```
*
* LANGUAGE SYSTEM GLOBALS
*
MACRO BOOL _ SCALAR
MACRO PROC _ SCALAR
MACRO REPEAT _ WHILE TRUE DO
MACRO ENDREPEAT _ ENDWHILE
MACRO UNLESS _ IF NOT
MACRO ENDUNLESS _ ENDIF
MACRO UNTIL _ WHILE NOT
MACRO ENDUNTIL _ ENDWHILE
MACRO MOD _ REM
CONSTANT IDLENGTH _ 60
STRING PROMPT [ 4 ]
MACRO MESSAGE(M) _ OUTLINE(M)
MACRO ERRMSG ( M ) _ MESSAGE ( M ) & ERRFLAG _ TRUE
BOOL ERRFLAG
*
CONSTANT TAILSIZE _ 1
STRING TEXTSEG _ "T"
STRING CODESEG _ "C"
STRING DEBUGSEG _ "D"
STRING DATASEG _ "$"
STRING CAPSEG _ ":"
*
CONSTANT PNAMEMAX _ 7
CONSTANT FNAMEMAX _ PNAMEMAX + TAILSIZE
*
BOOL LOADED
STRING PROGNAME [ PNAMEMAX ]
   STRING TEXTNAME [ FNAMEMAX ]
   STRING CODENAME [ FNAMEMAX ]
   STRING DEBUGNAME [ FNAMEMAX ]
   STRING DATANAME [ FNAMEMAX ]
   STRING CAPNAME [ FNAMEMAX ]
*
*  PRINT-SEGMENT SEGMENT NAME
*
   STRING PSNAME [ FNAMEMAX ]
*
*
BOOL RTLOADED
STRING RUNTNAME [ PNAMEMAX ]
   STRING RTEXTNAME [ FNAMEMAX ]
   STRING RCODENAME [ FNAMEMAX ]
   STRING RDEBUGNAME [ FNAMEMAX ]
*
MACRO NBLOCKS _ DIRSIZE(TXSEG) - 1
MACRO INVALID(BLK) _ NOGOOD(TXSEG,BLK)
MACRO RTINVALID(BLK) _ NOGOOD(RTXSEG,BLK)
*
*
CONSTANT LOCALLY _ 0, GLOBALLY _ 1
CONSTANT GLOBLK _ 1
```

```
STRING GLOBNAME _ "GLOBAL"
*
MACRO PCHECK _ UNLESS LOADED DO FRETURN
MACRO RCHECK _ UNLESS ( RUNFLAG = 1 OR RTLOADED ) DO FRETURN
MACRO GCHECK _ IF INVALID(GLOBLK) DO (ERRMSG("GLOBALLY INVALID&M")&RETUR
*
MACRO SCHECK _ UNLESS HAVE\STATE DO FRETURN
BOOL HAVE\STATE
MACRO PROCSEG(R) _ (IF (R) THEN RCDSEG ELSE CDSEG)
MACRO DBUGSEG(R) _ (IF (R) THEN RDBSEG ELSE DBSEG)
*
```

```
*
* STRING STUFF
*
MACRO CLEAR ( S ) _ SETS ( (S), 0, 0 )
CONSTANT BLANK _ ' ', BLANKS _ '&167', CRESC _ '&U', ..
         BLOT _ '$', BELL _ '&G', RUBOUT _ '&137',   ..
         CARRET _ '&M', NUL _ '&140'
CONSTANT NONCHAR _ -1
MACRO REAL ( CHAR ) _ ( (CHAR) # NONCHAR )
*
```

```
*
* MACHINE-DEPENDENT STUFF
*
CONSTANT CPW _ 4;     *CHARS/WORD
CONSTANT WPW _ 1;     *1 FOR SIMPLE, 2 FOR QSPL
CONSTANT INST1 _ 0;   * ADDR OF 1ST INST IN FUNCTION
                          * ( 0 FOR SIPU, 8 FOR APL )
*
CONSTANT BPW _ 2;     * INST BYTES PER WORD
                      * ( 2 FOR SIPU, 4 FOR APL )
CONSTANT LSZW _ 35;           * LINE-SIZE IN WORDS *
CONSTANT LSZ  _ LSZW * CPW;   * LINE SIZE IN CHARS *
*
STRING CSTRING [ LSZ ]
STRING TARGET  [ LSZ ]
*
FIELD LASTCHAR(0:24,31)
MACRO MORELINE (WORD)_ ((WORD)$LASTCHAR#CARRET)
*
* LINE POINTERS AND WINDOWS
*
FIELD SEGMENT ( 0 THRU 1 TO VECTOR ), BLOCK ( 2 )
*
*LINE POINTERS
*
FIELD LINE ( 3 ), STR ( 4 THRU 6 ), STARTX ( 7 ), ENDX ( 8 )
CONSTANT NPTRS _ 8, PTRSIZE _ 9
VECTOR PTRVEC [ NPTRS PTRSIZE WORD ]
REFERENCE CURRENT,SEARCH,IPTR1,IPTR2
CONSTANT NONAME _ -1
VECTOR PTRNAME [ NPTRS ] _ ..
   NONAME, NONAME, NONAME, NONAME, 'A', 'B', 'C', 'D'
MACRO PTRADDR(I) _ (@PTRVEC[I])
MACRO MAKENULL(P) _ P.BLOCK _ 0
MACRO NULL(P) _ ((P).BLOCK = 0)
MACRO SETPOINTER(P,Q) _ BCOPY(P,Q,PTRSIZE)
*
* WINDOWS
*
FIELD BARRAY ( 3 THRU 4 TO VECTOR ), BUFFER ( 5 )
MACRO BLENGTH ( W ) _ ( ( (W).BARRAY ) $ VDSIZE )
CONSTANT NWINS _ 4, WINSIZE _ 6
VECTOR WINVEC [ NWINS WINSIZE WORD ]
REFERENCE TXWINDOW,GDBWINDOW,CDBWINDOW,CCDWINDOW
MACRO WINADDR(I) _ (@WINVEC[I])
*
```

```
*
* TEXT BUFFER
*
SCALAR TSZ
VECTOR TARRAY
SCALAR TLENGTH, TLINES
*
* GLOBAL STATE VARIABLES
*
BOOL WRAP
SCALAR SCOPE

* DID THE USER PROGRAM JUST TRAP?

   BOOL USERTRAPPED
*
```

```
*
* EX-SIMULATED EXTERNAL-SEGMENT-FIELD MACHINERY
*
 MACRO XFIELD ( NAME, XWORD, XLBIT, XRBIT ) _ ..
     FIELD NAME ( XWORD: XLBIT, XRBIT )
*
*
*
 MACRO GETFIELD ( A , F ) _ A.F
 MACRO PUTFIELD ( A , F , X ) _ A.F _ X
*
* GENERAL-PURPOSE FIELDS FOR ACCESSING USER-PROG WORDS
*
* ADDRESS AND SIZE OF STATEWORD
*
  REFERENCE STATEWORD
   CONSTANT SW\SIZE _ 4
*
* SIPU STATEWORD FIELDS
*
  XFIELD ( SWSBASE, 0, 0, 15 );  XFIELD ( SWLTOP, 0, 16, 31 )
  XFIELD ( SWLBASE, 1, 0, 15 );  XFIELD ( SWFLAGS, 1, 16, 31 )
  XFIELD ( SWOPCODE, 2, 0, 15 ); XFIELD ( SWTRAPCLASS, 2, 16, 31 )
  XFIELD ( SWW1, 3, 0, 15 );     XFIELD ( SWW2, 3, 16, 31 )
*
* SIZE OF STACK FRAME
*
  CONSTANT FRSIZE _ 4
  MACRO FIND\FRAME ( F ) _ ..
          SETREFSIZE ( @DTSEG[(F)+(STATEWORD.SWSBASE)] , FRSIZE )
*
* SIPU STACK FRAME FIELDS
*
  XFIELD ( SFACC, 0, 0, 31 )
     XFIELD ( SFLACC, 0, 0, 15 ); XFIELD ( SFRACC, 0, 16, 31 )
   XFIELD ( SFEXACC, 1, 0, 31 )
      XFIELD ( SFLEXACC, 1, 0, 15 ); XFIELD ( SFREXACC, 1, 16, 31 )
  XFIELD ( SFPCTR, 2, 0, 15 ); XFIELD ( SFFBASE, 2, 16, 31 )
  XFIELD ( SFCALLDATA, 3, 0, 15 )
      XFIELD ( SFRUNT, 3, 0, 0 )
      XFIELD ( SFNOFAIL, 3, 1, 1 )
      XFIELD ( SFOPCALL, 3, 2, 6 )
      XFIELD ( SFBLKNO, 3, 7, 15 )
   XFIELD ( SFLASTLBASE, 3, 16, 31 )

   CONSTANT FCMARKER _ 177777B; * SPECIAL LASTLBASE AT BOTTOM OF STACK *

   REFERENCE ENV; * CURRENT ENVIRONMENT FRAME FOR DEBUGGING *



*
```

```
* COMPUTATION STACK
*
CONSTANT CSFSIZE _ 6;       * SIZE OF FRAME IN COMP STACK
CONSTANT MAXCOMPS _ 3;      * MAX NO OF COMPUTATIONS ALLOWED
VECTOR COMPSTACK [ MAXCOMPS CSFSIZE WORD ]
REFERENCE CSFR;                         * POINTER TO TOP CSF
SCALAR COMPLEVEL;                       * COMPUTATION LEVEL COUNTER
*
FIELD CSFFRAME ( 1 );       * BOTTOM FRAME OF COMP IN USER STACK
FIELD CSFPCTR ( 2 );        * PCTR SAVED FROM TOP FRAME
FIELD DOSAVEDCODE ( 3 );    * FLAG: MUST DO SAVED CODE TO RESUME?
FIELD IMMEDLINE ( 4 );      * LINE NO OF THE IMMED. STMT.
FIELD IMMEDTRAP ( 5 );      * ADDR OF TRAP BYTE ENDING IMMED STMT
FIELD CSFFLAGS ( 6 );       * FLAGS FROM STATEWORD
*
* DEBUGGER BLOCK HEADER
*
MACRO FIND\DBHDR ( A ) _ SETREFSIZE ( @A[0], 3 )
*
* DEBUGGER BLOCK AND LINE-TABLE FIELDS
*
FIELD LTEISAVE ( 0: 0,12 )
FIELD LTEFLAGS ( 0: 13,15 ), LTEBREAK ( 0: 13,13 ), ..
          LTESTEP  ( 0: 14,14 ), LTETSTEP ( 0: 15,15 )
FIELD LTECODEPTR ( 0: 16,31 )
CONSTANT LTENULL _ 0
*
* CURRENT LINE TABLE ENTRY
*
SCALAR LTE
*
* CURRENT SYMBOL-TABLE ENTRY
*
REFERENCE STE

VECTOR STEVEC [ 2 ]

MACRO ISFUNCTION ( STE ) _ ((STE).STYPE=SFUNC)




*
* FIELDS IN FUNCTION NUMBERS
*
FIELD FNRUNT ( 0: 16,16 ), FNBLK ( 0: 23,31 )
MACRO FNUMBER(RUNT,BLK) _ OR ( RUNT @ FNRUNT, BLK @ FNBLK )
MACRO STEFUNCNO ( E ) _ GETFN ( (E).SVAL1 )
MACRO GETFN ( X ) _ MASK\OUT ( FNJUNK, X )
FIELD FNJUNK ( 0: 17,22 )
MACRO MASK\OUT ( F, X ) _ AND ( (X), EOR ( (-1) @ F, -1 ) )
*
```

```
-----

        MACRO DM_MACRO
        DM DI_SCALAR
        DM DS_RING
        DM DV_VECTOR
        DM DC_CONSTANT
        DM DF_FIELD
        DM DR_REFERENCE
        DM DEQV_EQUIVALENCE
        DM LSH_SHIFT-
        DM RSH_SHIFT

        DF LEFT16(0:0,15)
        DF RIGHT16(0:16,31)
        DF RIGHT1(0:31,31)
*
```

```
        DC NCHWD_4;*NUMBER OF CHAR PER WORD
        DC NWDCON_1;*NUMBER OF WORDS PER CONSTANT


*DEBUGGER BLOCK DECLARATIONS

        DF VRGAPTR(0:0,15)
        DF VRGPTR(0:16,31)
        DF VRGLMT(1:0,15)
        DF STFREE(2:16,31)
        DF HCSIZE(1:16,31)
        DC HCBASE_3
        DF LTBASE(2:16,31)
        DF LTSIZE(2:0,15)


*SYMBOL TABLE ENTRY DECLARATIONS

        DC SDISP_2
        DF SCHAIN(0:16,31)
        DF STYPE(0:7,9)
        DC SCONST_1,SVAR_2,SMACRO_3,SOPER_4
        DC SLABEL_5,SFUNC_6,SVCONST_7
        DF SPD(0:0,0)
        DF SVTYPE(0:1,3)
        DF SRTYPE(0:4,6)
        DF SMACC(0:1,6)
        DC SSCALAR_4,SVECTOR_1,SREF_2,SDESC_3,SANY_7
        DF SLG(0:10,10)
        DC SG_0,SL_1
        DF SVAL(1:0,31)
        DF SVAL1(1:0,15)
        DF SVAL2(1:16,31)
        DF SVAL10(1:1,6)
        DF SVAL11(1:7,15)
        DF SVAL1S(1:0,0)
        DF SSIZE(0:11,15)
        DF SNAME1(2)
        DF SNAME2(3)
        DF SCVAL(2)
        DF SWD0(0)
        DF SPWD0(0:0,15)
*
```

```
*BUFFER ROUTINE DECLARATIONS

        DC NBUF_7;*NUMBER OF BUFFERS
        VECTOR BUFBASE[NBUF VECTORS];*VECTOR DESCRIPTORS FOR BUFFERS
        DM BUFSIZE(XX)_BUFBASE[XX]$VDSIZE
        VECTOR BUFSEG[NBUF VECTORS];*SEGMENT DESCRIPTOR OF BLOCK IN BUFFER
        VECTOR BUFBLK[NBUF];*BLOCK NUMBER OF BLOCK IN BUFFER
        VECTOR BUFREF[NBUF REFERENCES];*REFERENCES FOR BUFFERS
        DC GDBBUF_0,CDBBUF_1,CCDBUF_2,CTXBUF_3,TTBUF_4,TLBUF_5,TNBUF_6
        DEQV GDBBASE=BUFBASE[GDBBUF];*GLOBAL DEBUGGER BLOCK
        DM GDBSIZE_GDBBASE$VDSIZE
        DEQV GDBREF=BUFREF[GDBBUF]
        DEQV CDBBASE=BUFBASE[CDBBUF];*CURRENT DEBUGGER BLOCK
        DM CDBSIZE_CDBBASE$VDSIZE
        DEQV CDBREF=BUFREF[CDBBUF]
        DEQV CCDBASE=BUFBASE[CCDBUF];*CURRENT CODE BUFFER
        DM CCDSIZE_CCDBASE$VDSIZE
        DEQV CTXBASE=BUFBASE[CTXBUF];*CURRENT TEXT BUFFER
        DM CTXSIZE_CTXBASE$VDSIZE
        DEQV TT=BUFBASE[TTBUF];*TOKEN TABLE
        DM TTSIZE_TT$VDSIZE
        DEQV TLBASE=BUFBASE[TLBUF];*TEMP LINE TABLE
        DM TLSIZE_TLBASE$VDSIZE
        DEQV TNBASE=BUFBASE[TNBUF];*BLOCK STACK TABLE
        DM TNSIZE_TNBASE$VDSIZE



*RUNTIME FLAG

        DI RUNFLAG, MEASFLAG
*
```

```
* CAPABILITY SLOT ASSIGNMENTS *

        CONSTANT BIG _ 177777B;   * MAX SIZE VECTOR *
        MACRO EXT\VECTOR(V) _ ..
