/OUTPUT |
| 05/11/83 | S.W. | Replaced 1 call to COMCK1 w/COMCK+ |

## 23.8 POKEP - POKE Statement Parse

Category: STPARS File: JP&PR3::MS

```
Name:     POKEP  -  POKE Statement Parse
Name:(S)  STRNGP -  Parse of a Mandatory String Expression

Purpose:  POKEP parses POKE statement.

          STRNGP parses a mandatory string expression

Entry:    D(A) = (AVMEME)
          D1 points to input stream
          D0 points into output buffer
          2 entry points:
          1) POKEP  - D1 past POKE keyword
                      D0 past tPOKE
          2) STRNGP - D1 pts to alleged string expr

Exit:
          Valid parse =>
           Return with carry clear
           P=0
           POKEP entry:
            D1 points past syntactically correct stmt.
            POKE tokenization written to ouput buffer.
            D0 points past POKE tokenization.
           STRNGP entry:
```

D1 points past string expression.
String expr tokenization in output buffer.
D0 points past string expr tokenization.

Else error exit

Calls: OUT1TK, STRGCK, COMCK+

Uses: A-C,D(15-5),D0,D1,R0,R1,R3,S0-S3,S7,S11,FUNCD0

Stk lvls: 5

Detail: POKE <string expression>,<string expression>

History:

| Date | Programmer | Modifications |
|---|---|---|
| 05/11/83 | S.W. | Replaced call to COMCK1 w/COMCK+ |

## 23.9 CALLP - CALL Statement Parse

Category: STPARS File: JP&PR3::MS

Name:(S) CALLP - CALL Statement Parse

Purpose: Parses CALL Statement

Entry: D(A) = (AVMEME)
D1 past CALL keyword in input stream
D0 past tCALL in output buffer

Exit: Valid Statement Parse =>
P=0
Return with cary clear
D1 past syntactically correct CALL statement
CALL stmt tokenization written to output buffer.
D0 points past statement tokenization.

Else error exit

Calls: SUBNMP,OUTBYT,EXPPAR,NUMCK,CNV2UC,FSPECp
CLRPRM,COMCK,EOLCK+,DATACK,NTOKEN,OUTVAR,SBNMPO
NUMCK3,NUMCK1,OBFSPp,CNV2UC,PRENCK,COMCK1,R3EXPP

Uses: A-C,D(15-5), S0-S3,S7,S9-S11, D1,D0, R0-R3,
FUNCD0

Stk lvls: 6

Detail: Compiles to:

```
tCALL <name>
 [tPRMST (parm E<0|1> parm E<0|1> ... ] tPRMEN
  [tIN <filespec>]

         where E0 (tCREF) indicates a pass by reference
           and E1 (tCVAL) indicates a call by value.
               parm:= <#num expr|variable|expression>

               tIN is actually tSEMIC
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/11/82 | S.W. | Output E1 (tCVAL) after chnl# |
| 11/11/82 | S.W. | Added code to trap out user-defined functions |
| 12/09/82 | S.W. | CALL w/o parms allowed from keybd |
| 02/11/83 | J.P. | Made REDPRM straight line code. |
| 05/03/83 | S.W. | Added call to #CK |
| 05/23/83 | S.W. | Channel# ALWAYS tokenized as pass by reference |
| 06/02/83 | S.W. | Don't allow user-defined functions in channel numbers |

## 23.10 ADDP - ADD Statement Parse

Category: STPARS File: JP&PR3::MS

Name:(S) ADDP - ADD Statement Parse

Purpose: Parses ADD and DROP Statements

Entry: D(A) = (AVMEME)
D1 points at input stream past ADD or DROP keyword
D0 points into output buffer past tADD or tDROP

Exit: Valid statement parse =>
Return with carry clear
P=0
D1 points past syntactically correct statement
Tokenized statement written to output buffer
D0 points past statement tokenization

Else error exit

Calls: NUMS (NUMCK)

Uses: A-C,D(15-5),R0,R1,R3,D0,D1,S0-S3,S7,S11,FUNCD0

Stk lvls: 6

Detail: Syntax is:
ADD | DROP [num expr [,num expr...]]
Tokenization is:
tADD | tDROP [num expr [num expr...]]
(tCOMMA is NOT output between expressions)

History:

| Date | Programmer | Modifications |
|---|---|---|
| 02/08/83 | S.W. | No longer limits to 15 expr |
| 05/12/83 | S.W. | Use SFLAG/CFLAG parse |

```
+----------------------------------------+------------------+
|                                        |                  |
|  SYSTEM - System Level Major Entry Points |  CHAPTER 24   |
|                                        |                  |
+----------------------------------------+------------------+
```

## 24.1 CNFLCT - Report "Data Type" Error.

Category: SYSTEM    File: AB&FCN::MS

Name:(S) CNFLCT - Report "Data Type" Error.

Purpose:
To do a GOVLNG =RDATTY.

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | MB | Documentation |

## 24.2 ARGERR - Report "Invalid Arg" Error.

Category: SYSTEM    File: AB&FCN::MS

Name:(S) ARGERR - Report "Invalid Arg" Error.

Purpose:
To report "Invalid Arg" as an execution error.

Entry:
S13=0 if not a running program (i.e., keyboard
execution error)
S13=1 if a running program.

No other necessary conditions.

Exit:
Exits to BASIC main loop (ERRRTN)

Calls: MFERR

Uses....... Exits to main loop, can use anything

Stk lvls: Exits to main loop, can use all

NOTE:
ARGERR sets P=0 to select an execution error:
-- not a parse error
-- store ERRN (and ERRL, if S13=1)
-- display "ERR:" (or "ERR L<N>:") prefix
-- exit to BASIC main loop

Detail:

```
 =ARGERR P=     0
         LC(2)  =eIVARG
         GOLONG =MFERR
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | MB | Documentation |

## 24.3 NORDIM - Report "Var Context" Error

Category: SYSTEM File: AB&REG::MS

Name:(S) NORDIM - Report "Var Context" Error

Purpose:
Report "Var Context" as an execution error.

Entry:
P = 0
S13=0 if not a running program (i.e., keyboard

```
          execution error)
       S13=1 if running program

   Exit:
       Exits to BASIC main loopo (ERRRTN)

   Calls:      MFERR

   Uses....... BASIC main loop can use anything

   Stk lvls:   BASIC main loop can use all

   NOTE:
       Setting P=0 selects the following error options:
         -- not a parse error
         -- store ERRN (and ERRL if S13=1)
         -- display "ERR:" (or "ERR L<N>:")

   Detail:
      =NORDIM LC(2)  =eVCNTX
              GOLONG =MFER

   History:

     Date      Programmer              Modification
   --------    ----------    -----------------------------------
   11/09/83    MB            Documentation
```

## 24.4 BSCEXC - BASIC Stmt/Pgm Execution: Keyboard Exec

```
       Category:  SYSTEM     File:  JP&SYS::MS


   Name:(S) BSCEXC  -  BASIC Stmt/Pgm Execution: Keyboard Exec
   Name:(S) BSCEX2  -  BASIC Stmt/Pgm Execution: Program Exec
   Name:(S) BSCEXT  -  BASIC Stmt/Pgm Exec: Reentry into BASIC loop
   Name:(S) RUNRT1  -  Stmt reentry to BASIC loop;sERROR,sENDx clre??
   Name:(S) RUNRTN  -  Stmt reentry to BASIC loop; sERROR cleared
   Name:(S) ERRRTN  -  Error Exit reentry to BASIC loop

   Purpose:
       BASIC interpreter loop for program/statement execution
```

```
      Complete execution of RUN | CONT command

   Entry:
      BSCEXC: NoCont flag clear
                Keyboard Execute entry: PgmRun = 0
      BSCEX2: If PgmRun = 1
                 Program to be executed
                 DO @ EOL of prior statement
                 (RUN,CONT,SST entry)
                 (If NoCont=1 then SST)
              If PgmRun = 0
                 Statement to be executed
                 DO @ Statement length byte of statement
              Polls on entering BASIC interpreter

      BSCXLP: LABEL entry if within Multi-statement line
                DO @ EOL or @ of next statement to execute

      BSCEXT:
              Return to Keyboard "Reentry"
              RUN/CALL Binary return from "ENDBIN"
              ENDALL from PURGE/MERGE current file
                Filetype is read
                If BASIC and Program running (S13=1)
                   DO @ Next stmt to execute
                   SUSP will occur
                   Exceptions are NOT checked
                sERROR=1  (S0) --> Error has occured
                If not an error, Flush print buffers
                Poll on exiting BASIC interpreter
                Clears flags, goto Main Loop

      RUNRT1: Statement reentry into BASIC loop
              sENDx, sERRORx cleared
      RUNRTN: Statement reentry into BASIC loop
              sERRORx cleared --- used by END stmt
      ERRRTN: Error exit reentry into BASIC loop
              Assemes sERROR set; sENDx clear

   Exit:
      Jump to individual execute routine for statement
      S0-S11 are cleared before jumping

      ALL statements MUST return through NXTSTM or
        directly to RUNRT1/RUNRTN with DO set properly

        CALL, END SUB, FN, GOTO, GOSUB jump to RUNRT1
        NXTSTM returns to RUNRTN
        Errors return to ERRRTN
        SST @ Program End returns to BSCEXT
        Binaries return to BSCEXT (from ENDBIN)
```

```
        RUNRT1:  Clears sENDx flag, indicating not END stmt
                 Clears sERROR
        RUNRTN:  Assumes sENDx flag set appropriately
                 Clears sERROR

      If continuing execution:

      Timers are serviced at the end of each stmt execute
        If a timer expires & is within current program scope
          The appropriate ON TIMER code is jumped to.
          Statement execution will return to RUNRTN

      Execution stops if:
        End of program reached | STOP/END statement in program
        End of line of calculator statement
        Don't Continue (NoCont) flag set from:
          PAUSE, ATTN, Error Message Routine
          END/STOP within Program
          End of Program reached
          SST
          END(SUB),END(DEF), RETURN from keyboard
        Error flag (sERROR) set from Error Message routine


Calls:      EXCADR,CK"ON",BASCHK,SFLGCP,FLUSHA,CNTCUR,CKSREQ,
            EOLSCN,USRSTA,GTMRA+,FPOLL,ALMSRV,SCOPCK,TRFCK-,
            UPDPC,RDCHD+

Uses.......
 Exclusive: A,C,D1,D0,S13,S14,PCADDR,CURRL,R0,S0,S1
 Inclusive: A-D,D1,D0,S13,S14,PCADDR,CURRL,R0,S0-S7,
            SCRTCH (32 nibs),f1PRGM,f1SUSP,ANNAD1-4,STMTD1

         PCADDR must not be used for anything else
         sENDx  = END/STOP  Statement       S1
                  NXTSTM explicitly clears
                  RUNRT1 explicitly clears
         sERROR = ERROR occured             S0
                  RUNRTN,RUNRT1 clear
                  MFERR/BSERR sets
         Except = Service Request           S12
         PgmRun = Running program           S13
         NoCont = Don't Continue Execution  S14
         Trace  = TRACE Mode                S15

Stk lvls:   >=4

Algorithm:

BSCEXC:  Clear No Continue of Program flag     (NoCont)
BSCEX2:  Place current D0 into R0
```

```
          Fast poll on entering BASIC interpreter(pBSCen)
          If not running                      (not PgmRun)
             go update PC address             (goto BSCX+)
  BSCXLP: Read & Move past EOL | @
          If EOL and not running
             go Read filetype then            (goto BSCEXT)
             go exit BASIC                    (goto BSCEX+)
          If @  (multi-statement line)
             go Update PC address             (goto BSCX+)
          If End of current program           (PRGMEN > D0)
             go execute END statement             (END10)
          Skip line#
   BSCX+: Update PCADDR @ stmt  length byte       (PCADDR)
          Save addr @ statement length byte       (PCADDR)
          Skip statement length byte
          Clear lower status                      (S0-S11)
          Read Begin BASIC token
          If not Begin BASIC token range          (BASICs)
             Call Assignment Execute              (ASNMNT)
             Skip to next statement               (NXTSTM)
          else
             Move past BASIC token
             Calculate Execution addr             (EXCADR)
             Jump to Execution routine

  Statement Execute Return: (from NXTSTM or directly)

  RUNRT1: Clear END execute flag                   (sENDx)
  RUNRTN: Clear ERROR flag                        (sERROR)
  ERRRTN: Collaspe Math Stack
          If ERROR                                (sERROR)
             Skip exception checking              (goto 6)
          If no exceptions                      (Except=0)
             If no hardware service request         (SREQ)
                If any pending alarm set          (PNDALM)
                   Save D0 on stack
                   go Process timers              (goto 3)
                go continue                       (goto 6)
          Save D0 on stack
          Check Service requests                  (CKSREQ)
          If no exceptions                      (Except=0)
             go Restore D0 and continue           (goto 5)
          Clear Exception Flag                    (Except)
          Fast Poll on Exception                  (pExcpt)
          Restore low status from DSPSTA          (USRSTA)
      3:  If ATTN Key hit                           (CKON)
             Set NoCont flag                         (S14)
          else
             If Program running
                Load mask to check Timer bits
                Read Pending Alarm field           (PNDALM)
```

```
    4:        If Timer expired      (Bit 0|1|2 of PNDALM)
                 Get Timer Address              (GTMRA+)
                 If non-zero Timer address
                    Verify address in prgm scope(SCOPCK)
                 If within scope             (Carry clear)
                    Clear timer bit in PNDALM
                    Set Except if anthr timer due(ALMSRV)
                    If TRACING                   (TRFCK-)
                       Update PCADDR @ next stmt to exec
                    C <-- ON TIMER a dress
                    Set ONTIMER statement flag (sONTMR)
                    Clr ONERROR statement flag (sONERR)
                    go process ON TIMER stmt   (ONTIMR)
              go Check if any other Timers off (goto 4)
    5: Restore D0
       Clear Error occured                     (sERROR)
    6: If Continue                           (not NoCont)
          go process next of statement         (BSCXLP)
       else                                    (NoCont)
BSCEXT:   Clear PRGM Annunciator               (SflgCp)
          Read Filetype                        (RDCHD+)
          If non-BASIC file                    (BASCHK)
             go exit BASIC                 (goto BSCEX+)
          If not running                   (not PgmRun)
             go exit BASIC                 (goto BSCEX+)
          else
             If not END/STOP execute            (sENDx)
                If ELSE
                   Skip to End of Line         (EOLSCN)
                Update Continue Address
                Set SUSP Annunc/Flag           (SFLAGS)
                Compute & Update current line (CNTCUR)
BSCEX+:
       If not an error                         (sERROR)
          Flush all buffers                    (FLUSHA)
       Fast Poll on Exiting BASIC interp       (pBSCex)
       Clear Don't Continue flag               (NoCont)
       golong MAIN Loop                        (MAINLP)
```