          CONSTANT EVTEMP _ ALLOCATE\CAPS(1) ! ..
          VECTOR V [ BIG EXTERNAL EVTEMP ]


       EXT\VECTOR ( RTXSEG );*RUNTIME TEXT SEGMENT
       EXT\VECTOR ( RDBSEG );*RUNTIME DEBUGGER SEGMENT
       EXT\VECTOR ( RCDSEG );*RUNTIME CODE SEGMENT

       EXT\VECTOR ( TXSEG );*USER TEXT SEGMENT
       EXT\VECTOR ( DBSEG );*USER DEBUGGER SEGMENT
       EXT\VECTOR ( CDSEG );*USER CODE SEGMENT

       EXT\VECTOR ( DTSEG );*DATA SEGMENT
       EXT\VECTOR ( CLSEG );*CLIST SEGMENT

       EXT\VECTOR ( PSSEG );*PRINT-SEGMENT SEGMENT

       EXT\VECTOR ( AUXTXSEG );* AUX TEXT SEGMENT
       EXT\VECTOR ( AUXDBSEG );* AUX DEBUGGER SEGMENT

       EXT\VECTOR ( CDSCOM );* COMPILER CODE SEGMENT
       EXT\VECTOR ( RCDSCOM );* COMPILER RUNTIME
       EXT\VECTOR ( DTSCOM );* COMPILER DATA SEGMENT
       EXT\VECTOR ( CLSCOM );* COMPILER CLIST SEGMENT

       CONSTANT PRCAP _ ALLOCATE\CAPS ( 1 )
       CONSTANT TCAP  _ ALLOCATE\CAPS ( 1 )
*
```

```
        DC DIRESIZE_2;*DIRECTORY ELEMENT SIZE


*SEGMENT INPUT-OUTPUT ROUTINES

        DM DBI(WW,XX,YY,ZZ)_VCOPY(ZZ,SUBVEC(WW,YY,XX),XX);*READ BLOCK
        DM DBO(WW,XX,YY,ZZ)_VCOPY(SUBVEC(WW,YY,XX),ZZ,XX);*WRITE BLOCK


*SEGMENT ACCESS ROUTINES

        DM SEGSIZE(XX)_XX[0]$RIGHT16
        DM SSEGSIZE(XX,YY)_(SET\LENGTH(XX$VDCAP,YY)&SEGSIZE(XX)_YY)
        DM DIRSIZE(XX)_XX[1]$RIGHT16
        DM BLKBASE(XX,YY)_XX[(YY) LSH 1]$LEFT16
        DM BLKHEAD(XX,YY)_XX[(YY) LSH 1]$RIGHT16
        DM BLKSIZE(XX,YY)_XX[((YY) LSH 1)+1]$LEFT16
        DM BLKINFO(XX,YY)_XX[((YY) LSH 1)+1]$RIGHT16
        DF HDRNFA(SIGNED 0:26,31);* -( WORDS OF FORMAL ARGUMENTS + 1 )
        DF HDRNL(0:16,23);*NUMBER OF WORDS OF LOCAL VARIABLES + 4
        DF BLKINVALID(0:31,31);*CODE FOR BLOCK INVALID
*
```

```
*
* GLOBALS FOR LINE COLLECTOR
*
 SCALAR LCINIT
 REFERENCE OLDLINE, NEWLINE
 STRING LINE1[LSZ], LINE2[LSZ], HOLD[LSZ]
 SCALAR CHAR, TCHAR, SCHAR
 SCALAR INSMODE
*
 SCALAR OLDTEMP, NEWTEMP
 MACRO OLDSAVE _ RPSAVE(OLDLINE,OLDTEMP)
 MACRO NEWSAVE _ WPSAVE(NEWLINE,NEWTEMP)
 MACRO OLDRESTORE _ RPRESTORE(OLDLINE,OLDTEMP)
 MACRO NEWRESTORE _ WPRESTORE(NEWLINE,NEWTEMP)
*
 MACRO RPRESET ( S ) _ S.SDFPTR _ 0
 MACRO RPSAVE ( S, T ) _ T _ S.SDFPTR
 MACRO WPSAVE ( S, T ) _ T _ S.SDRPTR
 MACRO RPRESTORE ( S, T ) _ S.SDFPTR _ T
 MACRO WPRESTORE ( S, T ) _ S.SDRPTR _ T
*
*
 REFERENCE EXCHTEMP
 MACRO EXCHANGE(A,B) _  EXCHTEMP _ A & A _ B & B _ EXCHTEMP
*
*
*
 VECTOR TABS [ LSZ 8 BIT ]
*
 MACRO DISTANCE(S) _ S.SDFPTR
 MACRO COMPLAIN _ COUT ( BELL )
 *
```

```
*
* FREE-STORAGE POOL
*
   VECTOR FREEPOOL [ 10000 ]
END
```

```
FUNCTION SED()

* PRODUCTION-MODE FRONT-END

   SCOPY ( PROMPT, ":" )
   RUN()
   RETURN
END
```

```
FUNCTION ACTIVE ( REFERENCE P )
*
* ACTIVE ACTIVATES A NON-NULL POINTER P, OR FAILS IF
*   P WAS NULL.
*
   IF NULL ( P ) DO
      RETURN FALSE
   ELSE DO
      SELWINDOW ( TXWINDOW , P.SEGMENT , P.BLOCK )
      RETURN TRUE
   ENDIF
END
```

```
FUNCTION ADDR\LENGTH ( VECTOR CD, PC )

*GIVEN CODE VECTOR 'CD' AND P-CTR OF OPERAND 'PC'
*RETURN LENGTH OF OPERAND IN HALF WORDS

FIELD ABITS ( 0 : 23, 27 ), BIT16 ( 0 : 16, 16 )

   IF CD[PC] $ BIT16 = 0 AND CD[PC] $ ABITS = 1 DO
      RETURN 2     ;*LONG FORM ADDRESS
   ELSE DO
      RETURN 1     ;*SHORT FORM ADDRESS
   ENDIF
END
```

```
FUNCTION ADVANCE ( PROC MOVE , REFERENCE POINTER , REFERENCE LIMIT )
*
* ADVANCE STEPS THE POINTER TO THE NEXT LINE, BUT
*  FAILS IF THE POINTER WOULD HAVE PASSED LIMIT
*  (I.E. = LIMIT ON ENTRY) OR IF MOVE PROC FAILS.
*
   IF POINTER.BLOCK = LIMIT.BLOCK AND ..
      POINTER.LINE  = LIMIT.LINE DO
         FRETURN
   ELSE DO
      MOVE ( POINTER :FRETURN)
      RETURN
   ENDIF
END
```

```
FUNCTION AFPROC()
*
* APPEND-FUNCTION COMMAND
*
    PCHECK
    *
    IF CMDMOVE ( IPTR1 :FRETURN ) DO
       APFUNC ( IPTR1 :GOTO EMPTY )
    ENDIF
    RETURN
    *
  EMPTY:
    ERRMSG ( "TEXT BUFFER EMPTY&M" )
    RETURN
END
```

```
FUNCTION ALPHABETIC ( SCALAR C )
   RETURN 'A' <= C AND C <= 'Z'
END
```

```
FUNCTION ALPHANUMERIC ( SCALAR C )
   RETURN ALPHABETIC ( C ) OR NUMERIC ( C )
END
```

```
FUNCTION AMATCH ( STRING TARGET, STRING SUBJECT )
*
* AMATCH LOOKS FOR TARGET ANYWHERE IN SUBJECT STRING.
*  (I.E.  ARB TARGET  )
*  IT RETURNS TRUE OR FALSE. ALL MULTIBLANK SUBSTRINGS
*  ARE CONSIDERED EQUIVALENT.
*
   RING TARG, SUBJ, ALIGN
   SCALAR TC, SC, TARGETPTR

   TARG _ TARGET . SDVD
   TARGETPTR _ TARGET . SDRS

   ALIGN _ SUBJ _ SUBJECT . SDVD
   ALIGNPTR _ SUBJECT . SDRS

   REPEAT
      SUBJPTR _ ALIGNPTR
      TARGPTR _ TARGETPTR

      * LEADING BLANK IN TARGET MAY MATCH MULT BLANKS IN SUBJECT *

      IF FRONT(TARG :-1)=BLANK AND FRONT(SUBJ :-1)=BLANKS DO
         PUTF ( TARG ) _ BLANKS
      ENDIF

      * MAIN SCAN LOOP *

      WHILE(TC_GETF(TARG:RETURN TRUE))=(SC_GETF(SUBJ:RETURN FALSE))DO
         IF TC = BLANKS DO ( GETF ( TARG ) & GETF ( SUBJ ) )
      ENDWHILE

      * TRAILING BLANK IN TARGET MAY MATCH MULT BLANKS IN SUBJECT *

      IF TC = BLANK AND SC = BLANKS DO FRONT ( TARG :RETURN TRUE )

      * ADVANCE ALIGNMENT OF TARGET WITH SUBJECT

      IF GETF ( ALIGN ) = BLANKS DO GETF ( ALIGN )

   ENDREPEAT
END
```

```
FUNCTION APFUNC ( REFERENCE POINTER )
*
* APFUNC CREATES A BLOCK IN THE TEXT FILE AFTER THE ONE
*  POINTED TO BY THE POINTER, AND FILLS IT WITH THE
*  CONTENTS OF THE TEXT BUFFER.. FAILS IF TEXT BUFFER EMPTY.
*
   REFERENCE P
   SCALAR NEWBLOCK , I
   VECTOR FBODY
   *
   UNLESS ( TLENGTH > 0 ) DO FRETURN
   *
   OUTWINDOW ( TXWINDOW )
   *
   NEWBLOCK _ POINTER.BLOCK + 1
   BLKCREATE ( TXSEG , NEWBLOCK )
   UPPTRS ( NEWBLOCK )
   FBODY _ SMAKE ( TLENGTH )
   VCOPY ( FBODY , TARRAY , TLENGTH )
   *
   NEWWINDOW ( TXWINDOW, TXSEG, NEWBLOCK, FBODY )
   NEWPOINTER ( CURRENT , NEWBLOCK , 1 , 0 )
   MAKEINVALID ( GLOBALLY )
   RETURN
END
```

```
FUNCTION ARITHPROC()
*
* ARITHMETIC TRAP
*
   MESSAGE ( "ARITHMETIC TRAP.&M" )
   RETURN
END
```

```
FUNCTION ATPROC()
*
* APPEND-TEXT COMMAND
*
    PCHECK
    *
    IF CMDMOVE ( IPTR1 :FRETURN ) DO
       INTBUF ( FALSE, IPTR1 )
       SETPOINTER ( CURRENT, IPTR1 )
    ENDIF
    RETURN
    *
END
```

```
FUNCTION ATTNPROC()
*
* PROCESS ATTN TRAP
*
   MESSAGE ( "ATTENTION TRAP.&M" )
   RETURN
END
```

```
FUNCTION AWAY\SCOM()

* GET RID OF COMPILER

   MACRO C(V) _ V$VDCAP
   MACRO PUSH(X) _ ((X)&CODE(4114B))
   MACRO TRAP(N) _ ((N)&CODE(4164B,44106B,1,4100B))

   CLOSE ( C(RCDSCOM) )
   CLOSE ( C(CDSCOM) )
   CLOSE ( C(DTSCOM) )

   PUSH ( 1 ); TRAP ( 28 )

   RETURN
END
```

```
FUNCTION BACK ( REFERENCE POINTER )
*
* BACK MOVES AN ACTIVE POINTER BACK ONE LINE. START-OF-FILE
*  OR START OF BLOCK ARE AS IN FWD.
*
   SCALAR EX, B, L
   *
   L _ POINTER.LINE - 1
   EX _ POINTER.STARTX - 1
   IF EX < 0 DO
      IF SCOPE = GLOBALLY DO
         B _ POINTER.BLOCK - 1
         IF B = 0 DO
            IF WRAP DO
               B _ NBLOCKS
            ELSE DO
               FRETURN
            ENDIF
         ENDIF
         POINTER.BLOCK _ B
         UNLESS ACTIVE ( POINTER ) DO CRASH()
      ELSE DO
         UNLESS WRAP DO FRETURN
      ENDIF
      L _ NLINES ( TXWINDOW )
      EX _ BLENGTH ( TXWINDOW ) - 1
   ENDIF
   POINTER.LINE _ L
   POINTER.ENDX _ EX
   *
   BKSPAN ( POINTER )
   RETURN
   *
END
```

```
FUNCTION BADINFO ( F )
*
* TEST VALIDITY OF DEBUGGER BLOCK INFO FOR FUNCTION F
*
   UNLESS ( F $ FNRUNT ) DO
      IF INVALID ( GLOBLK ) OR INVALID ( F $ FNBLK ) DO
         RETURN TRUE
      ENDIF
   ENDUNLESS
   RETURN FALSE
END
```

```
FUNCTION BADOPPROC()
*
* BAD OPCODE TRAP
*

FIELD OPFLD ( 0 : 18, 22 )

   SOUT ( "BAD OPCODE TRAP: " )
   IOUT ( STATEWORD . SWOPCODE $ OPFLD,,8 )
   CRLF ( 1 )
   RETURN
END
```

```
FUNCTION BCOPPROC()
*
* BAD-COMPLEX-OPERAND TRAP
*

STRING SIND _ "INDIRECTION&M"
STRING SFLD _ "FIELD OPERATION&M"
STRING SSUB _ "VECTOR OPERATION&M"
STRING SFUN _ "FUNCTION CALL&M"
STRING SRNG _ "RING OPERATION&M"
VECTOR MESG[8 STRINGS] _ SIND,SSUB,SRNG,SRNG,SFLD,SFLD,SFLD,SFUN
CONSTANT FRET _ 4100B
FIELD CMPLX ( 0 : 20, 22 )
REFERENCE TOP
SCALAR FN, PC
VECTOR CD

   IF FCDONE() DO
      STATEWORD . SWLTOP _ 0
      MESSAGE ( "DONE&M" )
   ELSEIF STATEWORD . SWOPCODE = FRET DO
      MESSAGE ( "ILLEGAL FAIL RETURN&M" )
   ELSE DO
*GET P-CTR FOR TRAPPED INSTRUCTION
      TOP _ FIND\FRAME ( STATEWORD . SWLBASE )
      FN _ GETFN ( TOP . SFCALLDATA )
      CD _ PROCSEG ( FN $ FNRUNT )
      CD _ HALFVEC ( SUBVEC ( CD, BLKBASE(CD, FN$FNBLK) ) )
      PC _ FIND\INSTR ( CD, LTE$LTECODEPTR, TOP . SFPCTR )
*GET P-CTR OF COMPLEX OPERAND OF INSTRUCTION
      PC _ PC + ADDR\LENGTH ( CD, PC )
*PRINT MESSAGE FOR APPROPRIATE COMPLEX ADDRESSING TYPE
      SOUT ( "ILLEGAL " )
      MESSAGE ( MESG [ CD[PC] $ CMPLX ] )
   ENDIF
   RETURN
END
```

```
FUNCTION BKPTPROC()
*
* PROCESS BKPT TRAP
*
   REFERENCE TOP
   *
   TOP _ FIND\FRAME ( STATEWORD . SWLBASE )
   IF LTE $ LTEBREAK DO
      CSFR.DOSAVEDCODE _ TRUE
      MESSAGE ( "BREAK&M" )
   ELSEIF LTE $ LTESTEP DO
      CSFR.DOSAVEDCODE _ TRUE
      MESSAGE ( "STEP&M" )
   ELSEIF LTE $ LTETSTEP DO
      CRASH();    ***NOT YET IMPLEMENTED***
   ELSEIF TOP.SFPCTR = CSFR.CSFPCTR DO
      CRASH()
   ELSE DO
      MESSAGE ( "STATIC BREAK&M" )
   ENDIF
   *
   RETURN
END
```

```
FUNCTION BKSPAN ( REFERENCE POINTER )
*
* BKSPAN TAKES AN ACTIVE POINTER WITH ENDX SET, AND SCANS
*  TO THE START OF THE LINE, SETTING STARTX AND STR
*
   VECTOR A
   SCALAR I
   *
   A _ TXWINDOW.BARRAY
   I _ POINTER.ENDX - 1
   WHILE I >= 0 AND MORELINE ( A [ I ] ) DO
      I _ I - 1
   ENDWHILE
   POINTER.STARTX _ I + 1
   MAKESTRING ( @(POINTER.STR), A, I+1, POINTER.ENDX )
   RETURN
END
```

```
FUNCTION BLKCREATE(VECTOR SEG,BLK)

*CREATE BLOCK BLK IN SEGMENT SEG MOVING REST OF DIRECTORY DOWN

*EXPAND DIRECTORY IF FULL
       IF BLKSIZE(SEG,0) >= BLKSPACE(SEG,0) DO
          BLKEXPAND(SEG,0,240,0)
       ENDIF
*MOVE DIRECTORY FROM BLK UP ONE ENTRY
       FOR I _ DIRSIZE(SEG)-1 BY -1 TO BLK DO
          SEG[2*(I+1)] _ SEG[2*I]
          SEG[2*(I+1)+1] _ SEG[2*I+1]
       ENDFOR
*SET UP NEW BLOCK ENTRY IN DIRECTORY
       BLKSIZE(SEG,BLK)_0
       IF BLK >= DIRSIZE(SEG) THEN   ..
                        BLKBASE(SEG,BLK) _ SEGSIZE(SEG)
*INCREMENT DIRECTORY SIZE
       DIRSIZE(SEG) _ DIRSIZE(SEG) + 1
       BLKSIZE(SEG,0) _ BLKSIZE(SEG,0)+DIRESIZE
*UPDATE BUFFER BLOCK NUMBERS
       FOR I _ NBUF-1 BY -1 TO 0 DO
          IF BUFSIZE(I) # 0  ..
                    AND BUFSEG[I]$VDCAP = SEG$VDCAP  ..
                       AND BUFBLK[I] > BLK DO
             INC(BUFBLK[I])
          ENDIF
       ENDFOR
       RETURN
END
```

```
FUNCTION BLKDESTROY(VECTOR SEG,BLK)

*DESTROY BLOCK BLK OF SEGMENT SEG AND COMPRESS DRECTORY

*IF LAST BLOCK IN SEGMENT REDUCE SEGMENT SIZE
       IF BLK+1 = DIRSIZE(SEG) DO
          SSEGSIZE ( SEG, BLKBASE(SEG,BLK) )
       ENDIF
*REDUCE DIRECTORY SIZE
       DIRSIZE(SEG) _ DIRSIZE(SEG)-1
       BLKSIZE(SEG,0) _ BLKSIZE(SEG,0)-DIRESIZE
*COMPRESS DIRECTORY
       FOR I _ BLK TO DIRSIZE(SEG)-1 DO
          SEG[2*I] _ SEG[2*(I+1)]
          SEG[2*I+1] _ SEG[2*(I+1)+1]
       ENDFOR
*UPDATE BUFFER BLOCK BUMBERS
       FOR I _ NBUF-1 BY -1 TO 0 DO
          IF BUFSIZE(I) # 0  ..
                          AND BUFSEG[I]$VDCAP = SEG$VDCAP  ..
                         AND BUFBLK[I] >= BLK DO
             IF BUFBLK[I] = BLK DO
                BUFCLEAR(I)
             ELSE DO
                DEC(BUFBLK[I])
             ENDIF
          ENDIF
       ENDFOR
       RETURN
END
```

```
FUNCTION BLKEXPAND(VECTOR SEG, BLK, SIZE, FLAG)

*EXPAND BY SIZE WORDS THE SPACE ALLOCATED TO BLOCK BLK OF SEGMENT SEG
*IF FLAG = 1 THEN CONTENTS TO END OF EXPANDED BLOCK

DI NBLK,T,SLOP,S

   NBLK _ DIRSIZE(SEG) - 1
*SCAN FOR SUFFICIENT SLOP IN LATER BLOCKS
   SLOP _ 0
   FOR I _ BLK+1 TO NBLK DO
      SLOP _ SLOP + BLKSPACE(SEG,I) - BLKSIZE(SEG,I)
      IF SLOP >= SIZE THEN EXIT
   ENDFOR
   IF SLOP < SIZE DO
*EXPAND SEGMENT IF NOT ENOUGH SLOP
      SLOP_SEGSIZE(SEG)+SIZE-SLOP
      SSEGSIZE(SEG,SLOP)
      I _ NBLK
   ELSE DO
      SLOP _ BLKBASE(SEG,I) + BLKSPACE(SEG,I) + SIZE - SLOP
   ENDIF
*SLOP NOW IS INDEX OF END OF CURRENT BLOCK AFTER IT IS MOVED UP
*MOVE UP BLOCKS
   T _ BLKBASE(SEG,BLK)
   FLAG _ BLK + (NOT FLAG)
   FOR I _ I BY -1 TO FLAG DO
      S _ BLKSIZE(SEG,I)
      VCOPY(SUBVEC(SEG,SLOP-S),SUBVEC(SEG,BLKBASE(SEG,I)),S)
      BLKBASE(SEG,I) _ SLOP _ SLOP - S
   ENDFOR
   BLKBASE(SEG,BLK) _ T
   RETURN
END
```

```
FUNCTION BLKSPACE(VECTOR SEG,I)

*RETURNS SIZE OF SPACE ALLOCATED FOR BLOCK I OF SEGMENT SEG

       RETURN (IF I+1 # DIRSIZE(SEG)  ..
                     THEN BLKBASE(SEG,(I+1))  ..
                     ELSE SEGSIZE(SEG)) - BLKBASE(SEG,I)
END
```

```
FUNCTION BUFCLEAR(BUF)

*CLEAR BUFFER BUF

        SCALAR OLDSIZE

        OLDSIZE _ BUFSIZE(BUF)
        BUFSIZE(BUF) _ 0
        IF OLDSIZE #0 DO
*FREE UP STORAGE ONLY IF NO OTHER BUFFER IS USING THE SAME STORAGE
           FOR I _ 0 TO NBUF-1 DO
              IF BUFBASE[BUF]$VDBASE = BUFBASE[I]$VDBASE  ..
                                 AND BUF # I AND BUFSIZE ( I ) # 0 DO
                 RETURN
              ENDIF
           ENDFOR
           SFREE(FULLVEC(BUFBASE[BUF]))
        ENDIF
        RETURN
END
```

```
FUNCTION BUFCOPY(BUFA,BUFB)

*MAKE CONTENTS OF BUFFER BUFA SAME AS BUFB
*AFTER WRITING AND CLEARING BUFFER BUFA

        BUFWRITE(BUFA,0)
        BUFCLEAR(BUFA)
        BUFBASE[BUFA] _ BUFBASE[BUFB]
        BUFSEG[BUFA] _ BUFSEG[BUFB]
        BUFBLK[BUFA] _ BUFBLK[BUFB]
        BUFREF[BUFA] _ BUFREF[BUFB]
        RETURN
END
```

```
FUNCTION BUFFLUSH()

*WRITE AND CLEAR ALL BUFFERS

        FOR I _ 0 TO NBUF-1 DO
           BUFWRITE(I,0)
           BUFCLEAR(I)
        ENDFOR
        RETURN
END
```

```
FUNCTION BUFINIT()

*INITIALIZE ALL BUFFERS

        FOR I _ 0 TO NBUF-1 DO
           BUFSIZE(I) _ 0
        ENDFOR
        RETURN
END
```

```
FUNCTION BUFLOAD(VECTOR SEG,BLK,BUF,SLOP)

*LOAD BUFFER BUF FROM BLOCK BLK OF SEGMENT SEG

        IF BUFSIZE(BUF) # 0 AND BUFSEG[BUF]$VDCAP = SEG$VDCAP  ..
                AND BUFBLK[BUF] = BLK THEN RETURN
        BUFCLEAR(BUF)
        BUFREAD(SEG,BLK,BUF,SLOP)
        RETURN
END
```

```
FUNCTION BUFREAD(VECTOR SEG,BLK,BUF,SLOP)

*LOAD BUFFER BUF FROM BLOCK BLK OF SEGMENT SEG
*WITH SLOP WORDS OF SLOP

        DI S
        S _ BLKSIZE(SEG,BLK)
*DON'T DO READ IF BLOCK SIZE IS ZERO
        IF S = 0 THEN RETURN
*COPY BLOCK TO BUFFER
        DBI(SEG,S,BLKBASE(SEG,BLK),  ..
           BUFBASE[BUF] _ SMAKE(S+SLOP))
        BUFBLK[BUF] _ BLK
        BUFSEG[BUF] _ SEG
        BUFREF[BUF] _ REFVEC(BUFBASE[BUF])
        RETURN
END
```

```
FUNCTION BUFSET(VECTOR SEG,BLK,BUF,VECTOR SPACE)

*WORDS IN SPACE ARE PUT IN BUFFER BUF AS BLOCK BLK
*OF SEGMENT SEG AFTER PREVIOUS CONTENTS ARE UNLOADED AND CLEARED

        BUFWRITE(BUF,0)
        BUFCLEAR(BUF)
        BUFBASE[BUF] _ SPACE
        BUFSEG[BUF] _ SEG
        BUFBLK[BUF] _ BLK
        BUFREF[BUF] _ REFVEC(SPACE)
        RETURN
END
```

```
FUNCTION BUFSL(VECTOR SEG,BLK,BUF,SLOP)

*UNLOAD OLD BUFFER AND LOAD BUF WITH BLOCK BLK OF SEG SEG

        BUFWRITE(BUF,0)
        BUFLOAD(SEG,BLK,BUF,SLOP)
        RETURN
END
```

```
FUNCTION BUFWRITE(BUF,IDX)

*UNLOAD BUFFER BUF STARTING AT WORD IDX OF BLOCK IN SEGMENT

        DI B,SIZ,T
        DV S,VD
        B _ BUFBLK[BUF]
        S _ BUFSEG[BUF]
        IF BUFSIZE(BUF) = 0 OR B = 0 THEN RETURN
        VD _ FULLVEC(BUFBASE[BUF])
        SIZ _ VD$VDSIZE + IDX
*EXPAND BLOCK SPACE IF BUFFER CONTENTS WON'T FIT
        IF (T _ SIZ - BLKSPACE(S,B)) > 0 DO
           BLKEXPAND(S,B,T,0)
        ENDIF
*COPY BUFFER TO BLOCK
        DBO(S,VD$VDSIZE,BLKBASE(S,B)+IDX,VD)
        BLKSIZE(S,B) _ SIZ
        RETURN
END
```

```
FUNCTION CBPROC()
*
* CLEAR-BREAKPOINT COMMAND
*
*       NOTE: DOES NOT FIX SOURCE
*             DOES NOT HANDLE MULT LINES
*
   UNLESS CMDMOVE ( IPTR1 :FRETURN ) DO RETURN
   *
   LTEGET ( DBSEG, IPTR1.BLOCK, IPTR1.LINE :FRETURN )
   IF LTE $ LTEBREAK DO
      LTE $ LTEBREAK _ FALSE
      OUTBREAK ( IPTR1.BLOCK, LTE )
      LTEPUT ( DBSEG, IPTR1.BLOCK, IPTR1.LINE :CRASH() )
   ELSE DO
      ERRMSG ( "NO BREAKPOINT SET&M" )
   ENDIF
   RETURN
END
```

```
FUNCTION CLRWINDOW ( REFERENCE W )
*
* SETS WINDOW W TO EMPTY
*
   BUFCLEAR ( W.BUFFER )
   W.BLOCK _ 0
   RETURN
END
```

```
FUNCTION CMDARG ( STRING S )

* GET NEXT ARGUMENT FROM CURRENT COMMAND INTO 'S'

   ITEM ( S, CSTRING, ' ' )
   RETURN
END
```

```
FUNCTION CMDFN()
*
* FINDS USER PROGRAM FUNCTION NAMED BY NEXT PARAMETER
*
   SCALAR FNUM
   REFERENCE ENTRY
   STRING FNAME [ IDLENGTH ]
   *
   CMDARG ( FNAME )
   FNUM _ FNFIND ( FNAME :FRETURN )
   IF FNUM < 0 DO
      FRETURN
   ELSE DO
      RETURN FNUM
   ENDIF
END
```

```
FUNCTION CMDMOVE ( REFERENCE P )
*
* MOVE POINTER P TO NEXT LINE ADDRESS IN CURRENT COMMAND.
*
   STRING S [ LSZ ]
   *
   CMDARG ( S )
   RETURN MVPTR ( P, S :FRETURN )
   *
END
```

```
FUNCTION CMDPTR() RETURNING REFERENCE
*
* CMDPTR STRIPS THE NEXT ARGUMENT FROM THE CURRENT
*  COMMAND AND RETURNS A REFERENCE TO A LINE POINTER, OR FAILS.
*
   SCALAR CHAR, I
   *
   CHAR _ GNBCI ( CSTRING :RETURN PTRADDR ( 0 ) )
   FOR I _ 0 TO NPTRS-1 DO
      IF CHAR = PTRNAME [ I ] DO
         RETURN PTRADDR ( I )
      ENDIF
   ENDFOR
   FRETURN
END
```

```
FUNCTION COMMANDS()
*
* EXCOM COLLECTS A TTY LINE AND DECODES IT AS A COMMAND.
*  IT THEN CALLS THE APPROPRIATE ROUTINE IN THE
*  EDITOR, COMPILER, OR DEBUGGER.
*
*     VERSION 1.0
*
   SCALAR I, CMD
   BOOL SUCCESS
   STRING CMDBUF [ LSZ ], CMDLINE [ LSZ ]
   CONSTANT NCMDS _ 31
   *
   * COMMAND TABLES
   *
   VECTOR CMDS [ NCMDS ] _ ..
      'NP', 'SP', 'DP', 'ET', 'EX', ..