A note on CNTADR and CURRL:

When execution is not continued:
   The current Line# is updated
   If not an END/STOP statement and BASIC file
      The Continue Address is updated to the next statement

If the end of program scope (@ PRGMEN) | END/STOP
   CNTADR = 0
   Continue Address is NOT updated at end of BSC Loop
   Current Line is not touched
   This is normal program execution termination.

CONT will start execution at the start of the prog

If the end of program scope is NOT reached, but exec stops:
CNTADR = Current DO
CURRL = Line# of Continue address
Current line always points @ CNTADR statement

For Error Messages
CNTADR = Statement in error
CURRL = Line# of error

For ATTN Key
CNTADR = Next statement to execute
CURRL = Line# containing next statement to execute

For PAUSE:
CNTADR = Statement after PAUSE
CURRL = Line# containing statement after PAUSE

History:

| Date | Programmer | Modification |
|---|---|---|
| 02/04/83 | JP | Added ALMSRV call if Timer due |
| 03/07/83 | JP | Packed: added UPDCRL call |
| 03/08/83 | JP | Clear sEXTGS before ONTIMR jump |
| 03/28/83 | JP | If ERROR, skip exception check,sERROR |
| 03/28/83 | JP | If not error, flush buffers |
| 04/04/83 | JP | If tracing & ON TIMER update PCADDR |
| 04/04/83 | JP | Preserve S0-S11 during pExcept |
| 04/08/83 | JP | Zero Timer bit ONLY when servicing |
| 04/08/83 | JP | If no exceptions/SR check Timer bits |
| 04/21/83 | JP | Don't SUSP if non BASIC file |
| 04/25/83 | JP | Pass filetype in pBSCex poll ALWAYS |
| 04/25/83 | JP | Changed BSCEXT entry point |
| 05/18/83 | JP | Check Attn after pExcept (CK"ON") |
| 06/17/83 | JP | Update CURRL only if SUSPending |
| 06/17/83 | JP | CURRL points at CNTADR statement |

## 24.5 IMerr - Report "Invalid IMAGE" error

Category: SYSTEM File: MB&IMG::MS

Name:(S) IMerr - Report "Invalid IMAGE" error

Purpose:
To generate the error "Invalid IMAGE".

Entry:
No necassary conditions.

Exit:
Through MFERR.

Calls: MFERR

Uses: MFERR exits to BASIC main loop; may use anything

Stk lvls: MFERR exits to BASIC main loop; may use 7

Detail:

```
=IMerr  P=     0
        LC(2)  =eINVIM
        GOVLNG =MFERR
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 12/08/82 | MB | Documentation |

## 24.6 IVAERR - Report "Invalid Arg" error.

Category: SYSTEM File: PM&STA::MS

Name:(S) IVAERR - Report "Invalid Arg" error.

Purpose:
To do a GOLONG =ARGERR

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | MB | Documentation |

## 24.7 COLDST - Cold starts machine

Category: SYSTEM File: SB&DVR::MS

Name:(S) COLDST - Cold starts machine

Purpose:
Initializes all system RAM, IO Buffers, Pointers etc.

Entry:
None

Exit:
Exits to MAINLP

Calls: CONF,INITCL,DSPRST,WIPOUT,AUTCLR,BF2DSP,EDITWF,
I/OALL,FPOLL

Uses.......
Exclusive: Absolutely everything in the entire machine
except independent RAMs

NOTE:
This routine should be used with caution since it may
annoy the user.

Algorithm:

Enables interrupt system
Initialize CMOS test word

Initialize system RAM to zeroes
Reset display
Turn display on
Set display row drivers
Set display contrast nibble
Initialize DELAY parameters
Perform ColdStart configure
Create Statement Buffer
Initialize clock system
Check for low battery
Initialize flags and traps
Zero RAM between AVMEMS and RAMEND
Clear AUTO mode
Clear program running flag
Clear don't continue flag
Initialize IS-TBL table
Initialize PRINT and DISP position and width
Initialize ENDLINE string
Put Coldstart message in display
Create Workfile
Create file information buffer
Initialize random number seed
Perform coldstart fast poll

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/14/82 | B.S. | Updated documentation |

## 24.8 MAINLP - Main Loop

Category: SYSTEM File: SB&DVR::MS

Name:(S) MAINLP - Main Loop
Name:(S) MAIN05 - Main Loop
Name:(S) MAIN30 - Main Loop

Purpose:
These entry points implement the normal idle state where the cursor is blinking in the display.

```
Entry:
    MAINLP:  Almost nothing matters.  The system will
             check a few flags and clear a few.  Then...
    MAIN05:  Allows user to scroll displayed line if there
             is one then prompts for input.  Then...
    MAIN30:  Calls character editor to input a line until
             special key is hit then jumps to a routine
             to handle that key.

Exit:
    P      = 0
    Control is passed to one of LINEP,WAKEUP,ATTNTN,RUNK,
    CONTK,SST,CALC,PWROFF,CURTOP,CURBOT,CURSUj,CURSDj,
    CMDSTK,PWROFF,IEXKEY

Calls:      SFLAG?,SFLAGS,SFLAGC,FPOLL,AUTOCK,SCRLLR,BF2DPP
            COLLAP,CLCOLL,STMBCL,NOPRGM,I/ODAL,ATNCLR,
            CURSFR,TBLJMC

Algorithm:
    MAINLP:If f1TNOF or f1MKOF set then
             Go to PWROFF
           If CALC mode set then
             Go to CLCERR
           Fast Poll (pMNLP)                       (FPOLL )
           If in AUTO mode then
             Go to =AUTXQ7
    MAIN05:If CALC mode (f1CALC) is set then
             Go to =CLCERR
           Clear program annunciator & status bit  (NOPRGM)
           Set f1DORM
           If Don't Prompt flag (f1NOPR) is set then
             Go to MAIN30
           If scrolling needed (NEEDSC) then
             Allow user to scroll                  (SCRLLR)
           Send prompt string consisting of        (BF2DPP)
             Cursor off, prompt character(">"),
             Cursor on
    MAIN30:If Attn key has been pressed jump to
             clean up as necessary.                (ATTNTN)
           Clear Don't Continue flag (NoCont)
           Collapse math stack                     (COLLAP)
           Collapse AVMEMS,OUTBS,SYSEN to CLCSTK   (CLCOLL)
           Clear Don't Prompt flag (f1NOPR)
           Collapse statement buffer               (STMBCL)
           Delete Immediate Execute Key buffer (bIEXKY)
           Set "Dormant" flag (f1DORM)             (SFLAGS)
           Call Character Editor                   (CHEDIT)
           If Immediate Execute Key then
             Go to IEXKEY
           If not cursor up/down then
```

```
          Clear command stack flag (f1CMDS)       (SFLAGC)
         Clear "Dormant" flag (f1DORM)            (SFLAGC)
         Clear Attention Flag so HPIL won't abort(ATNCLR)
         Move cursor to far right of display      (CURSFR)
         Go to appropriate place to process key   (TBLJMC)
             Endline         (LINEP)
             Attention       (ATTNTN)
             RUN key         (RUNK)
             CONT key        (CONTK)
             SST key         (SST)
             Cursor Up       (CURSUj)
             Cursor Down     (CURSDj)
             Cursor Top      (CURSTj)
             Cursor Bottom   (CURSBj)
             G-Attention     (ATTNTN)
             CALC Mode key   (CALC)
             Off key         (PWROFF)
             Command Stack   (CMDSTK)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 01/05/83 | B.S. | Added documentation |

## 24.9 PWROFF - Power Off

Category: SYSTEM File: SB&DVR::MS

Name:(S) PWROFF - Power Off

Purpose:
Sends machine into deep sleep and waits for wakeup

Entry:

Exit:
Exits to LINEP+ if a command buffer needs processing
otherwise exits to MAINLP

Calls: DPS010(DSLEEP),SFLAGS,I/OFND

Algorithm:
    Set f1PWDN
    Call DPS010 to go to deep sleep
    If there is an external command buffer
        then jump to LINEP+ to process it
    If there is an STARTUP buffer
        then jump to LINEP+ to process it
    Jump to MAINLP

History:

| Date | Programmer | Modification |
|---|---|---|
| 07/15/82 | B.S. | Updated documentation |

## 24.10 RDATTY - Report "Data Type" error

Category: SYSTEM    File: SB&RD::MS

Name:(S) RDATTY - Report "Data Type" error

Purpose:
    To report "Data Type" as an execution error.

Entry:
    S13=0 if program not running (i.e., keyboard execution error)
    S13=1 if running program
    No other necessary conditions.