      'MT', 'TT', 'AF', 'AT', 'IT', ..
      'DT', 'LT', 'MV', 'RF', 'CP', ..
      'RT', 'ST', 'SR', 'RS', 'PC', ..
      'SB', 'CB', 'PD', 'SD', 'DS', ..
      'DO', 'GO', 'PV', 'IN', 'WH', ..
      'PS'
   *
   VECTOR CMDPROC [ NCMDS ] _ ..
      NPPROC, SPPROC, DPPROC, ETPROC, EXPROC, ..
      MTPROC, TTPROC, AFPROC, ATPROC, ITPROC, ..
      DTPROC, LTPROC, MVPROC, RFPROC, CPPROC, ..
      RTPROC, STPROC, SRPROC, RSPROC, PCPROC, ..
      SBPROC, CBPROC, PDPROC, SDPROC, DSPROC, ..
      DOPROC, GOPROC, PVPROC, INPROC, WHPROC, ..
      PSPROC
*
```

```
    CRLF ( 1 )
    SCOPY ( CMDBUF, "&M" )
    CLEAR ( CMDLINE )
    *
    * COMMAND LOOP
    *
    REPEAT
       SUCCESS _ FALSE
       IF LENGTH ( CMDLINE ) = 0 DO
          SOUT ( PROMPT )
          EDITLINE ( CMDBUF :GOTO  BADCOM )
          UNPOLISH ( CMDBUF, CMDLINE )
       ENDIF
       PEEL ( CMDLINE )
       *
       CMD _ 0
       FOR I _ 1 TO 2 DO
          CMD _ OR ( CMD LSH 8, GCI ( CSTRING :GOTO BADCOM ) )
       ENDFOR
       FOR I _ 0 TO NCMDS - 1 DO
          IF CMD = CMDS [ I ] DO
             ( CMDPROC [ I ] ) ( :GOTO BADCOM )
             SUCCESS _ TRUE
          ENDIF
       ENDFOR
       OUTWINDOW ( TXWINDOW )
       IF CMD = 'EX' DO RETURN
       IF USERTRAPPED DO TRAPS()
     BADCOM:
       IF NOT SUCCESS DO
          ERRMSG ( "INVALID COMMAND&M" )
       ENDIF
       IF ERRFLAG DO CLEAR ( CMDLINE )
       ERRFLAG _ FALSE
    ENDREPEAT
END
```

```
FUNCTION CPPROC()

* COMPILE-PROGRAM COMMAND: RUN FUNCTION CMPCTL IN COMPILER

   SETUP\SCOM()
   SPUSH ( RUNFLAG )
   SPUSH ( MEASFLAG )
   SPUSH ( 0 )
   RUN\SCOM ( 3 )
   AWAY\SCOM()
   RETURN
END
```

```
-----


FUNCTION CRASH()
*
* FATAL ERROR
*
   MESSAGE ( "CRASH!&M" ); CODE (4170B)
END
```

```
FUNCTION DOPROC()
*
* DO-FUNCTION COMMAND
*
   MACRO C(V) _ V $ VDCAP
   SCALAR FN, DATASIZE, CAPSIZE, STSIZE
   BOOL FOUND
   *
   PCHECK; RCHECK; GCHECK

   FOUND _ TRUE
   STSIZE _ GCONST ( "\STACKSIZE" :FOUND _ FALSE )
   DATASIZE _ STSIZE + BLKINFO ( CDSEG, GLOBLK )
   CAPSIZE  _ 2 * GCONST ( "\CLISTSIZE" :FOUND _ FALSE )

   IF NOT FOUND DO
      ERRMSG ( "\..SIZE SYMBOL MISSING&M" )
      RETURN
   ENDIF

   NEW\SEGMENT ( DATANAME, DATASIZE )
   NEW\SEGMENT ( CAPNAME, CAPSIZE )
   OPEN\WINDOWED ( C(DTSEG), DATANAME )
   OPEN\WINDOWED ( C(CLSEG), CAPNAME )

   FOR I _ 0 TO CAPSIZE-1 DO
      CLSEG [ I ] _ 0
   ENDFOR

   DTINIT()
   *
   FN _ CMDFN ( :FRETURN )
   FACPUSH ( FN )
   *
   * RUN THE FUNCTION
   *
KLUDGE\LOCK()
   GOUSER()
KLUDGE\UNLOCK()
   *
   USERTRAPPED _ TRUE
   HAVE\STATE _ TRUE
   *
   RETURN
END
```

```
FUNCTION DOWNPTRS ( SCALAR B )
*
* DOWNPTRS ADJUSTS PTRS DUE TO DELETION
*   OF FUNCTION BLOCK B.
*
   REFERENCE P
   SCALAR I
   *
   FOR I _ 0 TO NPTRS-1 DO
      P _ PTRADDR ( I )
      IF P.BLOCK = B DO
         MAKENULL ( P )
      ELSEIF P.BLOCK > B DO
         P.BLOCK _ P.BLOCK - 1
      ENDIF
   ENDFOR
   RETURN
END
```

```
FUNCTION DPPROC()
*
* DELETE-PROGRAM COMMAND
*
    PCHECK
    DROP()

    DELETE\SEGMENT ( TEXTNAME :FRETURN )
    DELETE\SEGMENT ( CODENAME :FRETURN )
    DELETE\SEGMENT ( DEBUGNAME :FRETURN )
    RETURN
END
```

```
FUNCTION DROP()
*
* DROP CLOSES ALL SEGMENTS OF THE CURRENT PROGRAM (IF ANY)
*  AND SETS ALL POINTERS TO NULL.
*
   MACRO C(V) _ V $ VDCAP
   SCALAR I
   REFERENCE P
   *
   CLEAR ( PROGNAME )
   *
   FOR I _ 0 TO NWINS-1 DO
      OUTWINDOW ( WINADDR ( I ) )
   ENDFOR
   *
   IF LOADED DO
      CLOSE ( C(TXSEG) )
      CLOSE ( C(CDSEG) : )
      CLOSE ( C(DBSEG) : )
      CLOSE ( C(DTSEG) : )
      CLOSE ( C(CLSEG) : )
   ENDIF
   *
   FOR I _ 0 TO NPTRS-1 DO
      MAKENULL ( PTRADDR (I) )
   ENDFOR
   *
   LOADED _ FALSE
   RETURN
END
```

```
FUNCTION DSPROC()
*
* DISPLAY-STACK COMMAND
*
   REFERENCE FRAME
   SCALAR CD, FN, PC, LN, FRLOC, PREVFRLOC
   BOOL SHOWPC

   SCHECK
   SHOWPC _ GNBCI ( CSTRING :0 ) = 'P'
   CRLF ( 1 )
   IF NOSTACK() DO RETURN
   FRLOC _ STATEWORD . SWLBASE
   UNTIL ( FRLOC = FCMARKER ) DO
      FRAME _ FIND\FRAME ( FRLOC )
      PREVFRLOC _ GETFIELD ( FRAME, SFLASTLBASE )
      FN _ GETFN ( FRAME . SFCALLDATA )
      IF BADINFO ( FN ) DO
         SOUT ( "INVALID FUNCTION" )
      ELSE DO
         PC _ GETFIELD ( FRAME, SFPCTR )
         LN _ LTEFIND ( FN, PC-1 :CRASH() )
         PLADDR ( FN, LN )
      ENDIF

      IF SHOWPC DO
         COUT ( '(' ); IOUT ( PC,,8 ); COUT ( ')' )
      ENDIF

      FRLOC _ PREVFRLOC
      CRLF ( 1 )
   ENDUNTIL
   CRLF ( 1 )
   RETURN
END
```

```
FUNCTION DTINIT()
*
* INITIALIZE USER PROCESS DATA SEGMENT
*
   STATEWORD . SWLTOP _ 0
   STATEWORD . SWSBASE _ BLKINFO ( CDSEG, GLOBLK )
   *
   RETURN
END
```

```
FUNCTION DTPROC()
*
* DELETE-TEXT COMMAND
*
    PCHECK
    *
    XTPROC ( OUTTBUF :FRETURN )
    RETURN
END
```

```
FUNCTION ETPROC()
*
* ETPROC ENTERS TEXT FROM THE TERMINAL INTO THE TEXT BUFFER.
*  QUITS WHEN USER TYPES CR-ESCAPE OR TEXT BUFFER IS FULL.
*
   TBFILL ( INLINE :FRETURN )
   RETURN
END
```

```
FUNCTION EXPROC()
*
* EXIT FROM LANGUAGE SYSTEM COMMAND
*
   DROP()
   RTDROP()
   MESSAGE ( "GOOD DAY&M" )
   RETURN
END
```

```
FUNCTION FACPUSH ( SCALAR FN )
*
* PUSHES AN ACTIVATION OF FUNCTION NUMBER 'FN' ONTO USER STACK.
*   ( ARGUMENTS MUST ALREADY BE ON STACK )
*
   SCALAR FBASE, LBASE, NWL, NWA, HDR, FLAGS
   CONSTANT FLSET  _ 003000B;      * SIPU FLAGS TO SET *
   CONSTANT FLKEEP _ 010757B;      * SIPU FLAGS TO KEEP UNCHANGED *
   CONSTANT FLRUNT _ 100000B;      * SIPU RUNTIME FLAG *
   *
   LBASE _ STATEWORD . SWLTOP
   HDR _ BLKHEAD ( PROCSEG ( FN $ FNRUNT ), FN $ FNBLK )
   NWL _ HDR $ HDRNL - FRSIZE
   NWA _ - ( HDR $ HDRNFA + 1 )
   FBASE _ LBASE - NWA
   *
   * ACCUMULATOR, EXTENDED ACCUMULATOR
   *
   PUSHWORD ( 0 )
   PUSHWORD ( 0 )
   *
   * P-COUNTER AND FRAME BASE
   *
   PUSHWORD ( INST1 @ LHALF + FBASE @ RHALF )
   *
   * CALL DATA AND LAST LBASE
   *
   PUSHWORD ( FN @ LHALF + FCMARKER @ RHALF )
   *
   * POINT LBASE TO FRAME DESCRIPTOR
   *
   PUTFIELD ( STATEWORD, SWLBASE, LBASE )
   *
   * LEAVE ROOM FOR LOCAL VARIABLES ABOVE STACK FRAME
   *
   STATEWORD . SWLTOP _ STATEWORD . SWLTOP + NWL
   *
   * SET FLAGS IN SIPU STATEWORD FOR FORCED CALL
   *
   FLAGS _ GETFIELD ( STATEWORD, SWFLAGS )
   FLAGS _ AND ( FLAGS, FLKEEP )
   FLAGS _ OR  ( FLAGS, FLSET  )
   IF FN $ FNRUNT DO
      FLAGS _ OR ( FLAGS, FLRUNT )
   ENDIF
   PUTFIELD ( STATEWORD, SWFLAGS, FLAGS )
   *
   RETURN
END
```

```
FUNCTION FCDONE()
*
* CHECK TRAP FOR VALID COMPLETION OF FORCED CALL
*
   SCALAR LASTLB, OPCODE
   REFERENCE TOP
   CONSTANT RET _ 4102B, BCOPTRAP _ 9
   *
   IF STATEWORD . SWTRAPCLASS = BCOPTRAP DO
      TOP _ FIND\FRAME ( STATEWORD . SWLBASE )
      LASTLB _ TOP . SFLASTLBASE
      OPCODE _ STATEWORD . SWOPCODE
      IF LASTLB = FCMARKER DO
         IF OPCODE = RET DO
            RETURN TRUE
         ENDIF
      ENDIF
   ENDIF
   RETURN FALSE
END
```

```
FUNCTION FIND\ACT ( FN, ACTN ) RETURNING REFERENCE

* FIND SPECIFIED FUNCTION ACTIVATION IN THE STACK
*  SCAN DOWN STACK FROM TOP TO 'ACTN'-TH ACTIVATION
*  OF FUNCTION 'FN' AND RETURN REFERENCE TO FRAME.

* FAILS: IF NO SUCH ACTIVATION

   SCALAR FADDR, ACTR
   REFERENCE FRAME

   FADDR _ STATEWORD . SWLBASE
   ACTR _ ACTN

   WHILE ACTR > 0 DO
      FRAME _ FIND\FRAME ( FADDR )
      IF FN = GETFN ( FRAME . SFCALLDATA ) DO
         ACTR _ ACTR - 1
      ENDIF
      FADDR _ FRAME . SFLASTLBASE
      IF FADDR = FCMARKER AND ACTR > 0 DO
         ERRMSG ( "FUNCTION ACTIVATION NOT FOUND&M" )
         FRETURN
      ENDIF
   ENDWHILE
   RETURN FRAME
END
```

```
FUNCTION FIND\INSTR ( VECTOR CD, SPC, EPC )

*GIVEN HALF WORD ELEMENT CODE VECTOR 'CD'
*RETURN P-COUNTER OF THE LAST INSTRUCTION BEFORE 'EPC'
*BUT AFTER OR INCLUDING INSTRUCTION AT 'SPC'

   FOR L _ INSTR\LENGTH ( CD, SPC, EPC ) WHILE SPC + L < EPC DO
      SPC _ SPC + L
   ENDFOR

   RETURN SPC
END
```

```
FUNCTION FIXPTRS(SCALAR BLK,SCALAR WRD,SCALAR WORDINC,SCALAR LINEINC)
*
* FIXPOINTERS CHECKS ALL TEXT POINTERS AND ADJUSTS THOSE WHICH
*  HAVE BECOME INVALID DUE TO MOVEMENT OF TEXT IN BLK, AT
*  OR BELOW WORD. SUCH POINTERS HAVE THEIR WORD AND LINE
*  FIELDS ADJUSTED BY WORDINC AND LINEINC, RESPECTIVELY.
*
   VECTOR A
   REFERENCE P
   SCALAR I
   *
   A _ TXWINDOW.BARRAY
   *
   FOR I _ 0 TO NPTRS -1 DO
      P _ PTRADDR ( I )
      IF P.BLOCK = BLK AND P.STARTX >= WRD DO
         P.STARTX _ P.STARTX + WORDINC
         P.ENDX _ P.ENDX + WORDINC
         MAKESTRING ( @(P.STR), A, P.STARTX, P.ENDX )
         P.LINE _ P.LINE + LINEINC
      ENDIF
   ENDFOR
   RETURN
END
```

```
FUNCTION FNAME ( SCALAR FNUM, STRING FSTR )
*
* FNAME CONVERTS A FUNCTION-NUMBER INTO A
*  SYMBOLIC NAME VIA THE GLOBAL SYMBOL TABLE
*
   IF INVALID ( GLOBLK ) AND FNUM $ FNRUNT = 0 DO
      CLEAR ( FSTR )
      CNS ( FSTR, FNUM $ FNBLK )
   ELSE DO
      FUNCNAME ( FNUM, FSTR, DBUGSEG ( FNUM $ FNRUNT ) )
   ENDIF
   *
   RETURN
END
```

```
FUNCTION FNFIND ( STRING NAME )
*
* LOOK UP A FUNCTION BY NAME OR NUMBER
*
   STRING FNAME
   SCALAR CHAR, FNUM
   REFERENCE ENTRY
   *
   SDCOPY ( FNAME, NAME )
   IF STREQL ( FNAME, "GLOBAL" ) DO
      RETURN GLOBLK
   ELSE DO
      CHAR _ GCNI ( FNAME :RETURN -1 )
      IF NUMERIC ( CHAR ) DO
         FNUM _ CSN ( FNAME )
         IF LENGTH ( FNAME ) > 0 DO RETURN -1
         IF FNUM < 1 OR FNUM > NBLOCKS DO
            RETURN -1
         ELSE DO
            RETURN FNUM
         ENDIF
      ELSE DO
         IF INVALID ( GLOBLK ) DO
            FRETURN
         ENDIF
         ENTRY _ STEFIND ( FNAME, GLOBLK :RETURN -1 )
         IF ISFUNCTION ( ENTRY ) DO
            RETURN STEFUNCNO ( ENTRY )
         ELSE DO
            RETURN -1
         ENDIF
      ENDIF
   ENDIF
END
```

```
FUNCTION FRESH\SEGS()

* MAKE NEW CODE AND DEBUG SEGS AND SET PROG INVALID

   NEW\SEGMENT ( CODENAME, 1 )
   NEW\SEGMENT ( DEBUGNAME, 1 )
   SETINVALID ( GLOBLK, TRUE )
   RETURN
END
```

```
FUNCTION FTRCPROC()
*
* FUNCTION-TRACE TRAP
*
   MESSAGE ( "FUNCTION TRACE TRAP.&M" )
   RETURN
END
```

```
FUNCTION FUNCNAME ( SCALAR FNUM, STRING FSTR, VECTOR SEG )

* CONVERT FUNCTION NUMBER INTO SYMBOLIC NAME

   SCALAR BLK, I, J, K, L, C
   REFERENCE ENTRY
   VECTOR A
   STRING ASTR

   BLK _ FNUM $ FNBLK
   IF BLK = GLOBLK DO
      SCOPY ( FSTR, GLOBNAME )
   ELSE DO
      SELWINDOW ( GDBWINDOW, SEG, GLOBLK )

      A _ GDBWINDOW . BARRAY
      I _ BLKINFO ( SEG, BLK )
      ENTRY _ @A[I]
      L _ ENTRY . SSIZE
      J _ I + SDISP
      K _ J + L - 1

      MAKESTRING ( ASTR, A, J, K )
      SCOPY ( FSTR, ASTR )
      CLRWINDOW ( GDBWINDOW )
      WHILE ( C _ GCD(FSTR) ) = BLANK DO; ENDWHILE
      WCI ( C, FSTR )
   ENDIF

   RETURN
END
```

```
FUNCTION FWD ( REFERENCE POINTER )
*
* FWD MOVES AN ACTIVE POINTER AHEAD ONE LINE. END-OF-FILE
*  OR END-OF-BLOCK CAUSE FAILURE, WRAPAROUND, OR MOVE TO NEXT
*  BLOCK, DEPENDING ON SETTING OF 'SCOPE' AND 'WRAP'
*
   SCALAR SX, B, L
   *
   L _ POINTER.LINE + 1
   SX _ POINTER.ENDX + 1
   IF SX = BLENGTH ( TXWINDOW ) DO
      IF SCOPE = GLOBALLY DO
         B _ POINTER.BLOCK + 1
         IF B > NBLOCKS DO
            IF WRAP DO
               B _ 1
            ELSE DO
               FRETURN
            ENDIF
         ENDIF
         POINTER.BLOCK _ B
         UNLESS ACTIVE ( POINTER ) DO CRASH()
      ELSE DO
         UNLESS WRAP DO FRETURN
      ENDIF
      L _ 1
      SX _ 0
   ENDIF
   POINTER.LINE _ L
   POINTER.STARTX _ SX
   *
   SPANLINE ( POINTER )
   RETURN
   *
END
```

```
FUNCTION GCONST ( STRING NAME )
*
* LOOK UP NAMED SCALAR CONSTANT IN GLOBAL SYMBOL TABLE
*
   REFERENCE ENTRY
   *
   ENTRY _ STEFIND ( NAME, GLOBLK :FRETURN )
   *
   IF ENTRY . STYPE = SCONST DO
      RETURN ENTRY . SVAL
   ELSE DO
      FRETURN
   ENDIF
   *
END
```

```
FUNCTION GETBYTE ( SCALAR FN, SCALAR ADDR )
*
* GET CODE BYTE FROM USER CODE SEGMENT
*  F-BASE REL ADDRESS 'ADDR' IN FUNCTION 'FN'.
*
   VECTOR A
   SCALAR VAL
   *
   SELWINDOW ( CCDWINDOW, CDSEG, FN $ FNBLK )
   A _ HALFVEC ( CCDWINDOW.BARRAY )
   VAL _ A [ ADDR ]
   CLRWINDOW ( CCDWINDOW )
   RETURN VAL
END
```

```
FUNCTION GNBCI ( STRING S )
*
* GET NON-BLANK CHARACTER AND INCREMENT
*
   SCALAR CHAR
   *
   WHILE ( CHAR _ GCI ( S :FRETURN )) = BLANK DO; ENDWHILE
   RETURN CHAR
END
```

```
FUNCTION GOPROC()
*
* GO-COMMAND:
*  RESUME EXECUTION
*
   REFERENCE TOP
   SCALAR BYTE, FLAGS, PREVSTEP, FN
   FIELD PATCH ( 0: 16,28 )
   CONSTANT FLSTEP _ 060020B
   CONSTANT STEPTRAP _ 4
   CONSTANT BRKOP _ 00417B
   *
   SCHECK
   IF NOSTACK() DO
      ERRMSG ( "STACK EMPTY&M" )
      RETURN
   ENDIF
   *
KLUDGE\LOCK()
   IF CSFR.DOSAVEDCODE DO
      TOP _ FIND\FRAME ( STATEWORD . SWLBASE )
      FN _ FNUMBER ( TOP . SFRUNT, TOP . SFBLKNO )
      *
      * BACK-UP PCTR OVER BKPT-TRAP INSTRUCTION
      *
      TOP.SFPCTR _ TOP.SFPCTR - 1
      *
      * RE-FIND LTE CORRESP TO PCTR
      *
      LTEFIND ( FN, TOP.SFPCTR :CRASH() )
      IF LTE $ LTEFLAGS # 0 DO
         *
         * RESTORE SAVED CODE FROM LINE-TABLE
         *
         BYTE _ GETBYTE ( FN, LTE $ LTECODEPTR )
         BYTE $ PATCH _ LTE $ LTEISAVE
         PUTBYTE ( FN, LTE $ LTECODEPTR, BYTE )
         *
         * SET SINGLE-INSTRUCTION-STEP MODE
         *
         FLAGS _ GETFIELD ( STATEWORD, SWFLAGS )
         PUTFIELD ( STATEWORD, SWFLAGS, OR ( FLAGS, FLSTEP ) )
         PREVSTEP _ FLAGS AND FLSTEP
         *
         * EXECUTE SAVED CODE
         *
         GOUSER()
         *
         * RESTORE PREVIOUS STATE OF STEP MODE
         *
         FLAGS _ GETFIELD ( STATEWORD, SWFLAGS )
         FLAGS _ AND ( FLAGS, EOR ( FLSTEP, -1 ) )
         FLAGS _ OR ( PREVSTEP, FLAGS )
         PUTFIELD ( STATEWORD, SWFLAGS, FLAGS )
```

```
            *
            * RESTORE BKPT-TRAP INSTRUCTION
            *
            BYTE $ PATCH _ BRKOP
            PUTBYTE ( FN, LTE $ LTECODEPTR, BYTE )
            *
            * IF SAVED CODE GOT UNEXPECTED TRAP, WE ARE DONE
            *
            IF STATEWORD . SWTRAPCLASS # STEPTRAP DO
               USERTRAPPED _ TRUE
            ENDIF
         ENDIF
      ENDIF
      UNLESS USERTRAPPED DO
         GOUSER()
         USERTRAPPED _ TRUE
      ENDUNLESS
KLUDGE\UNLOCK()
      RETURN
END
```

```
FUNCTION GOUSER()
*
*   RUN USER PROCESS UNTIL NEXT TRAP
*
    MACRO C(V) _ V $ VDCAP
    MACRO PUSH ( X ) _ ((X)&CODE(4114B))
    MACRO TRAP ( N ) _ ((N)&CODE(4164B,44106B,1,4100B))
    *
    PUSH( -1 )
    PUSH ( C(RCDSEG) )
    PUSH ( C(CDSEG)  )
    PUSH ( C(DTSEG) )
    PUSH ( C(CLSEG)  )
    STATEWORD . SWTRAPCLASS _ TRAP ( 26 ); *SET STATE IN CASE OF PANIC *
    RETURN
END
```

```
FUNCTION INBREAK ( SCALAR BLK, SCALAR LTE )
*
* INSERT BREAKPOINT-TRAP PATCH
*
   SCALAR BYTE
   FIELD PATCH ( 0: 16,28 )
   CONSTANT BRKOP _ 00417B
   *
   IF LTE $ LTEFLAGS = 0 DO
      BYTE _ GETBYTE ( BLK, LTE $ LTECODEPTR )
      LTE $ LTEISAVE _ BYTE $ PATCH
      BYTE $ PATCH _ BRKOP
      PUTBYTE ( BLK, LTE $ LTECODEPTR, BYTE )
   ENDIF
   RETURN LTE
END
```

```
FUNCTION INPROC()

* 'IN' COMMAND:
*      SET CONTEXT FOR SUBSEQUENT DEBUGGING COMMANDS

   SCALAR FN, ACTN

   SCHECK

   IF NOSTACK() DO FRETURN
   FN _ CMDFN ( :FRETURN )
   ACTN _ CSN ( CSTRING :1 )
   IF ACTN < 1 DO FRETURN
   ENV _ FIND\ACT ( FN, ACTN :RETURN )
   RETURN
END
```

```
FUNCTION INSTR\LENGTH ( VECTOR CD, PC, EPC )

*GIVEN HALF WORD ELEMENT CODE VECTOR 'CD', AND P-CTR OF INSTR 'PC'
*RETURN THE LENGTH OF THE INSTRUCTION IN HALF WORDS
*THE LENGTH OF A BRANCH INSTRUCTION WILL INCLUDE THE SPACE SPANNED
*BY THE BRANCH SO LONG AS THIS DOES NOT EXTEND TO 'EPC' OR BEYOND

SCALAR T
FIELD CMPLXA ( 0 : 17, 17 )
FIELD OPCODE ( 0 : 18, 22 )

   IF CD[PC] $ CMPLXA = 0 DO;*SHORT FORM OPERAND
      RETURN ADDR\LENGTH ( CD, PC )
   ELSEIF CD[PC] $ OPCODE = 4 DO;*BRANCH
      T _ PC + 2 + CD[PC+1]
      RETURN (IF T < EPC THEN T ELSE 2)
   ELSE DO;*COMPLEX OPERAND
      T _ ADDR\LENGTH ( CD, PC )
      RETURN T + ADDR\LENGTH ( CD, PC+T )
   ENDIF

END
```

```
FUNCTION INTBUF ( BOOL BEFORE, REFERENCE POINTER )
*
* INTBUF INSERTS THE CONTENTS OF THE TEXT BUFFER
*  BEFORE OR AFTER THE LINE INDICATED BY THE POINTER.
*
   SCALAR OLDLENGTH, NEWLENGTH, START, FIN, REST
   VECTOR OLD, NEW
   *
   UNLESS ACTIVE ( POINTER ) DO CRASH()
   OLDLENGTH _ BLENGTH ( TXWINDOW ); OLD _ TXWINDOW.BARRAY
   NEWLENGTH _ OLDLENGTH + TLENGTH
   NEW _ SMAKE ( NEWLENGTH )
   *
   IF BEFORE DO
      START _ POINTER.STARTX
   ELSE DO
      START _ POINTER.ENDX + 1
   ENDIF
   FIN _START + TLENGTH
   REST _ OLDLENGTH - START
   *
   BCOPY ( @NEW[0] , @OLD[0] , START )
   BCOPY ( @NEW[START] , @TARRAY[0] , TLENGTH )
   IF REST > 0 DO
      BCOPY ( @NEW[FIN] , @OLD[START] , REST )
   ENDIF
   *
   SETWINDOW ( TXWINDOW , NEW )
   MAKEINVALID ( LOCALLY )
   *
   FIXPTRS ( POINTER.BLOCK , START , TLENGTH , TLINES )
   *
   RETURN
   *
END
```

```
FUNCTION ITEM ( STRING T, STRING S, C )

* STRIP LEADING SUBSTRING FROM 'S' UP TO CHAR 'C' AND LEAVE IN 'T'

   SCALAR CHAR, QUOTE
   *
   QUOTE _ 0
   CLEAR ( T )
   CHAR _ GNBCI ( S :RETURN )
   *
   UNTIL ( CHAR = C AND QUOTE = 0 ) DO
      IF CHAR = '"' OR CHAR = ':' DO
         IF QUOTE = 0 DO
            QUOTE _ CHAR
         ELSEIF QUOTE = CHAR DO
            QUOTE _ 0
         ENDIF
      ENDIF
      WCI ( CHAR, T )
      CHAR _ GCI ( S :RETURN )
   ENDUNTIL
   RETURN
   *
END
```

```
FUNCTION ITPROC()
*
* INSERT-TEXT COMMAND
*
   PCHECK
   *
   IF CMDMOVE ( IPTR1 :FRETURN ) DO
      INTBUF ( TRUE, IPTR1 )
      SETPOINTER ( CURRENT, IPTR1 )
   ENDIF
   RETURN
END
```

```
FUNCTION KLUDGE\LOCK()

* LOCK USER PROG IN CORE AND TURN OFF TSS PRE-EMPTION

   MACRO C(V) _ V$VDCAP
   MACRO PUSH(X) _ ((X)&CODE(4114B))
   MACRO TRAP(N) _ ((N)&CODE(4164B,44106B,1,4100B))

   CLOSE ( C(CDSEG) )
   CLOSE ( C(DTSEG) )
   CLOSE ( C(CLSEG) )

   PUSH ( 0 ); TRAP ( 28 ); *DISABLE TSS

   OPEN\LOCKED ( C(CDSEG), CODENAME )
   OPEN\LOCKED ( C(DTSEG), DATANAME )
   OPEN\LOCKED ( C(CLSEG), CAPNAME )

   RETURN
END
```

```
FUNCTION KLUDGE\UNLOCK()

* UNLOCK USER PROGRAM FROM CORE AND RE-ENABLE TSS PRE-EMPTION

   MACRO C(V) _ V$VDCAP
   MACRO PUSH(X) _ ((X)&CODE(4114B))
   MACRO TRAP(N) _ ((N)&CODE(4164B,44106B,1,4100B))

   CLOSE ( C(CLSEG) )
   CLOSE ( C(DTSEG) )
   CLOSE ( C(CDSEG) )

   PUSH ( 1 ); TRAP ( 28 ); * RE-ENABLE TSS

   OPEN\WINDOWED ( C(CLSEG), CAPNAME :OPEN\LOCKED ( C(CLSEG), CAPNAME ) )
   OPEN\WINDOWED ( C(DTSEG), DATANAME :OPEN\LOCKED ( C(DTSEG), DATANAME ) )
   OPEN\WINDOWED ( C(CDSEG), CODENAME :OPEN\LOCKED ( C(CDSEG), CODENAME ) )

   RETURN
END
```

```
FUNCTION LMATCH ( STRING ARG\TARGET , STRING ARG\SUBJECT )
*
* LMATCH LOOKS FOR TARGET AS A LABEL IN SUBJ STRING.