Exit:
    Exits to BASIC main loop (ERRRTN)

Calls: MFERR

Uses....... BASIC main loop can use anything

Stk lvls: BASIC main loop can use anything

NOTE:
    RDATTY sets P=0 to selec the following error options:

-- not a parse error
-- store ERRN (and ERRL if S13=1)
-- display "ERR:" (or ERR L<N>:")

Detail:

```
    =RDATTY P=     0
            LC(2)  =eDATTY
            GOLONG =MFERR
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | MB | Documentation |

24.11 CORUPT - Report "System Error" error

Category: SYSTEM File: SG&EXC::MS

Name:(S) CORUPT - Report "System Error" error

Purpose:
To report "System Error" as an execution error.

Entry:
P = 0
S13=0 if not a running program (i.e., keyboard execution error)
S13=1 if running program
No other necessary conditions.

Exit:
Exits to BASIC main loop (ERRRTN)

Calls: MFERR

Uses....... BASIC main loop can use anything

Stk lvls: BASIC main loop can use anything

NOTE:

Setting P=0 selects the following error options:
- -- not a parse error
- -- store ERRN (and ERRL if S13=1)
- -- display "ERR:" (or "ERR L<N>:")

**Detail:**

```
=CORUPT LC(2)  =eMMCOR
        GOLONG =MFERR
```

**History:**

| Date | Programmer | Modification |
|---|---|---|
| 11/09/83 | MB | Documentation |

## 24.12 MFERR - Mainframe BASIC system error

Category: SYSTEM File: TI&ERD::MS

**Name:(S) MFERR - Mainframe BASIC system error**

**Purpose:**
Generate a BASIC system error from the mainframe tables. See BSERR entry for details.

## 24.13 BSERR - BASIC system error

Category: SYSTEM File: TI&ERD::MS

**Name:(S) BSERR - BASIC system error**

Purpose:
BSERR -- Generate a BASIC system error.
MFERR -- First sets C(3-2)=00, then falls into BSERR.

Entry: See MFERR*
Exit: See MFERR*
Uses: See MFERR*. Also S14, S1, S0.
Calls: MFERR*

Stk lvls: 3

NOTE:
MFERR and BSERR are generally for errors generated by the BASIC system, as they exit to the BASIC main loop. Those applications which wish to simply display an error and return should call MFERR* (a subroutine).

Detail:
MFERR -- Set C(3-2)= 00 for mainframe LEX ID.
BSERR -- Call MFERR*
Set NoCont flag (stop execution)
Clear END statement flag
Set Error flag
Exit through BASIC loop

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 06/29/82 | MB | documentation |
| 03/29/83 | JP | Set ERROR flag; Clear END flag |

## 24.14 MFERRS - Stop BASIC execution for error

Category: SYSTEM File: TI&ERD::MS

Name:(S) MFERRS - Stop BASIC execution for error

Purpose:
Return to BASIC main loop with status bits set to cause execution to stop.

Entry:
    P = 0

Exit:
    To ERRRTN (BASIC main loop)

Calls: Exits to ERRRTN (BASIC main loop)

Uses.......
 Exclusive: S13, S4, S0
 Inclusive: BASIC main loop uses everything.

Stk lvls: 0 (see BASIC main loop: RUNRTN)

NOTE:
    Standard entry point to stop BASIC execution because
    of an error.

Algorithm:
    Set status NoCont=1
    Set status sENDx=0
    Set status sERROR=1
    Exit to ERRRTN

History:

| Date | Programmer | Modification |
|---|---|---|
| 10/31/83 | MB | documented |

## 24.15 MEMERR - Insufficient Memory error

Category: SYSTEM File: TI&ERD::MS

Name:(S) MEMERR - Insufficient Memory error
Name:(S) MEMERX - Insufficient Memory error

Purpose:
    Process "Insufficient Memory", exit to BASIC main loop.

Entry:

```
     MEMERR -- No required conditions.
     MEMERX -- P=entry options as in MEMER*

  Exit:
     P      = 0
     Available Memory recoverd (AvMemSt and AvMemEnd
       collapsed).

  Calls:      MEMER*

  Stk lvls:   3

  NOTE:
     See MEMER* for all details.

  Detail:
     MEMERR -- sets P=0
     MEMERX -- sets C(3-0)= eMEM (18hex)
               falls into MEMER*
               exits to BASIC main loop with:
                     S14=1 (NoCont)
                     S0=1  (sERROR)
                     S1=0  (sENDx)

  History:

    Date      Programmer             Modification
  --------    ----------    ------------------------------------
  10/05/82    MB            Wrote code, documentation
```

24.16 MEMER* - Low-level memory error

```
     Category:  SYSTEM     File:  TI&ERD::MS


  Name:(S) MEMER*  -  Low-level memory error

  Purpose:
     Display low-level memory error to the user.

  Entry:
```

```
(1)-----------------
 | (same as MFERR*)
 |
 | P= (1xxx)!! Indicates Parse error.  THIS SHOULD
 |              NEVER BE SET FOR A MEMERR!  MEMER*
 |              collapses AvMemSt, causing the
 |              input buffer (address in INBS) to
 |              be destroyed!
 |     x1xx    Do not store ERRN
 |              (Else store ERRN and ERRL)
 |     xx1x    Display msg only (Else display
 |              "ERR:" or "ERR L:", too)
 |     bit0 not used at present (**)
 |


(2)-----------------
 | (same as MFERR*)
 |
 | C(B)= message ID number in Hex.
 | C(3-2)= LEX ID# in Hex  (=00 for mainframe tbl)
 |


(3)-----------------
 | (same as MFERR*)
 |
 | NEVER CALL MEMER* AS A PARSE ERROR! (I.e., never
 | enter with P=1xxx.)
 |
```

(**) Bit0 of the P register is reserved for future applications, as a way for the LEX file which generated the error to communicate with other LEX files; this bit can be detected during the pMEM poll in R0(S). The meaning of this bit is not yet decided. In the meantime, bit0 must=0.

```
Exit:
    P        = 0

Calls:       FPOLL, COLLAP, CLCOLL, AUTCLR, TRNFCK,
             MFER.6 (MFER.6 is an entry point in MFERR* --
               see MFERR* for more details)

Uses.......
 Exclusive:  A(W), B(W), C(W), D(W), P, D0, D1, R0
             S13 is tested for: "Running program?"
                 If you're calling this routine just for
                 message display, watch out for S13!!!
             Available Memory (starting at AvMemSt) is
```

also used as a building buffer for msg.

Inclusive: Same

Stk lvls: 2

NOTE:

The entry point MEMER* allows ANY message to be reported in lieu of "Insufficient Memory", and still be handled as a memory error. This means you can display, say, "Out of Scratch Area" as a way of reporting a memory error. This capability is included to allow external systems to generate memory errors and report them as they desire. But this capability can cause serious conditions (such as an infinite MEMERR loop) if some rules are not followed:

1) Never invoke MEMER* (or MEMERR or MEMERX) as a parse error.
2) Any error entering through MEMER* (includes MEMERR and MEMERX) disallows text insertion. This can be overridden in the pMEM poll. But never use a message which contains a type{5} insertion!!! A type{5} insertion may cause a slow pTRANS poll to be issued, which may cause an infinite MEMERR loop.

The preferred way for a LEX file operating in the BASIC system to generate a different memory error (i.e., other than "Insufficient Memory"), is to call MEMERR and then intercept the pMEM poll to change the message number or options. On the other hand, a LEX file which wants to generate a memory error which takes text insertions should set up the insertion codes in R2, call MEMER* with the appropriate message number, and adjust C(14-13) during the pMEM poll.

Detail:

R0 usage:

```
 F E D C B A 9 8 7 6 5 4 3 2 1 0
--------------------------------
| |  |F|                |      |
--------------------------------
 | | +- error code      +- msg number
 | +- insert codes
 +- option flags
```

Algorithm:

(1) Put option flags in C(S).
    Save options and LEX#, msg# in R0.
    Set C(14-12)=00F (suppresses text insertions)
    Call FPOLL
    Collapse Available Memory

Turn off AUTO mode
Check if TRANSFORM in effect (this essentially
  include TRANSFORM in the poll); if so
  branch back to TRANSFORM.
Jump to MFER.6 (see MFERR*)

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 10/05/82 | MB | documentation |

```
+---------------------------------------------+------------------+
|                                             |                  |
|  TIME    -  Time and Date Utilities         |  CHAPTER  25     |
|                                             |                  |
+---------------------------------------------+------------------+
```

## 25.1 CMPT - Return Current Time

Category: TIME File: MN&TM::MS

Name:(S) CMPT - Return Current Time

Purpose:
Read current time in 512ths since time 0.

Entry:
None.

Exit:
Current time in C and R1 (HEX ticks).
(Time represented as # of 512ths sec since midnight 1 Jan 0000).
R0 = TIMER value corresponding to current time.
HEX mode.
Carry clear.
P=0.

Calls: GETTIM, GETIRQ, GETLAF, GETAF, IDIV, PUTLAF, CLKUPD (falls through).

Uses.......
A,B,C,D,P,R0,R1,D0,D1,S0-S11

Stk lvls: 1

Detail:
Routine computes current time (NXTIRQ-TIMER) and places value of TIMER corresponding to current time in R0. Then accuracy factor corrections are computed and the code falls through to CLKUPD to perform an update.

Algorithm:
Read TIMER; save in R0.
Read NXTIRQ; current time = NXTIRQ-TIMER; save in R1.

```
Read TIMLAF; compute #ticks since last AF correction
  (TIME-TIMLAF); stash in D.
Compute (TIME-TIMLAF)/abs(AF); quotient to A;
  remainder to B.
Compute #ticks from old TIMLAF to new TIMLAF =
  (TIME-TIMLAF)-REMAINDER -> D.
Negate A (quotient from division) if AF is negative.
[At this point, A=time correction, D=#ticks from old
  TIMLAF to new TIMLAF.]
TIMLAF = TIMLAF + A + D.
TIME = TIME (from R1) + A.  Store TIME in R1.
Fall through to CLKUPD.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/07/82 | NM | Added documentation |

## 25.2 SETIME - Set And Normal Adjust Routine

Category: TIME        File: MN&TM::MS

Name: SETIME - Set And Normal Adjust Routine
Name:(S) ADJN - Set And Normal Adjust Routine

Purpose:
Set new system time and keep track of error for accuracy factor computation.

Entry:
SETIME, ADJN:
R1 = Current time (512ths sec since year 0).
R0 = Timer value corresponding to current time (from CMPT).
R2 = New time to set (512ths sec since year 0).
HEX mode.

Exit:
R1=New time (R2 on entry).
R0=New timer value corresponding to time.

Carry clear.
P=0.

Calls: CMPTE, GETOFS, PUTOFS, GETLST, PUTLST, GETLAF, PUTLAF, CLKUPD (falls through)

Uses.......
A,B,C,D,P,D0,D1,R0,R1,S0-S11.

Stk lvls: 2

Detail:
SETIME, ADJN are two names for same entry point.

The adjustment amount is rounded to the nearest half-hour. The difference between that and the adjustment amount (which will be between -15 and +15 minutes) is considered the error adjustment. The rest of the adjustment is considered a time zone change, and is not added to TIMOFS (time error accumulator).

Algorithm:
```
Q := Newtime - currenttime {total adjustment amount}.
Te := sign(Q)*((abs(Q)+15) mod 30 - 15) {error
  adjustment amount: between -15 and +15 minutes}.
TIMLST := TIMLST + Q - Te {update TIMLST by non-error
  amount}.
TIMLAF := TIMLAF + Q {update TIMLAF by adjustment
  amount}.
TIMOFS := TIMOFS + Te {update error accumulator by
  error amount}.
Fall through to CLKUPD.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/08/82 | NM | Added documentation |

## 25.3 ADJA - Absolute Time Adjust Routine

Category: TIME File: MN&TM::MS

Name:(S) ADJA - Absolute Time Adjust Routine

Purpose:
Set new system time without timebase accuracy correction. The entire adjustment amount is considered a time zone change... none of it is an accuracy adjustment.

Entry:
R1 = Current time (ticks since year 0).
R0 = Timer value corresponding to current time (stored when CMPT was done) (ticks).
R2 = New time to set (ticks since year 0).
HEX mode.

Exit:
R1=New time (R2 on entry).
R0=New timer value corresponding to time.
P=0.
Carry clear.

Calls: GETLST, PUTLST, GETLAF, PUTLAF, CLKUPD (falls through)

Uses.......
A,B,C,D,P,D0,D1,R0,R1,S0-S11.

Stk lvls: 1

Algorithm:
Q := Newtime - currenttime {total adjustment amount}.
Te := 0 {error adjustment amount = 0}.
TIMLST := TIMLST + Q - Te {update TIMLST by non-error amount}.
TIMLAF := TIMLAF + Q {update TIMLAF by adjustment amount}.
TIMOFS := TIMOFS + Te {update error accumulator by error amount}.
Fall through to CLKUPD.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/08/82 | NM | Added documentation |

## 25.4 EXACT - Compute New Accuracy Factor.

Category: TIME File: MN&TM::MS

**Name:(S) EXACT - Compute New Accuracy Factor.**

**Purpose:**
Inform time system that time currently contained is exact.

The first time EXACT is called after a coldstart or a RESET CLOCK, the exact flag is clear. This routine will simply set it, note the current time and start a new adjustment period.
Each subsequent call will note the elapsed time since the last call and the corrections which have been applied since the last call. From this an accuracy factor is computed.

**Entry:**
None.

**Exit:**
A new adjustment period has been started.
Carry set: Reasonable accuracy factor computed.
Carry clear: Illegal accuracy factor computed.

**Calls:** CMPT, GTFLAG, COMPAF, PUTAF, PUTOFS PUTLST.

**Uses.......**
A,B,C,D,P,D0,D1,R0,R1,S0-S11.

**Stk lvls:** 2

**Algorithm:**

```
If exact=true then
  compute AF
  if AF valid then store AF.
TIMLST:=TIMLAF:=currenttime  {start of new adjustment
  period}.
TIMOFS:=0.
EXACT:=true.
return with carry clear if:
  exact was false
  exact was true, computed AF is valid.
```

return with carry set if:
exact was true, computed AF was invalid.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/08/82 | NM | Added documentation |

## 25.5 SETALM - Set Absolute Alarm Time

Category: TIME File: MN&TM::MS

Name:(S) SETALM - Set Absolute Alarm Time

Purpose:
Set detonation time for any of alarms 1-6.

Entry:
Alarm time in A[11-0] (ticks since 1 Jan 0000).
Alarm#-1 (0-5) in C[0].

Exit:
Through CMPT.
Carry clear.
P=0.
R1 = Current time (512ths sec since year 0)
R0 = timer value corresponding to current time.

Calls: GETPND, PUTPND, CMPT (falls through).

Uses.......
A,B,C,D,P,D0,D1,S0-S11,R0,R1.

Stk lvls: 2

Algorithm:
Write alarm time to proper RAM location (ALRM1-ALRM6).
Clear proper bit (0-5) in PNDALM.
Fall through to CMPT.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/09/82 | NM | Added documentation |

## 25.6 SETALR - Set Alarm Relative To Current Time

Category: TIME File: MN&TM::MS

Name:(S) SETALR - Set Alarm Relative To Current Time

Purpose:
Set alarm time relative to current time.

Entry:
A[11-0] = Interval (512ths sec)
C[0] = Alarm#-1

Exit:
Through CLKUPD.
Carry clear.
P=0.
R1 = current time (512ths sec since year 0).
R0 = timer value corresponding to current time.

Calls: CMPT, SETALM (falls through).

Uses.......
A,B,C,D,P,D0,D1,R0,R1,R3,S0-S11.

Stk lvls: 2

Algorithm:
Add interval to current time.
Wrap around end-of-time.
Write out new alarm time to appropriate slot.
Update clock.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/09/82 | NM | Added documentation |

## 25.7 YMDHMS - Return Time And Date

Category: TIME File: MN&TM::MS

Name:(S) YMDHMS - Return Time And Date
Name:(S) YMDH01 - Convert Time To YYMMDD And HHMMSS

Purpose:
YMDHMS: Return current time and date in format compatible with file header time/date field.
YMDH01: Convert passed time (seconds since year 0) into time/date format compatible with file header time/date field.

Entry:
YMDHMS: None.
YMDH01: C[W]=Time (seconds since midnight, 1 Jan 0000).

Exit:
C = 0000YYMMDDHHMMSS. (year,mo,day,hrs,min,sec)
A[B] = HH (same as HH in C).
B[B] = MM (same as MM -- minutes in C).
D[B] = SS (same as SS in C).
HEX mode.
Carry clear.

Calls: CMPT, TIMRND, TODT, DAYYMD, SECHMS.

Uses.......
A,B,C,D,P,D0,D1,R0,R1,S0-S11.

Stk lvls: 2

Algorithm:
Get current time.
Compute day#, time-of-day.
Compute YYMMDD from day#.

Compute HHMMSS from time-of-day.
Format into YYMMDDHHMMSS.

History:

| Date | Programmer | Modification |
|---|---|---|
| 60/11/82 | NM | Added documentation |

## 25.8 SETTMO - Set System Timeout

Category: TIME File: MN&TM::MS

Name:(S) SETTMO - Set System Timeout

Purpose:
Set 10-minute system timeout.

Entry:
None.

Exit:
Carry set.
HEX mode.
10-minute timeout alarm has been scheduled.

Calls: STO1, SFLAG?, SETALR, SETALM, RCO1.

Uses.......
A,B,C,D,P,DO,D1, SCRTCH[0-31], SCREXO.

Stk lvls: 3

Detail:
Typically used to schedule automatic power-down.
Also used to schedule timeout during "Align" message in card reader.
If =flCTON (continuous on) flag is set, the timeout is disabled (never comes due).

Algorithm:

Stash scratch regs.
If fICTON set, set ALRM4 = 0 (SETALM)
  else set ALRM4 = current time + 10 minutes (SETALR).
Clock update (CMPT).
Restore scratch regs.

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/11/82 | NM | Added documentation |

## 25.9 TODT - Time To Time-of-day And Day#

Category: TIME File: MN&TM::MS

Name:(S) TODT - Time To Time-of-day And Day#

Purpose:
Convert from time (since 0000) to day# (since day 0) and time-of-day (since midnight).

Entry:
C = Time (HEX seconds).
Hex mode.

Exit:
B,C = Time-of-day (HEX seconds).
A = Day# (HEX days since day 0).
Hex mode.
P=15.
Carry set.

Calls: IDIV (falls through)

Uses.......
A,B,C,P

Stk lvls: 0

Detail:

The following terms are used in this and the following documentation:

time: time in seconds since midnight 1 jan 0000
time-of-day: seconds since midnight.
day#: day# relative to 1 jan 0000
h,m,s: hours, minutes, seconds.
d,m,y: day, month, year.

Date routines are valid from 1 jan 0000 to
31 dec 9999.
Assumptions being made in the date routines are:
year<=9999
month<=12
day<=31 (this is intentionally violated for JD2DAY)
day#<=3652424
THIS MEANS THAT HIGHER-ORDER DIGITS ARE ZEROES!!

Algorithm:
Day#=Time div 15180H.
Time-of-day=Time mod 15180H.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| 05/24/82 | NM | Added documentation |

25.10 SECHMS - Convert Secs To Hours, Mins, Secs

Category: TIME File: MN&TM::MS

Name:(S) SECHMS - Convert Secs To Hours, Mins, Secs

Purpose:
Convert time in seconds (expressed in HEX) to hours, minutes and seconds (expressed in DEC).

Entry:
C[W] = Time-of-day (HEX seconds).

Exit:
A[W] = Hours (BCD integer).
B[W],C[W] = Minutes (BCD integer).
D[W] = Seconds (BCD integer).
HEX mode.
Carry clear.
P=15.

Calls: HEXDEC, IDIV.

Uses.......
A,B,C,D,P.

Stk lvls: 1

Algorithm:
Convert to decimal.
Divide by 60; remainder=secs.
Divide quotient by 60; remainder = minutes,
quotient = hours.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/27/82 | MM | Added documentation |

## 25.11 HMSSEC - Hours, Mins, Secs To Seconds.

Category: TIME File: MN&TM::MS

Name:(S) HMSSEC - Hours, Mins, Secs To Seconds.

Purpose:
Convert from hours, minutes, secs (DEC) to seconds (HEX).

Entry:
A[W] = Hours (BCD integer).
B[W] = Minutes (BCD integer).
D[W] = Seconds (BCD integer).

Exit:
    A,B,C = Seconds since midnight (HEX).
    HEX mode.
    P=0.
    Carry clear.

Calls: MP60, IDIV.

Uses.......
    A,B,C,D,P.

Stk lvls: 1

Algorithm:
    Compute ((hrs * 60) + mins) * 60 + secs.
    Convert to HEX.

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/27/82 | NM | Added documentation |

## 25.12 YMDDAY - Convert Year,month,day To Day#

Category: TIME    File: MN&TM::MS

Name:(S) YMDDAY - Convert Year,month,day To Day#

Purpose:
    Convert date to absolute day#.

Entry:
    A = Year (BCD number).
    B = Month (BCD number).
    D = Day (BCD number).

Exit:
    A,B,C = Day# since day 0 (HEX).
    HEX mode.
    P=0.

Carry clear.

Calls: M306, SUM3, DECHEX (falls through)

Uses.......
A,B,C,D,P

Stk lvls: 1

Detail:
Day# is expressed relative to 1 January 0000.

Algorithm:

Define the following conditionally depending on the value of MONTH:

```
If MONTH < 3 then let M = MONTH + 13
              and let Y = YEAR - 1.
If MONTH >= 3 then let M = MONTH + 1
              and let Y = YEAR.
```

Also define the following functions:

```
SUM3(Y) = int(Y * 365.25) - int(Y / 100) + int(Y / 400)
        = -366 if Y=-1
M306(M) = int(M * 30.6001)
```

Mapping DATE to DAY NUMBER:

```
DAYN(MONTH,DAY,YEAR) = SUM3(Y) + M306(M) + DAY - 63
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/27/82 | NM | Added documentation |

## 25.13 DAYYMD - Day# To Year, Month, Day

Category: TIME File: MN&TM::MS

Name:(S) DAYYMD - Day# To Year, Month, Day

Purpose:
Convert from absolute day# to date.

Entry:
C = Day# since day 0 (HEX).

Exit:
A = Year (BCD number).
B = Month (BCD number).
D = Day (BCD number).

Calls: HEXDEC, ESTYO, SUM3, CHKYO, IDIV, M306, ASLW4

Uses.......
A,B,C,D,P

Stk lvls: 1

Algorithm:

Define the following conditionally depending on the value of MONTH:

```
If MONTH < 3 then let M = MONTH + 13
                  and let Y = YEAR - 1.
If MONTH >= 3 then let M = MONTH + 1
                  and let Y = YEAR.
```

Also define the following functions:

```
SUM3(Y) = int(Y * 365.25) - int(Y / 100) + int(Y / 400)
        = -366 if Y=-1
M306(M) = int(M * 30.6001)
```

Mapping DAY NUMBER to DATE:

Calculate the value of YO as follows:

```
YO = int( [(DAY# + 63) - 121.5] / 365.2425)
```

This is an approximation of the correct year.

\# Now calculate MO as follows:

```
MO = int( [(DAY# + 63) - SUM3(YO)] / 30.6001)
```

If this MO is less than 4 then the year was one too high; therefore let YO = YO - 1 and recalculate MO using the new YO (ie YO := YO - 1 ; GO TO #).