*  ( I.E.  POS(0) ARBNO(' ') TARGET (COLON!BLANK!CR)  )
*  IT RETURNS TRUE OR FALSE
*
   STRING TARGET, SUBJECT
   MACRO LABEL\END ( C ) _ ( (C) = ':' OR (C) = BLANK ..
                          OR (C) = BLANKS OR (C) = NUL OR (C) = CARRET )
   SCALAR CHAR, TCHAR
   *
   SDCOPY ( TARGET, ARG\TARGET )
   SDCOPY ( SUBJECT, ARG\SUBJECT )
   *
   CHAR _ GNBCI(SUBJECT)
   *
   IF CHAR = BLANKS DO
      GCI ( SUBJECT )
      CHAR _ GCI ( SUBJECT )
   ENDIF
   TCHAR _ GCI ( TARGET :RETURN LABEL\END(CHAR) )
   WHILE CHAR = TCHAR DO
      CHAR  _ GCI ( SUBJECT :RETURN LENGTH ( TARGET ) = 0 )
      TCHAR _ GCI ( TARGET :RETURN LABEL\END(CHAR) )
   ENDWHILE
   *
   RETURN FALSE
END
```

```
FUNCTION LOADTBUF ( REFERENCE PTR1 , REFERENCE PTR2 )
*
* LOADTBUF COPIES ALL LINES BEWTEEN PTR1 AND PTR2 INCLUSIVE
*  INTO THE TEXT BUFFER IF THEY FIT, AND RETURNS TRUE.
*  IF THE TEXT BUFFER WOULD HAVE OVERFLOWED, NOTHING
*  IS DONE AND FALSE IS RETURNED. LOADTBUF FAILS IF
*  PTR1 AND PTR2 ARE ARRANGED UNREASONABLY.
*
   VECTOR A
   SCALAR TL, START
   *
   UNLESS ACTIVE ( PTR1 ) DO CRASH()
   IF PTR1.BLOCK # PTR2.BLOCK OR PTR1.LINE > PTR2.LINE DO
      FRETURN
   ENDIF
   *
   TL _ PTR2.ENDX - PTR1.STARTX + 1
   IF TL > TSZ DO
      RETURN FALSE
   ENDIF
   TLENGTH _ TL
   TLINES _ PTR2.LINE - PTR1.LINE + 1
   *
   START _ PTR1.STARTX
   A _ TXWINDOW.BARRAY
   *
   BCOPY ( @TARRAY[0] , @A[START] , TLENGTH )
   *
   RETURN TRUE
   *
END
```

```
FUNCTION LOOKUP ( STRING PNAME )
*
* LOOKUP REFORMATS NAME AND CALLS "STLK" TO LOOK IN
*  SYMBOL TABLE IN BUFFER.
*
   SCALAR WLENGTH
   VECTOR WVEC
   STRING NAME [ IDLENGTH + 3 ]
   *
   SCOPY ( NAME , PNAME )
   *
   UNTIL ( LENGTH ( NAME ) MOD CPW = 0 ) DO
      WCI(BLANK,NAME)
   ENDUNTIL
   WVEC _ FULLVEC ( NAME.SDVD )
   WLENGTH _ LENGTH ( NAME ) / CPW
   *
   RETURN STLK ( GDBWINDOW.BUFFER, WVEC, WLENGTH :FRETURN)
   *
END
```

```
FUNCTION LSINIT()
*
* INITIALIZE EDITOR AND DEBUGGER GLOBAL VARIABLES
*
   SCALAR I
   *
   * SET UP FREE-POOL
   *
   INIT ( FREEPOOL )
   *
   * LINE POINTERS
   *
   FOR I _ 0 TO NPTRS-1 DO
      MAKENULL ( PTRADDR ( I ) )
   ENDFOR
   *
   CURRENT _ PTRADDR(0)
   SEARCH _ PTRADDR(1)
   IPTR1 _ PTRADDR(2)
   IPTR2 _ PTRADDR(3)
   *
   * WINDOWS
   *
   TXWINDOW _ WINADDR(0)
      TXWINDOW.BUFFER _ CTXBUF
   GDBWINDOW _ WINADDR(1)
      GDBWINDOW.BUFFER _ GDBBUF
   CDBWINDOW _ WINADDR(2)
      CDBWINDOW.BUFFER _ CDBBUF
   CCDWINDOW _ WINADDR(3)
      CCDWINDOW.BUFFER _ CCDBUF
   *
   *
*
* TEXT BUFFER
*
   TSZ _ 1000
   TARRAY _ SMAKE ( TSZ )
   *
   STE _ SETREFSIZE ( @ STEVEC [ 0 ], SDISP )

   STATEWORD _ SETREFSIZE ( @DTSEG[0], SW\SIZE )

*
* STATE VARIABLES
*
   WRAP _ TRUE
   SCOPE _ GLOBALLY
   *
   MEASFLAG _ FALSE;* INITIALLY NOT MEASURING
   RUNFLAG _ FALSE;* INITIALLY IN USER-MODE
   USERTRAPPED _ FALSE
   *
   LOADED _ FALSE
```

```
    RTLOADED _ FALSE
    RTSELECT ( "SRUN" :CRASH() )

    COMPLEVEL _ 0
    CSFR _ @COMPSTACK[0]
    CLEAR ( PSNAME )
    *
    RETURN
    *
    *
END
```

```
FUNCTION LTEFIND ( FN, P )
*
* FIND LINE-TABLE-ENTRY CORRESPONDING TO FUNCTION-RELATIVE
*  P-COUNTER P IN FUNCTION FN. LOADS THE
*  GLOBAL LTE BUFFER AND RETURNS THE LINE NUMBER OF THE
*  STATEMENT-LINE. RETURNS -1 IF P IS OUTSIDE OF F.
*  FAILS IF DEBUGGER BLOCK OF F IS INVALID.
*
   SCALAR START, FINISH, FIRST, NLINES, I, CP, R, F
   REFERENCE HDR
   VECTOR A
   *
   R _ FN $ FNRUNT
   F _ FN $ FNBLK
   *
   *CHECK FOR INVALID DEBUGGER BLOCK
   *
   IF BADINFO ( FN ) DO
      FRETURN
   ENDIF
   *
   * CHECK FOR BAD P-COUNTER
   *
   START _ INST1
   FINISH _ BLKSIZE(PROCSEG(R),F)*BPW - 1
   IF P<START OR FINISH<P DO
      RETURN -1
   ENDIF
   *
   * GET LINE TABLE
   *
   SELWINDOW ( CDBWINDOW, DBUGSEG(R), F )
   A _ CDBWINDOW.BARRAY
   HDR _ FIND\DBHDR ( A )
   FIRST _ HDR.LTBASE
   NLINES _ HDR.LTSIZE
   *
   * SEARCH FOR LTE CONTAINING P-COUNTER VALUE
   *
   FOR I _ NLINES-1 BY -1 TO 0 DO
      LTE _ A[FIRST + I*WPW]
      CP _ LTE $ LTECODEPTR
      IF I = 0 OR ( CP # LTENULL AND P >= CP ) DO
         CLRWINDOW ( CDBWINDOW )
         RETURN I+1
      ENDIF
   ENDFOR
   *
   * FALL THRU IF BAD LINE TABLE
   *
   CRASH()
END
```

```
FUNCTION LTEGET ( VECTOR S, SCALAR F, SCALAR L )
*
* GET LINE TABLE ENTRY FOR LINE L OF FUNCTION F OF
*  DEBUG SEGMENT S. LOADS GLOBAL LTE BUFFER, AND
*  RETURNS TRUE. RETURNS FALSE IF NO SUCH LINE IN F.
*  FAILS IF DEBUGGER BLOCK OF F IS INVALID.
*
   SCALAR FIRST, NLINES
   REFERENCE HDR
   VECTOR A
   *
   * CHECK FOR INVALID DEBUGGER BLOCK
   *
   IF BADINFO ( F ) DO
      FRETURN
   ENDIF
   *
   * GET LINE TABLE
   *
   SELWINDOW ( CDBWINDOW, S, F )
   A _ CDBWINDOW.BARRAY
   HDR _ FIND\DBHDR ( A )
   FIRST _ HDR.LTBASE
   NLINES _ HDR.LTSIZE
   *
   * CHECK FOR BAD LINE NUMBER
   *
   IF L > NLINES DO
      CLRWINDOW ( CDBWINDOW )
      RETURN FALSE
   ENDIF
   *
   * FETCH LTE
   *
   LTE _ A [ FIRST + L - 1 ]
   CLRWINDOW ( CDBWINDOW )
   RETURN TRUE
   *
END
```

```
FUNCTION LTEPUT ( VECTOR S, SCALAR F, SCALAR L )
*
* PUT MODIFIED LTE FOR LINE L OF FUNCTION F OF DEBUG SEGMENT
*  S BACK INTO LINE-TABLE.
*
   SCALAR FIRST, NLINES
   REFERENCE HDR
   VECTOR A
   *
   IF BADINFO ( F ) DO
      FRETURN
   ENDIF
   *
   SELWINDOW ( CDBWINDOW, S, F )
   A _ CDBWINDOW.BARRAY
   HDR _ FIND\DBHDR ( A )
   FIRST _ HDR.LTBASE
   NLINES _ HDR.LTSIZE
   *
   IF L > NLINES DO
      CLRWINDOW ( CDBWINDOW )
      RETURN FALSE
   ENDIF
   *
   A [ FIRST + L - 1 ] _ LTE
   OUTWINDOW ( CDBWINDOW )
   RETURN TRUE
END
```

```
FUNCTION LTPROC()
*
* LOAD-TEXT COMMAND
*
    PCHECK
    *
    IF XTPROC ( LOADTBUF :FRETURN ) DO
       ACTIVE ( CURRENT ); FWD ( CURRENT )
       TTPROC()
    ENDIF
    RETURN
END
```

```
FUNCTION LVOUT ( STRING L, SCALAR V )
*
* PRINT LABELED OCTAL VALUE
*
   SOUT ( L )
   IOUT ( V,, 8 )
   RETURN
END
```

```
FUNCTION MAKEINVALID ( SCALAR SCOPE )
*
* TURNS ON INVALID BIT IN TEXT SEGMENT DIR ENTRY
*
   IF SCOPE = GLOBALLY DO
      SETINVALID ( GLOBLK , TRUE )
   ELSE DO
      SETINVALID ( TXWINDOW.BLOCK , TRUE )
   ENDIF
   RETURN
END
```

```
FUNCTION MAKESTRING ( STRING S, VECTOR V, I, J )
*
* MAKES STRING 'S' HOLD CHARACTERS PACKED IN
*  WORD-VECTOR 'V', ADDING THE "EXTRA" CHAR AT
*  THE END TO MAKE A KOSHER STRING.
*
   V _ SUBVEC ( V, I, J-I+1 )
   V _ BYTEVEC ( V )
   V $ VDSIZE _ V $ VDSIZE + 1
   SETSTRING ( S, V )
   RETURN
END
```

```
FUNCTION MAKE\PROGRAM ( STRING PNAME )
*
* MAKE THE CURRENT USER PROGRAM BE 'PNAME'
*
   SCOPY ( PROGNAME, PNAME )
   SEGNAME ( TEXTNAME, TEXTSEG, PNAME )
   SEGNAME ( CODENAME, CODESEG, PNAME )
   SEGNAME ( DEBUGNAME, DEBUGSEG, PNAME )
   SEGNAME ( DATANAME, DATASEG, PNAME )
   SEGNAME ( CAPNAME, CAPSEG, PNAME )
   RETURN
END
```

```
FUNCTION MAKE\RUNTIME ( STRING RTNAME )
*
* MAKE THE CURRENT RUNTIME BE 'RTNAME'
*
   SCOPY ( RUNTNAME, RTNAME )
   SEGNAME ( RTEXTNAME, TEXTSEG, RTNAME )
   SEGNAME ( RCODENAME, CODESEG, RTNAME )
   SEGNAME ( RDEBUGNAME, DEBUGSEG, RTNAME )
   RETURN
END
```

```
FUNCTION MOVEWINDOW ( REFERENCE W, VECTOR SEG, BLK )

* MOVE CONTENTS OF WINDOW W TO DIFFERENT
*   BLOCK OF DIFFERENT SEGMENT

    SCALAR B

    B _ W . BUFFER

    W . SEGMENT _ BUFSEG [ B ] _ SEG
    W . BLOCK   _ BUFBLK [ B ] _ BLK

    RETURN
END
```

```
FUNCTION MTPROC()
*
* MTPROC MODIFIES THE SINGLE LINE IN THE TEXT BUFFER.
*  IT FAILS IF MULTIPLE LINES ARE IN THE BUFFER.
*
   STRING TSTRING
   *
   UNLESS ( TLINES = 1 ) DO FRETURN
   *
   MAKESTRING ( TSTRING, TARRAY, 0, TSZ-1 )
   SETS ( TSTRING, 0, TLENGTH*CPW )
   UNPAD ( TSTRING )
   *
   EDITLINE ( TSTRING :CRASH() )
   *
   PADLINE ( TSTRING )
   TLENGTH _ LENGTH ( TSTRING ) / CPW
   *
   RETURN
END
```

```
FUNCTION MVPROC()
*
* MOVE-POINTER COMMAND
*
    *
    PCHECK
    *
    IF CMDMOVE ( IPTR1 :FRETURN ) DO
       SETPOINTER ( CMDPTR ( :FRETURN ), IPTR1 )
    ENDIF
    RETURN
END
```

```
FUNCTION MVPTR ( REFERENCE CP, STRING DESTIN )
*
* MOVE POINTER CP TO LINE ADDRESS DESTIN.