Once a value for MO greater than or equal to 4 is

```
obtained, the values of MONTH, DAY, and YEAR are
calculated as follows:
  DAY = [(DAY# + 63) - SUM3(YO)] - M306(MO).
  If MO >=14 then MONTH = MO - 13 and YEAR = YO + 1.
  If MO < 14 then MONTH = MO - 1 and YEAR = YO.
```

360-day calendar is not done in this code. Here is how to do it:

For 360 day calendar, the number of days between two dates is calculated as follows:

```
Let M1 = month of first date
Let D1 = day of month of first date
Let Y1 = year of first date
Let M2 = month of second date
Let D2 = day of month of second date
Let Y2 = year of second date
```

```
Now make the following adjustments:
     If D1 >= 30 then
       begin
         D1 := 30;
         if D2 = 31 then D2 := 30
       end;
Now compute:
  Delta-days = (Y2-Y1)*360  +  (M2-M1)*30  +  (D2-D1)
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 05/27/82 | NM | Added documentation |

## 25.14 DAY2JD - Day# To Julian Date

Category: TIME    File: MN&TM::MS

Name:(S) DAY2JD - Day# To Julian Date

Purpose:

Convert day# (since 1 Jan 0000) to Julian date (year and day-in-year)

Entry:
C[W] = Day# (HEX days since day 0).

Exit:
A[W] = Year (BCD number).
B,C = Day-of-year (BCD number).
DEC mode.

Calls: HEXDEC, ESTYO, SUM3, CHKYO.

Uses.......
A,B,C,D,P.

Stk lvls: 1

Algorithm:

```
     Convert day# to DEC.
     Estimate YO.
  1: Compute SUM3(YO).
     CHKYO; if too high, decrement and goto 1.
     If SUM3(YO) <= 365 then goto 2.
     Day-in-year = SUM3(YO)-365.
     Year = YO+1.
     RTN.
  2: If year divisible by 100 then point at digit 2,
       else point at digit 0.
     If selected digit divisible by 4 then
         day-in-year=SUM3(YO)+1
       else
         day-in-year=SUM3(YO).
     Year = YO.
     RTN.
```

History:

| Date | Programmer | Modification |
|---|---|---|
| 06/03/82 | NM | Added documentation |

```
+---------------------------------------+-----------------+
|                                       |                 |
|  VARMGT  -  Variable Management       |  CHAPTER  26    |
|                                       |                 |
+---------------------------------------+-----------------+
```

## 26.1 STRASN - String Assignment

Category: VARMGT     File: AB&ASN::MS

```
Name:(S) STRASN  -  String Assignment

Purpose: Store a string from stack to a string variable

Entry:
     D1 = Stack pointer
     A  = String header from stack ( A=DAT1 W)
     S-R0-0 = Destination address ( @ String length)
            = 00000 if hokey destination.

Exit:
     P = 0
     Carry clear => No error
         Carry set => String too long

Calls:       MOVED3, MOVEU3

Uses:        A,B,C,D

Stk lvls:    2

History :

Date         Programmer   Modification
----------   -----------  ------------------------------
6/17/82      SC           straight line code=> subroutine
```

## 26.2 DEST - Save Variable Destination Info

Category: VARMGT File: AB&ASN::MS

Name:(S) DEST - Save Variable Destination Info

Purpose:
Save variable destination information for use by STORE subroutine.

Entry:
B = Exit condition from EXPEXC (see note below)
D1 = Exit condition from EXPEXC (see note below)
F-R1-0 = Exit condition from EXPEXC (see note below)
F-R1-3 = Exit condition from EXPEXC (see note below)

Exit:
P=0.
Following information has been stored:
S-R0-1 = First substring parameter.
S-R0-2 = Second substring parameter.
S-R0-3 = Variable type.
S-R1-0 = Array element number.
S-R1-1 = Maximum string length.
S-R1-3 = Subscript count.

Calls: None.

Uses.......
D1,C.

Stk lvls: 0

NOTE:
Whenever EXPEXC evaluates a variable (simple or array element), it leaves destination information about that variable in B[W] and function scratch. This routine puts that information in statement scratch, where it is safe from further abuse during expression execute, and can be subsequently accessed for a store operation.

In computing the destination information, the recall code sets up information about the variable's address, substring parameters, type, array register number, maximum string length and subscript count. If the variable does not exist, that fact is somehow encoded into this information and the variable will be created in the store subroutine.

Detail:
Typically called after EXPEXC, which left information around about the location of the last variable evaluated (if evaluating a variable was the last thing done). Typical use is in variable assignment:
EXPEXC (evaluate destination variable).
DEST (save destination information for STORE).
EXPEXC (evaluate expression).
STORE (store result in destination variable).

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.3 BASE - Determine Option Base

Category: VARMGT File: AB&ASN::MS

Name:(S) BASE - Determine Option Base

Purpose:
Determine whether we are in option base 0 or 1.

Entry:
HEX mode.

Exit:
If carry set:
We are in option base 1. C[XS]=1.
If carry clear:
We are in option base 0. C[XS]=0.

Calls: None.

Uses.......

DO,C[XS].

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.4 SHRT - Store Into Short Variable

Category: VARMGT File: AB&ASN::MS

Name:(S) SHRT - Store Into Short Variable

Purpose:
Store a number into a short variable, with IEEE rounding.

Entry:
12-digit form in A[W].
DO pointing at variable storage location.

Exit:
R3 contains copy of number as stored.
DEC mode

Calls: SPLITA, uRESNX.

Uses.......
DO,D1,A,B,C,D,RO,R3,S7-S11.

Stk lvls: 3

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |

| | | |
|---|---|---|
| 10/13/83 | NM | Attempted to document |

## 26.5 INTGR - Store Into An Integer Variable

Category: VARMGI File: AB&ASN::MS

Name:(S) INTGR - Store Into An Integer Variable

Purpose:
Store a number into an integer variable.

Entry:
Number in 12-digit floating-point form in A.

Exit:
P = 0

Calls: IF12A, OVFL, RND-12, SIGCHK, uRESXT.

Uses.......
A,B,C,D,D0,D1,R0,R3,S7-S11.

Stk lvls: 3

Detail:
Handles overflow according to IEEE trap settings.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.6 DYNAMC - Variable Recall

Category: VARMGT     File: AB&EXP::MS

```
Name:     DYNAMC  -  Variable Recall
Name:     STATIC  -  Variable Recall
Name:(S)  RECALL  -  Variable Recall

Purpose:
     Recall a variable.
     Also set up destination address information for
     possible use by DEST after expression execution
     terminates.

Entry:
     P=0.
     HEX mode.
  STATIC: Expression execution controller jumped on variable
             token (non-alpha-digit).
          D0=PC.
          D1=top of stack.
  DYNAMC: Expression execution controller jumped on alpha-
             digit variable token.
          D0=PC.
          D1=top of stack.
  RECALL: D0=PC.
          A[A]=top of stack.
          D0,B[A]=address of variable register (register
             contains variable if simple, else contains
             dope vector).

Exit:
     Through FNRTN2.
     D0=PC, pointing past expression.
     D1=stack pointer.
     Value recalled in on top of stack.

Calls:
     If we are end of expression (this recall is last thing
       done): ADRS10, ADRS40, MOved3, READIN, RECADR.
     If we are not at end of expression, control reverts
       to expression execution controller, which could call
       anything.

Uses.......
             If we are not at end of expression: everything
```

available to expression execution controller.
If we are at end of expression: A-D,D0,D1,P.

Stk lvls: 4
2, if we are at end of expression.

NOTE:

This is part of expression execution. It does not return, it goes back to the expression execution controller. The way to use this routine is to set up the tokenized form of the variable you want to access (whether for recall or for computing the store address), complete with a terminator, point D0 at it and perform an expression execute. You can, with some cleverness, set things up to look as though an expression execution is in progress and call this code instead of calling EXPEXC. This might save a little execution time.

Detail:

In addition to recalling the variable, this routine sets up information relevant to using the variable as a destination. This information includes the variable address, substring parameters, type, array register number, maximum string length and subscript count. If this is the last thing done before the expression terminates, that information is intact upon return from the expression execution controller, and can be passed to the DEST subroutine for storage somewhere safe.

WHY? One purpose of this code is to evaluate a variable on the left side of an assignment operator (=) so it can be stored into after the expression on the right side is evaluated. DEST serves the purpose of saving the destination information so the assignment can take place later.

The destination information is stored in function scratch and B[W]. DEST moves it to statement scratch.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.7 RECADR - Some Recall Utility

Category: VARMGT File: AB&EXP::MS

Name:(S) RECADR - Some Recall Utility

Purpose:
Perform D0:=D0+11 ; C[9-5]:=D0-C[9-5]. Evidently useful for recalling things.

Entry:
Things in C and D0.
HEX mode.

Exit:
D0 has been incremented by 11.
C[9-5] = New D0 - C[9-5].
HEX mode.

Calls: 0

Uses.......
D0,C[9-5].

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 11/09/83 | NM | Attempted to document |

## 26.8 ADRSUB - Get Variable Name From Token Stream

Category: VARMGT File: AB&EXP::MS

Name:(S) ADRSUB - Get Variable Name From Token Stream

Purpose:
Read a token stream for a variable and return 3-digit code for that variable

Entry:
P=0.
HEX mode.
D0 points at token stream

Exit:
P=0.
B(X) = 3-digit code for variable
(Defining aa = ASCII code for variable name)
= 0aa if simple variable.
= qaa if alpha-digit variable, where q = digit+1.
= 0bb if string var, where bb = aa ! 20H.
= qbb if alpha-digit string var.
D0 points past last byte of variable tokenization.
Carry set

Calls: None

Uses.......
Inclusive: B(X),C(X),D0.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.9 ADDRSS - Find Address Of A Variable

Category: VARMGT File: AB&EXP::MS

Name:(S) ADDRSS - Find Address Of A Variable

```
Name:(S) ADRS40  -  Find Address Of A Variable
Name:(S) ADRS50  -  Find Address Of Var Not Of Parm Chain
Name:(S) FIND    -  Find Address Of Var Not Of Parm Chain
Name:(S) ADRS80  -  Find Address Of Var Not Of Parm Chain

Purpose:
  ADDRSS, ADRS40: Search parameter chain and then variable
                  chains to find a variable.
  ADRS50: Search variable chains to find variable (do not
          search parameter chain.
  FIND  : Same as ADRS50 except search already in progress.
  ADRS80: Same as FIND except DAT0 already read.

Entry:
     P=0.
     ADDRSS:  D0 points at token stream of variable to be
                   found.
     ADRS40:  B[X] contains 3-digit code for variable to be
                   found.
     ADRS50:  B[X] contains 3-digit code for variable to be
                   found.
     FIND  :  Search already in progress.  B[X] as above.
              D0 points at a variable name entry in variable
                chain.
              D[B] = #entries left in chain.
     ADRS80:  Same as FIND + C[X] contains entry already
                read at D0.

Exit:
     P      =  0
     Carry set if variable not found
     Carry clear if variable found
            D0,B(A) = Address of variable register
     A[A] = D0 at time of entry (if ADRS40 called).
            Pointer past variable tokenization (if ADDRSS
                               called).
            A[A] at time of entry (if ADRS50, ADRS80
                               called).

Calls:      CHNHED,ADRS70,ADDRSS calls ADRSUB

Uses.......
            D0,A(A),B(A),C(6-0),D(A)

Stk lvls:   1

Detail:
     First searches parameter chain for variable (in case
     passed in CALL).  Then searches variable chain.

History:
```

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.10 CHNHED - Point To Variable Chain Head

Category: VARMGT    File: AB&EXP::MS

**Name:(S) CHNHED - Point To Variable Chain Head**

**Purpose:**
Point to variable chain head and return # entries in chain.

**Entry:**
P=0.
HEX mode.
B[X] = three-digit variable name (see ADRSUB doc hdr).

**Exit:**
P=0.
HEX mode.
D[B]=# items in chain - 1.
Carry set iff chain empty.
C[A], D0=pointer to chain head.

**Calls:** None.

**Uses.......**
C[A],C[6-0],D0.

**Stk lvls:** 0

**History:**

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/13/83 | NM | Attempted to document |

## 26.11 DPVCTR - Creates Vars, Computes # Of Elements

Category: VARMGT File: AB&REG::MS

Name:(S) DPVCTR - Creates Vars, Computes # Of Elements

Purpose:
Creates primary variables(dope vectors), computes number of array units to allocate

Entry:
Same as exit conditions from PREP, ie
P = 0
D0 points to dimension expression(s) if array
A(X),(S-R1-2) = 3-digit code for variable
B(A),(S-R0-0) = Address of variable(if it exists(S2=0))
(S-R0-1 thru S-R1-1) zeroed
Array(S0) set iff it is an array
NonEx(S2) set iff variable/array doesn't already exist
String(S1) set iff string variable/array
OpBase(S3) set iff OPTION BASE 1

Exit:
P = 0
C-register has the following information:

```
+------------+----------+----------+--+--+--+
| ////////// |dimlimt 1 |dimlimt 2 | b| d| t|
+------------+----------+----------+--+--+--+
                  4          4       1  1  1
```

where t is datatype indicator
d is dimcount
b is baseoption
dimlimt 2 is second dimlimit or max string length
dimlimt 1 is first dimlimit
or

```
+-------------------------------------+--+
|                zeroes               | t|
+-------------------------------------+--+
```

where t is datatype indicator (0 for real) for real, short, and integer simple variables.