*
   BOOL FOUND
   PROC MOVE
   SCALAR NLINES, CHAR, I
   REFERENCE DP
   *
   SETPOINTER ( CP, CURRENT )
   *
   * LOOP THRU ELEMENTS OF DESTINATION
   *
   WHILE REAL ( CHAR _ GCI ( DESTIN :NONCHAR ) ) DO
      CLEAR ( TARGET )
      *
      MOVE _ FWD
      IF CHAR = '+' DO
         CHAR _ GCI ( DESTIN :FRETURN )
      ELSEIF CHAR = '-' DO
         CHAR _ GCI ( DESTIN :FRETURN )
         MOVE _ BACK
      ENDIF
      *
      * ABSOLUTE-LOCATION ELEMENTS
      *
      IF ALPHABETIC ( CHAR ) DO
         FOUND _ FALSE
         FOR I _ 0 TO NPTRS-1 DO
            IF CHAR = PTRNAME [ I ] DO
               FOUND _ TRUE
               DP _ PTRADDR ( I )
               UNLESS ACTIVE ( DP ) DO GOTO BADPTR
               SETPOINTER ( CP, DP )
            ENDIF
         ENDFOR
         UNLESS FOUND DO FRETURN
      ELSEIF CHAR = '/' DO
         UNTIL ( ( CHAR _ GCI ( DESTIN :FRETURN )) = '/' ) DO
            WCI ( CHAR, TARGET :CRASH() )
         ENDUNTIL
         IF NOT PTBLK ( TARGET, CP :GOTO BADGST ) DO
            NOTFOUND ( TARGET, '/' )
            RETURN FALSE
         ENDIF
*
```

```
      *
      * RELATIVE-LOCATION ELEMENTS
      *
      ELSEIF CHAR = ':' DO
         UNTIL ( ( CHAR _ GCI ( DESTIN :FRETURN )) = ':' ) DO
            WCI ( CHAR, TARGET :CRASH() )
         ENDUNTIL
         IF NOT SCAN ( LMATCH, MOVE, TARGET, CP :GOTO BADPTR ) DO
            NOTFOUND ( TARGET , ':' )
            RETURN FALSE
         ENDIF
      ELSEIF CHAR = '"' DO
         UNTIL ( ( CHAR _ GCI ( DESTIN :FRETURN )) = '"' ) DO
            WCI ( CHAR, TARGET :CRASH() )
         ENDUNTIL
         IF NOT SCAN ( AMATCH, MOVE, TARGET, CP :GOTO BADPTR ) DO
            NOTFOUND ( TARGET, '"' )
            RETURN FALSE
         ENDIF
      ELSEIF NUMERIC ( CHAR ) DO
         WCD ( CHAR, DESTIN )
         UNLESS ACTIVE ( CP ) DO GOTO BADPTR
         NLINES _ CSN ( DESTIN )
         FOR I _ 1 TO NLINES DO
            MOVE ( CP :GOTO NOSUCH )
         ENDFOR
      ELSE DO
         FRETURN
      ENDIF
   ENDWHILE
   *
   IF ACTIVE ( CP ) DO
      RETURN TRUE
   ENDIF
   *
```

```
  BADPTR:
   ERRMSG ( "POINTER IS NULL&M" )
   RETURN FALSE
   *
  NOSUCH:
   ERRMSG ( "NO SUCH LINE&M" )
   RETURN FALSE
   *
  BADGST:
   ERRMSG ( "ALL FUNCTIONS UNKNOWN. (GST INVALID)&M" )
   RETURN FALSE
   *
END
```

```
FUNCTION NEWPOINTER ( REFERENCE P, SCALAR BLK, SCALAR LN, SCALAR INDEX )
*
* SET FIELDS IN POINTER P
*
   P.SEGMENT _ TXSEG
   P.BLOCK _ BLK
   P.LINE _ LN
   P.STARTX _ INDEX
   UNLESS ACTIVE ( P ) DO CRASH()
   SPANLINE ( P )
   RETURN
END
```

```
FUNCTION NEWPROGRAM ( STRING PNAME )
*
* NEWPROGRAM DROPS THE CURRENT PROGRAM AND CREATES A
*  NEW PROGRAM NAMED 'PNAME'. IT FAILS IF THE PROGRAM
*  NAME IS ILLEGAL.
*
   MACRO C(V) _ V $ VDCAP
   VECTOR LNVEC
   *
   IF LENGTH ( PNAME ) > PNAMEMAX OR LENGTH ( PNAME ) < 1 DO
      ERRMSG ( "BAD NAME&M" )
      RETURN
   ENDIF
   IF RTLOADED AND STREQL ( PNAME, RUNTNAME ) DO ..
      ERRMSG ( "SAME AS RUNTIME$M" )
   *
   DROP()
   *
   MAKE\PROGRAM ( PNAME )

   CREATE\SEGMENT ( TEXTNAME, 1: )
   CREATE\SEGMENT ( CODENAME, 1: )
   CREATE\SEGMENT ( DEBUGNAME, 1: )

   OPEN\WINDOWED ( C(TXSEG), TEXTNAME : GOTO FUSS )
   OPEN\WINDOWED ( C(DBSEG), DEBUGNAME : GOTO FUSS )
   OPEN\WINDOWED ( C(CDSEG), CODENAME : GOTO FUSS )

   SEGINIT ( TXSEG )
   SEGINIT ( DBSEG )
   SEGINIT ( CDSEG )
   *
   BLKCREATE ( TXSEG , GLOBLK )
   LNVEC _ SMAKE ( 1 )
   LNVEC [ 0 ] _ 'END&M'
   NEWWINDOW ( TXWINDOW, TXSEG, GLOBLK, LNVEC )
   NEWPOINTER ( CURRENT , GLOBLK , 1 , 0 )
   SETINVALID ( GLOBLK , TRUE )
   *
   LOADED _ TRUE
   RETURN
  FUSS:
   ERRMSG ( "DIRECTORY FULL&M" )
   RETURN
END
```

```
FUNCTION NEWWINDOW ( REFERENCE W, VECTOR SEG, SCALAR BLK, VECTOR NEW )
*
* PUTS A NEW BLOCK IN WINDOW W
*
   W.SEGMENT _ SEG
   W.BLOCK _ BLK
   BUFSET ( SEG, BLK, W.BUFFER, NEW )
   W.BARRAY _ NEW
   RETURN
END
```

```
FUNCTION NEW\SEGMENT ( STRING SEGNAME, LENGTH )
*
* CREATES A NEW SEGMENT, OR ADJUSTS SIZE OF EXISTING ONE.
*
   SCALAR L
   BOOL EXISTS

   EXISTS _ TRUE
   L _ READ\LENGTH ( SEGNAME :EXISTS _ FALSE )
   IF EXISTS DO
      OPEN\WINDOWED ( TCAP, SEGNAME :OPEN\LOCKED(TCAP,SEGNAME:CRASH()))
      REPEAT
         CLOSE ( TCAP :EXIT )
      ENDREPEAT
      IF L # LENGTH DO
         OPEN\WINDOWED ( TCAP, SEGNAME )
         SET\LENGTH ( TCAP, LENGTH )
         CLOSE ( TCAP )
      ENDIF
   ELSE DO
      CREATE\SEGMENT ( SEGNAME, LENGTH :CRASH() )
   ENDIF
   RETURN
END
```

```
FUNCTION NLINES ( REFERENCE WINDOW )
*
* NLINES RETURN THE NUMBER OF LINES IN THE WINDOW.
*
   VECTOR A
   SCALAR I, N, L
   *
   N _ BLENGTH ( WINDOW ) - 1
   A _ WINDOW.BARRAY
   L _ 0
   FOR I _ 0 TO N DO
      UNLESS MORELINE ( A [ I ] ) DO L _ L + 1
   ENDFOR
   RETURN L
END
```

```
FUNCTION NOGOOD ( VECTOR SEG, SCALAR BLK )
*
* TESTS INVALID BIT OF BLOCK BLK IN SEGMENT SEG
*
   SCALAR INFO
   *
   INFO _ BLKINFO ( SEG, BLK )
   RETURN INFO $ BLKINVALID
END
```

```
FUNCTION NOGOPROC()
*
* START-UP-IMPOSSIBLE TRAP
*
   MESSAGE ( "START-UP IMPOSSIBLE.&M" )
   RETURN
END
```

```
FUNCTION NOSTACK()
*
* DUMMY
*
   RETURN ( STATEWORD . SWLTOP = 0 )
END
```

```
FUNCTION NOTFOUND ( STRING S, SCALAR C )
*
* COMPLAIN THAT 'S' COULD NOT BE FOUND
*
   COUT ( C );  SOUT ( S );  COUT ( C )
   ERRMSG ( " NOT FOUND&M" )
   RETURN
END
```

```
FUNCTION NPPROC()
*
* NEW-PROGRAM COMMAND
*
   XPPROC ( NEWPROGRAM :FRETURN )
   RETURN
END
```

```
FUNCTION NSPROC()
*
* DUMMY
*
   FRETURN
END
```

```
FUNCTION NUMERIC ( SCALAR C )
   RETURN '0' <= C AND C <= '9'
END
```

```
FUNCTION OUTBREAK ( SCALAR BLK, SCALAR LTE )
*
* REMOVE BREAKPOINT-TRAP PATCH
*
   SCALAR BYTE
   FIELD PATCH ( 0: 16,28 )
   IF LTE $ LTEFLAGS = 0 DO
      BYTE _ GETBYTE ( BLK, LTE $ LTECODEPTR )
      BYTE $ PATCH _ LTE $ LTEISAVE
      PUTBYTE ( BLK, LTE $ LTECODEPTR, BYTE )
   ENDIF
   RETURN
END
```

```
FUNCTION OUTTBUF ( REFERENCE PTR1 , REFERENCE PTR2 )
*
* OUTTBUF DELETES ALL LINES BETWEEN PTR1 AND PTR2 INCLUSIVE,
*  AND SAVES THE DELETED LINES IN THE TEXT BUFFER IF THEY FIT,
*  AND RETURNS TRUE. IF THE TEXT BUFFER WOULD HAVE
*  OVERFLOWED, NOTHING IS DONE, AND FALSE IS RETURNED.
*
*  FAILS IF PTR1 AND PTR2 ARE ARRANGED UNREASONABLY
*
   VECTOR OLD, NEW
   SCALAR OLDLENGTH, NEWLENGTH, START, FIN, REST, TL, BLK
   *
   UNLESS ACTIVE ( PTR1 ) DO CRASH()
   IF PTR1.BLOCK # PTR2.BLOCK OR PTR1.LINE > PTR2.LINE DO
      FRETURN
   ENDIF
   *
   TL _ PTR2.ENDX - PTR1.STARTX + 1
   IF TL > TSZ DO
      RETURN FALSE
   ENDIF
   TLENGTH _ TL
   TLINES _ PTR2.LINE - PTR1.LINE + 1
   *
   OLDLENGTH _ BLENGTH ( TXWINDOW ); OLD _ TXWINDOW.BARRAY
   NEWLENGTH _ OLDLENGTH - TLENGTH
   NEW _ SMAKE ( NEWLENGTH )
   *
   START _ PTR1.STARTX
   FIN _ START + TLENGTH
   REST _ NEWLENGTH - START
   *
   BCOPY ( @TARRAY[0] , @OLD[START] , TLENGTH )
   IF NEWLENGTH = 0 DO
      IF PTR1.BLOCK = GLOBLK DO
         ERRMSG ( "CAN'T DELETE ENTIRE GLOBAL BLOCK&N" )
         RETURN TRUE
      ENDIF
      BLKDESTROY ( TXSEG , PTR1.BLOCK )
      DOWNPTRS ( PTR1.BLOCK )
      CLRWINDOW ( TXWINDOW )
      MAKEINVALID ( GLOBALLY )
      RETURN TRUE
   ENDIF
   *
   BCOPY ( @NEW[0] , @OLD[0] , START )
   IF REST > 0 DO
      BCOPY ( @NEW[START] , @OLD[FIN] , REST )
   ENDIF
   SETWINDOW ( TXWINDOW , NEW )
   MAKEINVALID ( LOCALLY )
   *
   BLK _ PTR1 . BLOCK
   ZAPPTRS ( BLK , START , FIN )
```

```
    FIXPTRS ( BLK , FIN , -TLENGTH , -TLINES )
    RETURN TRUE
END
```

```
FUNCTION OUTWINDOW ( REFERENCE W )
*
* MOVES CONTENTS OF WINDOW W OUT OF MEMORY
*
    BUFWRITE ( W.BUFFER, 0 )
    BUFCLEAR ( W.BUFFER )
    W.BLOCK _ 0
    RETURN
END
```

```
FUNCTION PADLINE ( STRING L )
*
* PADLINE PADS LINE L OUT TO AN INTEGRAL NUMBER
*  OF WORDS USING NULS.
*
   GCD ( L )
   UNTIL ( LENGTH ( L ) MOD CPW  =  CPW - 1 ) DO
      WCI ( NUL , L )
   ENDUNTIL
   WCI ( CARRET , L )
   RETURN
END
```

```
FUNCTION PANICPROC()
*
* PROCESS USER PANIC
*
   MESSAGE ( "PANIC!&M" )
   RETURN
END
```

```
FUNCTION PCPROC()
*
* 'PC' COMMAND: PRINT CODE AND LTE CORRESPONDING TO LINE ADDRESS
*
   VECTOR A, S
   SCALAR FIRST, LAST, I, B, L
   *
   IF CMDMOVE ( IPTR1 :FRETURN ) DO
      S _ DBSEG
      B _ IPTR1.BLOCK
      L _ IPTR1.LINE
      IF LTEGET ( S, B, L :FRETURN ) DO
         *
         * PRINT LTE
         *
         SOUT ( "LTE = " )
         IOUT ( LTE $ LTEISAVE,,8 ); COUT ( ' ' )
         IOUT ( LTE $ LTEFLAGS,,8 ); COUT ( ' ' )
         IOUT ( LTE $ LTECODEPTR,,8 ); CRLF ( 1 )
         *
         * LOCATE CODE ( IF ANY )
         *
         FIRST _ LTE $ LTECODEPTR
         IF FIRST # LTENULL DO
            LAST _ LTENULL
            WHILE LAST = LTENULL DO
               L _ L + 1
               IF LTEGET ( S, B, L :FRETURN ) DO
                  LAST _ LTE $ LTECODEPTR
               ELSE DO
                  LAST _ BLKSIZE ( CDSEG, B ) * BPW
               ENDIF
            ENDWHILE
            *
            * PRINT CODE
            *
            SELWINDOW ( CCDWINDOW, CDSEG, B )
            A _ HALFVEC ( CCDWINDOW.BARRAY )
            FOR I _ FIRST TO LAST-1 DO
               IOUT ( A [ I ],,8 )
               CRLF ( 1 )
            ENDFOR
            CLRWINDOW ( CCDWINDOW )
         ENDIF
         RETURN
      ENDIF
   ENDIF
   FRETURN
END
```

```
FUNCTION PCTRPROC()
*
* P-COUNTER TRAP
*
   MESSAGE ( "P-COUNTER TRAP.&M" )
   RETURN
END
```

```
FUNCTION PDPROC()
*
* PRINT-DATA COMMAND
*
   SCALAR L, U, A, VU
   *
   L _ CSN ( CSTRING, 8 :FRETURN )
   U _ CSN ( CSTRING, 8 :L )
   VU _ DTSEG $ VDSIZE - 1
   IF U > VU DO U _ VU
   FOR A _ L TO U DO
      PWORD ( A , DTSEG )
   ENDFOR
   RETURN
END
```

```
FUNCTION PEEL ( STRING S )
*
* PEEL NEXT COMMAND FROM S, AND PUT IT IN CSTRING
*
   ITEM ( CSTRING, S, ';' )
   RETURN
END
```

```
FUNCTION PLADDR ( SCALAR FN, SCALAR LN )
*
* PRINTS LINE ADDR OF LINE 'LN' IN FUNCTION 'FN'
*
   STRING FUNC [ IDLENGTH ]
   *
   FNAME ( FN, FUNC )
   COUT ( '/' )
   SOUT ( FUNC )
   SOUT ( "/+" )
   IOUT ( LN - 1 )
   RETURN
END
```

```
FUNCTION PLPROC()
*
* 'PL' COMMAND: PRINT LINE NUMBER CORRESPONDING TO CODE ADDRESS
*
   STRING S [ IDLENGTH ]
   SCALAR A,L
          SCALAR PTBLK
          FRETURN
   *
   CMDARG ( S )
   IF PTBLK ( S, IPTR1 :FRETURN ) DO
      A _ CNS ( CSTRING, 8 )
      L _ LTEFIND ( IPTR1.BLOCK, A :FRETURN )
      IF L < 0 DO
         MESSAGE ( "BAD ADDR&N" )
      ELSE DO
         SOUT ( "LINE NUMBER = " )
         IOUT ( L )
         CRLF ( 1 )
      ENDIF
      RETURN
   ELSE DO
      FRETURN
   ENDIF
END
```

```
FUNCTION PSPROC()
*
*   PRINT-SEGMENT COMMAND
*
    BOOL OK
    SCALAR L, U, VU, RS
    MACRO C ( V ) _ ( V $ VDCAP )
    *
    CMDARG ( TARGET )
    OK _ TRUE
    IF LENGTH ( TARGET ) = 0 THEN WCI ( '0' , TARGET )
    RS _ TARGET . SDRS
    CSN ( TARGET, 8 : OK _ FALSE )
    IF ( NOT OK ) OR LENGTH ( TARGET ) # 0 THEN ;* NEW NAME
       TARGET . SDRS _ RS
       OPEN\WINDOWED ( C ( PSSEG ), TARGET : GOTO CANTOPEN )
       SCOPY ( PSNAME, TARGET )
       CMDARG ( TARGET )
    ELSE
       TARGET . SDRS _ RS
       OPEN\WINDOWED ( C ( PSSEG ), PSNAME : ..