```
A(A) = number of array units
B(X) = 3-nibble code for variable
S-R0-1 = 1st subscript if is an array
S-R0-2 = 2nd subscript if is a 2 dimensional array
       = Maximum string length if string
S-R1-0 = Number of elements for numeric array
```

**Calls:** LIMITS,GETDIM,A-MULT

**Uses.......**
**Inclusive:** A,B,C,D,R0,R1,R2,R3,R4,D0,D1

**Stk lvls:** 6

**History:**

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |

## 26.12 GETDIM - Get A Dimlimit From Stack

Category: VARMGT File: AB&REG::MS

**Name:(S) GETDIM - Get A Dimlimit From Stack**

**Purpose:**
Pop dimension limit from stack and check range.

**Entry:**
D1=stack pointer.

**Exit:**
P=0.
HEX mode.
Errors out if result comples (eDATTY) or out of range (eARGOR).
A[A]=dimlimit.

Calls: FLTDH, POP1N.

Uses.......
A,B,C,P.

Stk lvls: 2

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 26.13 SPACE - Compute Space Needs For An Array

Category: VARMGT File: AB&REG::MS

Name:(S) SPACE - Compute Space Needs For An Array

Purpose:
Calculate space requirements for an array.

Entry:
P=0.
A[A] = number of array units needed.
C[0] = data type:
A - Integer
B - Short real
C - Real
D - Short complex
E - Complex
Error exit (eMEM) if > address space.

Exit:
A,R0 = space requirements in nibbles.
P=0.

Calls: LENGTH, A-MULT.

Uses.......

A,B,C,R0.

Stk lvls: 1

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 26.14 PREP - Prepare To Create A Variable/array

Category: VARMGT File: AB&REG::MS

Name:(S) PREP - Prepare To Create A Variable/array

Purpose:
Prepare to create a variable or array

Entry:
D0 points to tokenization of a variable or array in some "dim" statement.

Exit:
P = 0
A(X),(S-R1-2) = 3-digit code for variable
B(A),(S-R0-0) = Address of variable(if it exists(S2=0))
(S-R0-1 thru S-R1-1) zeroed
Array(S0) set iff it is an array
NonEx(S2) set iff variable/array doesn't already exist
String(S1) set iff string variable/array
Carry and OpBase(S3) set iff OPTION BASE 1

Calls: ADDRSS,C=ACTV,BASE

Uses.......
Inclusive: D0,D1,S0,S1,S2,S3,A(A),B(A),C(W),D(A)

Stk lvls: 2

Note: Takes error exit if trying to change a function parameter.

History:

| Date | Programmer | Modification |
| --- | --- | --- |
| | SA | Wrote |

## 26.15 DMNSN - Create And Allocate Memory For Variable

Category: VARMGT File: AB&REG::MS

Name:(S) DMNSN - Create And Allocate Memory For Variable

Purpose:
Create simple numeric/string variable, numeric array and string vector.

Entry:
```
Array(S0)  = 1 Create array
           = 0 Create simple variable
String(S1) = 1 String variable
           = 0 Numeric variable
NonEx(S2)  = 1 Create new variable
           = 0 Redimension existing array
D = Dope vector of the variable
A = # of elements of the array
C = Element length in nibbles
D0 = PC
R2(X) = Variable name
S-R1-1 = Variable address if already exist
```

Exit:
Carry CLEAR if PC is pointing at end of line.

Calls: A-MULT, CR-VAR, CR-ARR, AJDEST, ARYSIZ, CR-ADJ, ADRS40, WIPOUT

Uses: A,B,C,D,R0,R1,R2,R3,S3,P

Stk lvls: 3

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |

## 26.16 DATLEN - Compute Data Length Given Type

Category: VARMGT File: AB&REG::MS

Name:(S) DATLEN - Compute Data Length Given Type

Purpose:
Compute length of a data item.

Entry:
C[0]=data type.
5 - Integer.
4 - Short real.
3 - Real.
2 - Short complex.
1 - Complex.
P=0.

Exit:
C[A]=Length of data item:
Integer: 6.
Short real: 9.
Real: 10H.
Short complex: 12H.
Complex: 20H.

Calls: None.

Uses.......
C.

Stk lvls: 0

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 26.17 ARYSIZ - Compute Array Size, # Elements

Category: VARMGT File: AB&REG::MS

Name:(S) ARYSIZ - Compute Array Size, # Elements
Name:(S) ARYELM - Compute Array Size, # Elements

Purpose:
ARYSIZ: Compute array size in bytes.
ARYELM: Compute number of elements in an array.

Entry:
D1 points at the dope vector of the array.

Exit:
P=0.
ARYELM: D1 points at first subscript limit.
A = number of elements in the array.
ARYSIZ: D1 points at the array pointer within the array dope vector.
C = array pointer (is an offset from the array pointer to the start of the array).
A = array size in nibbles.

Calls: ARYELM: A-MULT.
ARYSIZ: ARYELM, DATLEN, A-MULT.

Uses.......
A,B,C,D,D1.

Stk lvls: ARYELM: 1.
ARYSIZ: 2.

History:

| Date | Programmer | Modification |
|---|---|---|
| | SA | Wrote |
| 10/18/83 | NM | Attempted to document |

## 26.18 GETNAM - Get variable name

Category: VARMGT File: SC&SUB::MS

Name:(S) GETNAM - Get variable name

Purpose: Read the variable into B(X) and check if is a string or a number

Entry: D0 pts variable token
P=0

Exit: B(X) = Variable name
D0 past the variable name
S0 = 1 - is a string variable
0 - is a numeric variable
Carry set

Calls: ADRSUB

Uses: B(A),C, S0, D0

Stk lvls: +1

Version 79.10.13 of RUNIT's INDEX program

#CK, 16-26

-LINE, 12-5

1/X15, 13-2

?PRFI+, 9-4
?PRFIL, 9-4

A

A-MULT, 15-8
ACCEPT, 16-47
ACOS12, 13-19
ACOS15, 13-19
AD15m, 13-2
AD15s, 13-2, 15-28
AD2-12, 13-2
AD2-15, 13-2
ADDF, 13-2
ADDONE, 13-1
ADDP, 23-14
ADDRSS, 26-9
ADHEAD, 14-13
ADJA, 25-4
ADJN, 25-2
ADRS40, 26-10
ADRS50, 26-10
ADRS80, 26-10
ADRSUB, 26-9
ALINFO, 19-3
ALLDUN, 16-1
ARG12, 13-16
ARG15, 13-16
ARGERR, 24-1
ARGF, 13-16
ARGPR+, 14-8
ARGPRP, 14-9

ARGST-, 14-11
ARGSTA, 14-11
ARITH, 7-8
ARRY01, 16-29
ARRYCK, 16-29
ARYDC, 7-16
ARYELM, 26-18
ARYSIZ, 26-18
ASCO2, 7-15
ASCICK, 7-15
ASCII, 6-6
ASIN12, 13-19
ASIN15, 13-19
ASLW3, 11-23
ASLW4, 11-23
ASLW5, 11-23
ASNMNT, 22-1
ASNSTO, 22-1
ASRW3, 11-23
ASRW4, 11-23
ASRW5, 11-23
ATAN12, 13-19
ATAN15, 13-19
ATCHK, 8-17
ATNCLR, 11-33
AVE=C, 18-3
AVE=D1, 18-3
AVM2DS, 6-13
AVS2DS, 6-29
AVS=C, 18-13
AVS=DO, 18-13

B

BACK1B, 9-44
BACK2B, 9-44
BACK3B, 9-44
BASCHA, 9-22
BASCHK, 9-22
BASE, 26-3
BBCOLL, 18-13
BEEP, 22-16
BEEPP, 23-3
BF2DPP, 6-13
BF2DS+, 6-13
BF2DSP, 6-13
BF2ST+, 14-14
BF2STK, 14-14
BIASA+, 15-37
BIASA-, 15-37
BIASC+, 15-37

BIASC-, 15-37
BIG, 15-5
BIG+, 15-5
BLDBIT, 6-10
BLDCON, 15-3
BLDDSP, 6-8
BldIM+, 8-27
BldIMA, 8-26
BldIMG, 8-26
BLDLCD, 6-8
BLNKCK, 7-17
BOPNM-, 8-22
BP, 11-19
BP+, 11-19
BP+C, 11-19
BRT30, 13-19
BRTF, 13-19
BSCEX2, 24-3
BSCEXC, 24-3
BSCEXT, 24-3
BSERR, 24-16

C

C+A2D1, 11-18
CA2D1+, 11-18
CALBIN, 22-23
CALL, 22-20
CALLP, 23-13
CAT, 22-24
CAT$, 10-3
CAT$20, 10-3
CAT100, 22-24
CATC++, 16-42
CATCH+, 16-42
CATCHR, 16-42
CATEDT, 22-24
CHAIN*, 9-29
CHAIN+, 9-29
CHAIN-, 9-29
CHEDIT, 12-1
CHIRP, 11-20
CHKEOL, 8-59
CHKmem, 18-11
CHKSPC, 18-11
CHKSPF, 18-11
CHNHED, 26-11
CK"ON", 8-16
CKINF-, 8-53
CKINFO, 8-53
CkLoop, 8-20

CkLpNC, 8-20
CKSREQ, 11-37
CLASSA, 15-39
CLCOLL, 18-13
CLOSEM, 9-56
CLOSEA, 9-56
CLOSEF, 9-56
CLRFRC, 15-12
CMD1ST, 12-7
CMDFND, 12-8
CMDINI, 12-9
CMDPR", 6-7
CMDS00, 12-8
CMDS10, 12-8
CMDS20, 12-8
CMPT, 25-1
CNFFND, 4-13
CNFLCT, 24-1
CNV2UC, 5-6
CNVUCR, 5-7
CNVWUC, 5-6
COLDST, 24-10
COLLAP, 14-15
COMCK, 16-32
COMCK+, 16-24
COMCK1, 16-32
COMCK0, 16-24
COMPLM, 9-25
CONCOM, 16-48
CONF, 4-2
CONVUC, 5-7
COPYDC, 21-4
COPYu, 8-12
CORUPT, 24-15
COS12, 13-17
COS15, 13-17
COUNTC, 8-44
CPLM10, 9-25
CPLM15, 9-25
CRDFIL, 9-35
CREATE, 22-19
CREATF, 9-11
CRETF+, 9-11
CRFSB-, 9-50
CRFSUB, 9-50
CRLFND, 6-17
CRLFOF, 6-17
CRLFSD, 6-17
CRTF, 9-51
CSL9R0, 11-14
CSLC1, 11-9
CSLC10, 11-9

CSLC11, 11-9
CSLC12, 11-9
CSLC13, 11-9
CSLC14, 11-10
CSLC15, 11-10
CSLC2, 11-9
CSLC3, 11-9
CSLC4, 11-9
CSLC5, 11-9
CSLC6, 11-9
CSLC7, 11-9
CSLC8, 11-9
CSLC9, 11-9
CSLW3, 11-23
CSLW4, 11-23
CSLW5, 11-23
CSRC1, 11-9
CSRC10, 11-9
CSRC11, 11-9
CSRC12, 11-9
CSRC13, 11-9
CSRC14, 11-9
CSRC15, 11-9
CSRC2, 11-9
CSRC3, 11-9
CSRC4, 11-9
CSRC5, 11-9
CSRC6, 11-9
CSRC7, 11-9
CSRC8, 11-9
CSRC9, 11-9
CSRW3, 11-23
CSRW4, 11-23
CSRW5, 11-23
CURBOT, 7-1
CURDVC, 9-74
CURSFL, 6-3
CURSFR, 6-3
CURSRD, 7-1
CURSRT, 6-28
CURSRU, 7-1
CURTOP, 7-1
CVUCW, 5-6

D

D0+2RD, 9-42
D0=AVS, 18-10
D0=FIB, 18-4
D0=OBS, 18-9
D0=PCA, 18-10

DOASC+, 11-56
DOASCI, 11-56
D12ROA, 11-16
D1=AVE, 18-2
D1=IBS, 18-13
D1@AVS, 18-13
D1C=R3, 11-12
D1FSTK, 8-65
D1MSTK, 8-64
D=AVME, 18-1
D=AVMS, 18-1
D=WORD, 11-2
DATACK, 16-34
DATLEN, 26-17
DAY2JD, 25-16
DAYYMD, 25-15
DBLPI4, 15-42
DBLSUB, 15-41
DCHX=C, 5-2
DCHXF, 5-1
DCHXW, 5-10
DCPLIN, 7-1
DCRMNT, 8-42
DEBNCE, 12-13
DECDC, 21-1
DECHEX, 5-2
DECP, 23-9
DELAYp, 23-3
DEST, 26-2
DISPDC, 21-2
DISPP, 23-10
DIVF, 13-4
DMNSN, 26-16
DONNA, 6-27
DPART2, 8-50
DPART3, 8-51
DPVCTR, 26-12
DRANGE, 11-2
DROPDC, 21-2
DSLEEP, 11-33
DSP$00, 6-22
DSPBUF, 6-31
DSPCHA, 6-14
DSPCHC, 6-14
DSPCL?, 6-16
DSPCNA, 6-30
DSPCNB, 6-30
DSPCNO, 6-30
DSPLI+, 7-1
DSPLIN, 7-1
DSPPO2, 23-10
DSPRST, 6-20

DSPUPD, 6-11
DSIp, 23-7
DSIRDC, 21-1
DV15M, 13-5
DV15S, 13-4
DV2-12, 13-5
DV2-15, 13-4
DXP100, 13-9
DYNAMC, 26-6

E

EDIT, 9-71
EDIT20, 9-71
EDIT80, 9-71
EDITWF, 9-71
END, 22-5
END10, 22-5
END20, 22-5
ENDALL, 22-5
ENDBIN, 22-5
ENDDC, 7-18
ENDING, 8-36
ENDSUB, 22-23
EOLCK, 16-14
EOLCKR, 16-14
EOLDC, 11-45
EOLSCN, 8-70
EOLSN5, 8-70
EOLSN7, 8-70
EOLXC*, 7-18
EOLXCK, 11-45
ERR3, 16-17
ERRM$f, 14-16
ERRRTN, 24-3
ESCSEQ, 6-19
EX-115, 13-9
EX12, 13-13
EX15M, 13-13
EX15S, 13-13
EXAB1, 15-23
EXAB2, 15-23
EXACT, 25-5
EXCAD+, 2-3
EXCADR, 2-3
EXCHRe, 16-17
EXCPAR, 8-55
EXDCLP, 7-4
EXF, 13-13
EXITRN, 22-5
EXP15, 13-9

EXPEX+, 8-2
EXPEX-, 8-2
EXPEX1, 8-2
EXPEXC, 8-2
EXPP10, 16-44
EXPPAR, 16-44
EXPPLS, 16-44
EXPR, 8-3
EXPRDC, 7-4
EXPSKP, 9-59

F

FAC15S, 13-11
FACTF, 13-11
FASCFD, 9-48
FCHLBL, 9-24
FCSTRT, 13-11
FGTBL, 12-11
FIBAD-, 9-49
FIBADR, 9-49
FIBOFF, 9-57
FIBON, 9-57
FILCRD, 9-30
FILDC*, 7-22
FILEF, 9-67
FILEMF, 9-67
FILEP, 16-40
FILEP+, 16-40
FILEP-, 16-40
FILEP1, 16-40
FILFIL, 9-77
FILSK+, 9-76
FILSKP, 9-76
FILXQ$, 9-61
FILXQ^, 9-61
FIND, 26-10
FINDA, 11-29
FINDD0, 11-29
FINDF, 9-67
FINDF+, 9-67
FINDL, 9-16
FINDLB, 8-17
FINDWF, 9-67
FINITA, 15-21
FINITC, 15-22
FINLIN, 6-24
FIXDC, 21-2
FIXP, 23-2
FLADDR, 9-79
FLDEV+, 8-80

FLDEVX, 8-79
FLIP10, 15-45
FLIP11, 15-45
FLIP8, 15-45
FLOAT, 5-4
FLSKPB, 9-76
FLTDH, 5-1
FLTYPp, 16-39
FNDFCN, 9-60
FNPWDS, 15-22
FNRTN1, 8-3
FNRTN2, 8-3
FNRTN3, 8-3
FNRTN4, 8-3
FORUPD, 18-7
FPOLL, 17-21
FSPC10, 16-36
FSPECe, 16-17
FSPECp, 16-36
FSPECx, 9-65
FTBSCH, 9-47
FTYPDC, 7-26
FTYPF#, 9-46
FTYPFD, 9-47

G

GDISP$, 10-2
GETAVM, 18-2
GETCHM, 8-64
GETCON, 15-40
GETDIM, 26-13
GetEXP, 8-37
GETMSK, 6-12
GETNAM, 26-19
GETPeF, 9-20
GETPR, 9-75
GETPR+, 9-75
GETPR1, 9-75
GETPRO, 9-75
GETSA, 15-30
GETSDO, 15-30
GETST*, 9-20
GETST-, 9-20
GETSTC, 9-20
GETSTe, 9-20
GETVAL, 15-40
GNXCR+, 16-22
GNXTCR, 16-22
GOSUB, 22-8
GOSUBp, 23-1

GOTO, 22-8
GOTODC, 21-3
GOTOp, 23-1
GTEXT, 7-11
GTEXT+, 7-17
GTEXT1, 7-17
GTEXTM, 7-11
GTEXTX, 7-11
GTFLAG, 11-28
GTKYC+, 8-73
GTKYCD, 8-73
GTPTRS, 9-46
GTPTRX, 9-46
GTXT++, 7-17

H

HASH1, 8-6
HASH2, 8-6
HDFLT, 5-3
HEXASC, 5-5
HEXDEC, 5-9
HMSSEC, 25-12
HNDLFL, 15-20
HOWARD, 16-1
HTRAP, 15-19
HUGE, 15-5
HXDASC, 7-8
HXDCW, 5-9

I

I/OAL+, 3-6
I/OALL, 3-6
I/OCOL, 3-5
I/OCON, 3-4
I/ODAL, 3-8
I/OEX2, 3-7
I/OEXP, 3-7
I/OFND, 3-2
I/ORES, 3-3
IDIV, 15-25
IDIVA, 15-24
IF12A, 15-13
ILCNTe, 16-17
IMDO+2, 11-15
IMDO-2, 11-15
IMerr, 24-9
IMin01, 8-24
IMinit, 8-24

IMoffs, 8-28
IMxq27, 8-30
INBS=C, 18-13
INF^0, 15-10
INFR15, 15-13
INPOFF, 8-56
INPUTP, 23-7
INTGR, 26-5
INTR50, 11-31
INTRPT, 11-31
INVNaN, 13-6
IOCNDO, 3-4
IOFNDO, 3-2
IOFSCR, 3-1
ISRAM?, 4-1
IVAERR, 24-10
IVEXPe, 16-17
IVPARe, 16-17
IVVARe, 16-17

K

KEY$, 10-3
KEYCOD, 12-12
KEYDEL, 8-72
KEYFND, 8-71
KEYMRG, 9-60
KEYNAM, 8-76
KEYRD, 12-2
KEYSCN, 12-13
KYDN?, 4-16
KYFND+, 8-71
KYMRG+, 9-61

L

LABELP, 16-40
LABLDC, 7-21
LBLINP, 16-12
LBLNAM, 8-18
LBLNIF, 16-12
LCDINI, 6-20
LDCEXT, 7-9
LDCM10, 7-9
LDCOMP, 7-9
LDCSET, 18-9
LDSS11, 7-9
LDSS12, 7-9
LEAVE, 16-1
LEXBF+, 4-14

LEXBUF, 4-14
LGT15, 13-10
LIMITS, 8-5
LIN#A+, 7-13
LIN#AU, 7-13
LIN#CK, 7-13
LIN#D+, 7-13
LIN#DC, 7-13
LINEP, 16-6
LINEP+, 16-6
LINP, 16-12
LINPTP, 23-7
LINSKP, 8-67
LISTDC, 7-24
LN1+15, 13-7
LN1+XF, 13-7
LN12, 13-8
LN15, 13-8
LN30, 13-8
LNEP66, 16-6
LNPEXT, 16-6
LNSKP-, 8-67
LOCADR, 9-74
LOCFI#, 9-2
LOCFIL, 9-2
LSLEEP, 11-36
LSTLEN, 6-26
LXFND, 3-9

M

MAIN05, 24-11
MAIN30, 24-11
MAINLP, 24-11
MAKE1, 15-41
MAKEBF, 6-7
MEMBER, 11-3
MEMCK+, 18-11
MEMCKL, 18-11
MEMER*, 24-19
MEMERR, 24-18
MEMERX, 24-18
MESSG, 15-21
MFER42, 8-76
MFERR, 24-16
MFERR*, 11-51
MFERR-, 11-51
MFERRS, 24-17
MFERsp, 11-54
MFLG=0, 11-39
MFLG=X, 11-39

MFWRN, 11-47
MFWRNQ, 11-47
MFWRQ8, 11-47
MGOSUB, 8-45
MOVE*M, 11-59
MOVED0, 11-5
MOVED1, 11-5
MOVED2, 11-5
MOVED3, 11-5
MOVEDA, 11-5
MOVEDD, 11-5
MOVEDM, 11-5
MOVEU0, 11-7
MOVEU1, 11-7
MOVEU2, 11-7
MOVEU3, 11-7
MOVEU4, 11-7
MOVEUA, 11-7
MOVEUM, 11-7
MP1-12, 13-3
MP15S, 13-3
MP2-12, 13-3
MP2-15, 13-3
MPOP1N, 14-5
MPOP2N, 14-4
MPY, 15-26
MsgAvs, 8-78
MSN12, 15-38
MSN15, 15-38
MSPARe, 16-17
MTADDR, 2-2
MTADR+, 2-2
MULTF, 13-3
MVMEM, 11-60
MVMEM+, 11-60

N

NORDIM, 24-2
NOSCRL, 6-1
NRMCON, 15-2
NTOKEN, 16-1
NTOKNL, 16-1
NULLP, 9-28
NUM+0, 16-30
NUMC++, 16-30
NUMCK, 16-30
NUMK0, 16-30
NUMSCN, 16-5
NwOFFS, 11-17
NXTADR, 8-61

NXTELM, 8-62
NXTEXP, 8-43
NXTLIN, 9-18
NXTP, 16-26
NXTSTM, 8-68
NXTVA-, 8-59
NXTVAR, 8-59

O

OAGNXT, 16-22
OBCOLL, 18-13
OBEDIT, 9-8
OBLCMP, 18-13
OBPRD, 18-13
ONDC, 21-3
ONDC20, 21-3
ONERR, 22-3
ONP, 23-5
ONP40, 23-5
ONTIMR, 22-3
OPENF, 9-53
OPENF*, 9-53
OPENF-, 9-53
OPNF+, 9-53
ORGSB, 13-15
ORSB, 13-14
ORXM, 13-14
OUT1T+, 11-10
OUT1TK, 11-10
OUT2TC, 11-10
OUT2TK, 11-10
OUT3TC, 11-10
OUT3TK, 11-10
OUTBY+, 11-10
OUTBYT, 11-10
OUTC15, 11-46
OUTEL1, 7-18
OUTELA, 7-18
OUTEOL, 7-18
OUTLI1, 16-34
OUTLIT, 16-33
OUTNBC, 11-46
OUTNBS, 11-46
OUTNIB, 11-10
OUTRES, 8-4
OUTVAR, 16-35
OVFL, 15-17

## P

P1-10, 16-46
PARERR, 16-17
PART3, 22-18
pBSCen, 17-35
pBSCex, 17-37
pCALRS, 17-117
pCALSV, 17-115
pCAT, 17-140
pCAT$, 17-142
pCLDST, 17-76
pCMPLX, 17-1
pCONFG, 17-70
pCOPYx, 17-11
pCRDAB, 17-69
pCREAT, 17-105
pCRT=8, 17-107
pCURSR, 17-13
pDATLN, 17-112
PDEV, 9-64
PDEV+, 9-64
PDEV1, 9-64
pDEVCp, 17-27
pDIDST, 17-111
pDSWKY, 17-82
pDSWNK, 17-80
PEDIT, 9-15, 17-137
PEDITD, 9-15
PEDITM, 9-15
pENTER, 17-161
pEOFIL, 17-94
pERROR, 17-154
pExcpt, 17-39
pFASCH, 17-99
pFILDC, 17-139
pFILXQ, 17-124
pFINDF, 17-109
PFINDL, 9-26
PFNDL*, 9-26
PFNDZL, 9-26
pFNIN, 17-119
pFNOUT, 17-120
pFPROT, 17-135
pFSPCp, 17-24
pFSPCx, 17-126
pFTYPE, 17-98
PI/2, 15-44
PI/2D, 15-44
PI/4, 15-40

pIMbck, 17-48
pIMCHR, 17-44
pIMcpi, 17-51
pIMcpw, 17-62
pIMXCH, 17-58
pIMXQT, 17-55
pKYDF, 17-74
pLIST, 17-144
pLIST2, 17-146
pMEM, 17-157
pMERGE, 17-148
pMNLP, 17-77
pMRGE2, 17-150
POKEP, 23-12
POLL, 17-16
POLLD+, 17-16
POP1N, 14-2
POP1N+, 14-5
POP1R, 14-10
POP1S, 14-4
POP2N, 14-1
POP2N+, 14-4
POPBUF, 12-14
POPGSB, 11-43
POPMTH, 14-7
POPSTK, 11-43
POPSTR, 14-7
POPUPD, 11-43
pPARSE, 17-23
pPRGPR, 17-130
pPRINM, 17-96
pPRTCL, 17-89
pPRTIS, 17-87
pPURGE, 17-128
pPWROF, 17-78
pRCRD, 17-67
pRDCBF, 17-102
pRDNBF, 17-91
pREADM, 17-92
pREN, 17-114
PREP, 26-15
PRESCN, 16-1
PRGFMF, 9-70
PRINT*, 22-17
pRNAME, 17-132
PRNEXe, 16-17
PRNTDC, 21-2
PRPSND, 6-25
PRSCOO, 8-19
PRSCKB, 8-19
PRSCOP, 8-19
PRSsc+, 8-29

PRSscn, 8-29
PRTWDC, 7-25
pRTNTp, 17-121
PRTP, 23-10
pRUNft, 17-30
pRUNnB, 17-32
PSHGSB, 11-41
PSHMCR, 11-41
PSHSTK, 11-40
PSHSTL, 11-40
PSHUPD, 11-41
pSREC#, 17-100
pSREQ, 17-83
pTEST, 17-161
pTIMR#, 17-8
pTRANS, 17-163
pTRFMx, 17-3
PUGFIB, 9-57
PURGDC, 21-4
PURGEF, 9-3
PUTRES, 8-52
pVER$, 17-86
pWARN, 17-151
pWCRD, 17-65
pWCRD8, 17-64
pWRCBF, 17-104
PWROFF, 24-13
pWTKY, 17-72
pZERPG, 17-42

Q

QUOEXe, 16-17
QUOTCK, 11-38

R

R3=D1+, 11-13
R3=D10, 11-13
R3=D1C, 11-13
R<RSTK, 20-4
RAMROM, 9-73
RANGE, 11-2
RCCD1, 15-23
RCCD2, 15-23
RCL*, 15-34
RCLW1, 15-34
RCLW2, 15-34
RCLW3, 15-34
RCLW4, 15-34

RCSCR, 15-33
RCVOFS, 11-18
RDATTY, 24-14
RDBAS, 9-5
RDBYTA, 9-42
RDCHD+, 9-19
RDCHDR, 9-19
RDHDR1, 9-19
RDINFD, 19-1
RDINFO, 19-1
RDINFS, 19-1
RDLNAS, 9-41
RDLNFX, 9-41
RDTEXT, 9-6
READIN, 15-4
READNB, 9-7
READP, 23-7
READP5, 23-7
RECADR, 26-8
RECALL, 26-6
REDUCE, 15-1
RELJMP, 11-44
REMDC, 7-18
RENMDC, 21-4
RENSUB, 9-58
REPROM, 8-56
RESCAN, 16-1
RESPTR, 16-23
REST*, 16-19
RESTAR, 16-19
RESTOR, 22-8
RESTRP, 23-2
REV$, 14-6
REVPOP, 14-3
REWIND, 9-49
RFAD++, 18-8
RFAD+I, 18-8
RFAD--, 18-5
RFAD-I, 18-5
RFADJ+, 18-8
RFUPD+, 18-6
RJUST, 5-8
RLINFO, 20-8
RND-12, 15-7
RND12+, 15-17
RNDAHX, 15-27
RNDNRM, 15-18
ROMCHK, 11-21
ROMFND, 11-21
RPLLIN, 9-80
RPLSBH, 9-9
RPTKY, 12-6

RSTK<R, 20-3
RSTST, 15-4
RUNRT1, 24-3
RUNRTN, 24-3

## S

SALLOC, 19-3
SAVESB, 13-15
SAVEXM, 13-15
SAVGSB, 13-15
SB15S, 15-28
SCAN, 16-4
SCNRT, 6-18
SCOPCK, 8-75
SCRLLR, 12-10
SE1-10, 16-47
SECHMS, 25-11
SEND20, 8-49
SENDEL, 8-48
SENDIT, 8-49
SENDWD, 6-21
SETALM, 25-6
SETALR, 25-7
SETFMT, 6-4
SETIME, 25-2
SETSB, 13-14
SETTMO, 25-9
SFLAG?, 11-27
SFLAGC, 11-25
SFLAGS, 11-24
SFLAGT, 11-26
SHF10, 15-9
SHFLAC, 15-43
SHFRAC, 15-43
SHFRBD, 15-44
SHFTKN, 16-1
SHRT, 26-4
SIGCHK, 15-6
SIGTST, 15-31
SIN12, 13-17
SIN15, 13-17
SKIPDC, 7-23
SLEEP, 11-36
SMALL, 15-5
SNAPLC, 20-6
SNAPR*, 20-5
SNAPRS, 20-5
SNAPSV, 20-6
SNDWD+, 6-21
SPACE, 26-14

SPLITA, 15-11
SPLITC, 15-15
SPLTAC, 15-14
SPLTAX, 15-31
SQR15, 13-5
SQR17, 13-5
SQR70, 15-9
SQRSAV, 13-14
SRLEAS, 20-8
STAB1, 15-23
STAB2, 15-23
STATIC, 26-6
STATR+, 20-1
STATRS, 20-1
STATSV, 20-2
STCD2, 15-23
STKCH+, 15-35
STKCHR, 15-35
STKCMD, 11-1
STKVCT, 8-60
STMBCL, 8-74
STMBUF, 8-74
STOP, 22-5
STORE, 22-2
STR$00, 5-12
STR$SB, 5-12
STRASN, 26-1
STREQL, 11-8
STRGCK, 16-31
STRHDR, 8-47
STRHED, 8-63
STRNGP, 23-12
STRTST, 11-8
STSCR, 15-32
STUFF, 11-4
SUBONE, 13-1
SVINF+, 19-1
SVINFO, 19-1
SVTRC, 8-1
SWPBYT, 9-10
SYCOLL, 18-13
SYNTXe, 16-17

T

TAN12, 13-17
TAN15, 13-17
TBLJMC, 11-30
TBLJMP, 11-30
TBMSG$, 8-78
TBMSTX, 8-78

TFHDLR, 9-1
TKSCN+, 8-69
TKSCN4, 8-69
TKSCN7, 8-69
TODT, 25-10
TONE, 11-19
TRACDC, 7-18
TRC90, 13-18
TRFROM, 8-65
TRMNTR, 10-1
TRSFMu, 8-7
TRTO, 8-66
TRTO*, 8-66
TRTO+, 8-66
TRTO-, 8-66
TST12A, 15-36
TST15, 15-36
TstEnd, 8-39
TWO*, 15-43

U

UPCPOS, 9-45
UPDANN, 6-5
UPDANX, 6-5
uRES12, 15-16
uRESD1, 15-29
uRESNX, 15-16
uRESXT, 15-16
uRND>P, 15-17
USGch+, 8-32
USGch-, 8-32
USGnum, 8-35
USGrst, 8-34
USING, 22-13
USINGp, 23-10
USloop, 8-40
USnm05, 8-35
USst03, 8-31
USst05, 8-31
uTEST, 13-12

V

VAL00, 8-57
VARDC, 7-20
VARDC+, 7-20
VARNB-, 5-11
VARNBR, 5-11
VARP, 16-27

VARPOS, 16-27
VIEWD1, 6-2
VRIABL, 16-1

W

WFTMDT, 9-13
WIPOUT, 11-4
WRBYTC, 9-43
WRBYTD, 9-43
WRDSC+, 16-15
WRDSCN, 16-15
WRITNB, 9-7
WRTFIB, 9-54
WRTNUM, 9-40
WRTSTR, 9-39
WSTRFX, 9-38

X

XMTADR, 2-1
XXHEAD, 14-12
XYEX, 15-11

Y

YMDDAY, 25-13
YMDHO1, 25-8
YMDHMS, 25-8
YX2-12, 13-10
YX2-15, 13-10

Z

ZERBUF, 22-19

HEWLETT
PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route Du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

00071-90069 English

Printed in U.S.A. 2/84