             SCOPY ( TARGET, PSNAME ) & GOTO CANTOPEN )
    ENDIF
    L _ CSN ( TARGET, 8 : 0 )
    U _ CSN ( CSTRING, 8 : L )
    VU _ READ\LENGTH ( PSNAME ) - 1
    IF U > VU DO U _ VU
    FOR I _ L TO U DO
       PWORD ( I, PSSEG )
    ENDFOR
    CLOSE ( C ( PSSEG ) )
    RETURN
    *
  CANTOPEN:
    ERRMSG ( "CANT OPEN WINDOWED&M" )
    RETURN
END
```

```
FUNCTION PTBLK ( STRING NAME , REFERENCE POINTER )
*
* PTBLK SETS POINTER TO HEAD OF NAMED BLOCK.
*   THE BLOCK NAME IS LOOKED UP IN THE GLOBAL SYMBOL TABLE.
*   IF IT IS FOUND, THE POINTER IS SET TO THE 1ST LINE OF THE
*   BLOCK, AND TRUE IS RETURNED. IF THE NAME IS NOT FOUND,
*   FALSE IS RETURNED. IF THE GLOBAL SYMBOL TABLE IS INVALID,
*   THE FUNCTION FAILS.
*
   STRING BNAME
   SCALAR CHAR, BNUM
   REFERENCE ENTRY
   *
   SDCOPY ( BNAME, NAME )

   BNUM _ FNFIND ( BNAME :FRETURN )
   IF BNUM $ FNRUNT DO RETURN FALSE; ** TEMP **
   IF BNUM < 0 DO
      RETURN FALSE
   ELSE DO
      POINTER . BLOCK _ BNUM $ FNBLK
   ENDIF

   POINTER.SEGMENT _ TXSEG
   POINTER.LINE _ 1
   POINTER.STARTX _ 0
   UNLESS ACTIVE ( POINTER ) DO CRASH()
   SPANLINE ( POINTER )
   RETURN TRUE
END
```

```
FUNCTION PUSHWORD ( W )
*
* PUSH ONE WORD ONTO USER PROGRAM STACK
*
   SCALAR LTOP, SBASE
   *
   LTOP _ STATEWORD . SWLTOP
   SBASE _ STATEWORD . SWSBASE
   DTSEG [ LTOP + SBASE ] _ W
   STATEWORD . SWLTOP _ LTOP + 1
   RETURN
END
```

```
FUNCTION PUTBYTE ( SCALAR FN, SCALAR ADDR, SCALAR BYTE )
*
* PUT CODE BYTE INTO USER CODE SEGMENT AT
*  F-BASE REL ADDRESS IN FUNCTION FN
*
   VECTOR A
   *
   SELWINDOW ( CCDWINDOW, CDSEG, FN $ FNBLK )
   A _ HALFVEC ( CCDWINDOW.BARRAY )
   A [ ADDR ] _ BYTE
   OUTWINDOW ( CCDWINDOW )
   RETURN
END
```

```
FUNCTION PVPROC()

* PRINT VARIABLE COMMAND

   REFERENCE STE
   SCALAR FN, TYPE, ADDR, L, W1, W2
   BOOL FOUND
   VECTOR CSEG
   FIELD CBRUNT ( 0: 19,19 ), CBINDEX ( 0: 20,31 )
   FIELD LHALF ( 0: 0,15 ), RHALF ( 0: 16,31 )

   SCHECK
   CMDARG ( TARGET )

   FOUND _ FALSE
   UNLESS NOSTACK() DO
      FN _ GETFN ( ENV . SFCALLDATA )
      IF FN > 0 DO
         FOUND _ TRUE
         STE _ STEFIND ( TARGET, FN :FOUND _ FALSE & LEAVE )
      ENDIF
   ENDUNLESS

   IF NOT FOUND DO
      FOUND _ TRUE
      STE _ STEFIND ( TARGET, GLOBLK :FOUND _ FALSE &LEAVE )
   ENDIF

   IF FOUND DO
      TYPE _ STE . STYPE
      IF TYPE = SCONST DO
         L _ 1
         W1 _ STE . SVAL
      ELSEIF TYPE = SVAR DO
         ADDR _ STE . SVAL2
         IF STE . SLG DO
            ADDR _ ADDR + ENV . SFFBASE + STATEWORD . SWSBASE
         ENDIF
         IF STE . SVTYPE = SSCALAR DO
            L _ 1
            W1 _ DTSEG [ ADDR ]
         ELSE DO
            L _ 2
            W1 _ DTSEG [ ADDR ]
            W2 _ DTSEG [ ADDR+1 ]
         ENDIF
      ELSEIF TYPE = SFUNC DO
         L _ 1
         W1 _ STEFUNCNO ( STE )
      ELSEIF TYPE = SVCONST DO
         CSEG _ PROCSEG ((STE.SVAL2) $ CBRUNT )
         ADDR _ SEGSIZE ( CSEG ) -1 - (STE.SVAL2) $ CBINDEX
         L _ 2
         W1 _ CSEG [ ADDR ]
```

```
            W2 _ CSEG [ ADDR+1 ]
         ELSE DO
            FOUND _ FALSE
         ENDIF
      ENDIF

      IF NOT FOUND DO
         SOUT ( TARGET ); ERRMSG ( " NOT FOUND&M" )
         RETURN
      ENDIF

      IOUT ( W1$LHALF,,8 ); COUT ( ' ' ); IOUT ( W1$RHALF,,8 )

      IF L = 2 DO
         COUT ( ' ' )
         IOUT ( W2$LHALF,,8 ); COUT ( ' ' ); IOUT ( W2$RHALF,,8 )
      ENDIF

      CRLF()

      RETURN
END
```

```
FUNCTION PWORD ( A, VECTOR SEG )
*
* PRINT WORD FROM DATA SEGMENT AT ADDRESS 'A'
*
   IOUT ( A,, 8 ); SOUT ( ": " )
   IF SEG [ A ] = 0 DO
      MESSAGE ( "0&M" )
   ELSE DO
      IOUT ( SEG [ A ] $ LHALF,, 8 )
      COUT ( ' ' )
      IOUT ( SEG [ A ] $ RHALF,, 8 )
      CRLF ( 1 )
   ENDIF
   RETURN
END
```

```
FUNCTION QOVPROC()
*
* QUANTUM-OVERFLOW TRAP...SHOULD NOT EVER HAPPEN
*
   MESSAGE ( "QUANTUM-OVERFLOW TRAP !!!!&M" )
   RETURN
END
```

```
FUNCTION RELSTACK ( SCALAR RELATIVIZE )
*
* RELATIVIZE OR ABSOLUTIZE THE P-CNTRS IN SIPU USER PROG STACK
*
   FIELD SBFIELD ( 0: 0,15 ), LDFIELD ( 0: 0,15 )
   FIELD FBFIELD ( 0: 0,15 )
   FIELD RUNT ( 3: 0,0 ), BLCK ( 3: 7,15 )
   FIELD PCNTR ( 2: 0,15 ), PREVFR ( 3: 16,31 )
   FIELD REFSIZE ( 0: 10,15 )
   SCALAR STACKBASE, FRPTR, PCTR, FNCBASE
   VECTOR DSVEC, CDVEC, UCDVEC, RCDVEC
   REFERENCE FRAME
   CONSTANT FCMARKER _ 177777B

   DSVEC _ DTSEG
   RCDVEC _ RCDSEG
   UCDVEC _ CDSEG

   STACKBASE _ DSVEC [ 0 ] $ SBFIELD
   FRPTR  _    DSVEC [ 1 ] $ LDFIELD
   *
   WHILE NOT(FRPTR=FCMARKER) DO
      FRAME _ @ DSVEC [ FRPTR + STACKBASE ]
      FRAME $ REFSIZE _ 4
      IF FRAME.RUNT DO
         CDVEC _ RCDVEC
      ELSE DO
         CDVEC _ UCDVEC
      ENDIF
      *
      FNCBASE _(CDVEC [ FRAME.BLCK * 2 ] $ FBFIELD)*2
      *
      PCTR _ FRAME.PCNTR
      IF RELATIVIZE DO
         PCTR _ PCTR - FNCBASE
      ELSE DO
         PCTR _ PCTR + FNCBASE
      ENDIF
      FRAME.PCNTR _ PCTR
      *
      FRPTR _ FRAME.PREVFR
   ENDWHILE
   RETURN
END
```

```
FUNCTION RFPROC()

* READ-FUNCTIONS COMMAND
*  READS ALL FUNCTIONS FROM SPECIFIED PROGRAM INTO
*  CURRENT PROGRAM, REPLACING ANY DUPLICATED
*  FUNCTIONS, AND APPENDING THE OTHERS. COMPLAINS IF
*  ATTEMPT IS MADE TO REPLACE A RUNTIME FUNCTION

   MACRO C(V) _ V$VDCAP
   STRING APNAME [ FNAMEMAX ], FNAME [ IDLENGTH ]
   REFERENCE ENTRY
   SCALAR I, J
   BOOL FOUND

   PCHECK
   IF INVALID ( GLOBLK ) DO
      SOUT ( PROGNAME ); ERRMSG ( " GLOBALLY INVALID&M" )
      RETURN
   ENDIF

   CMDARG ( APNAME )

   * KLUDGE: SHOULD USE EDITOR'S "SEGNAME" FUNCTION HERE
      WCD ( 'T', APNAME )
      OPEN\WINDOWED ( C(AUXTXSEG), APNAME :FRETURN )
      GCI ( APNAME )
      WCD ( 'D', APNAME )
      OPEN\WINDOWED ( C(AUXDBSEG), APNAME :FRETURN )
      GCI ( APNAME )
   * ENDKLUDGE

   IF NOGOOD ( AUXTXSEG, GLOBLK ) DO
      SOUT ( APNAME ); ERRMSG ( " GLOBALLY INVALID&M" )
      RETURN
   ENDIF

   FOR I _ 2 TO DIRSIZE(AUXTXSEG)-1 DO
      FUNCNAME ( I, FNAME, AUXDBSEG )
      SOUT ( FNAME )
      FOUND _ TRUE
      ENTRY _ STEFIND ( FNAME, GLOBLK :FOUND _ FALSE & LEAVE )
      IF FOUND DO
         J _ STEFUNCNO ( ENTRY )
         IF J $ FNRUNT DO SOUT ( " NOT" )
         MESSAGE ( " REPLACED&M" )
      ELSE DO
         J _ NBLOCKS + 1
         BLKCREATE ( TXSEG, J )
         MESSAGE ( " APPENDED&M" )
      ENDIF
      UNLESS J $ FNRUNT DO
         SELWINDOW ( TXWINDOW, AUXTXSEG, I )
         MOVEWINDOW ( TXWINDOW, TXSEG, J )
         OUTWINDOW ( TXWINDOW )
```

```
      ENDUNLESS
   ENDFOR

   MAKEINVALID ( GLOBLK )
   CLOSE ( C(AUXTXSEG) )
   CLOSE ( C(AUXDBSEG) )
   RETURN
END
```

```
FUNCTION RSPROC()
*
* RECOVER-STATE COMMAND
*
   MACRO C(V) _ V $ VDCAP
   *
   PCHECK
   RCHECK
   GCHECK
   OPEN\WINDOWED ( C(DTSEG), DATANAME :GOTO BADDATA )
   OPEN\WINDOWED ( C(CLSEG), CAPNAME :GOTO BADCAP )

   IF GNBCI ( CSTRING :-1 ) = 'R' DO RELSTACK ( TRUE )

   USERTRAPPED _ TRUE
   HAVE\STATE _ TRUE
   *
   RETURN
   *
  BADDATA:
   SOUT ( DATANAME ); ERRMSG ( " MISSING&M" )
   RETURN
  BADCAP:
  CLOSE ( C(DTSEG) )
   SOUT ( CAPNAME ); ERRMSG (   " MISSING&M" )
   RETURN
END
```

```
FUNCTION RTDROP()
*
* GET RID OF CURRENT RUNTIME
*
   MACRO C(V) _ V $ VDCAP
   CLEAR ( RUNTNAME )
   IF RTLOADED DO
      CLOSE ( C(RTXSEG) )
      CLOSE ( C(RCDSEG) )
      CLOSE ( C(RDBSEG) )
   ENDIF
   RTLOADED _ FALSE
   RETURN
END
```

```
FUNCTION RTPROC()

* REPLACE-TEXT COMMAND

   PCHECK

   IF CmDMOVE ( IPTR1 :FRETURN ) DO
      SETPOINTER ( CURRENT, IPTR1 )
      IF CMDMOVE ( IPTR2 :FRETURN ) DO
         ACTIVE ( CURRENT )
         INTBUF ( TRUE, CURRENT )
         BACK ( CURRENT )
         OUTWINDOW ( TXWINDOW )
         UNLESS OUTTBUF ( IPTR1, IPTR2 :GOTO BAD ) DO
            ERRMSG ( "TEXT BUFFER WOULD OVERFLOW&M" )
         ENDUNLESS
      ENDIF
   ENDIF
   RETURN

  BAD:
   ERRMSG ( "INVALID POINTER ARRANGEMENT&M" )
   RETURN
END
```

---

```
FUNCTION RTSELECT ( STRING RTNAME )
*
* SELECT RUNTIME LIBRARY
*
   MACRO C(V) _ V $ VDCAP

   IF RUNFLAG DO
      ERRMSG ( "ILLEGAL IN RUNTIME MODE&M" )
      RETURN
   ENDIF
   *
   IF LOADED AND STREQL ( RTNAME , PROGNAME ) DO
      ERRMSG ( "SAME AS USER PROGRAM&M" )
      RETURN
   ENDIF
   *
   IF LENGTH ( RTNAME ) > PNAMEMAX  DO
      ERRMSG ( "NAME TOO LONG&M" )
      RETURN
   ENDIF
   *
   RTDROP()
   *
   MAKE\RUNTIME ( RTNAME )
   *
   OPEN\WINDOWED ( C(RTXSEG), RTEXTNAME :GOTO BADTEXT )
   IF RTINVALID ( GLOBLK ) DO
      SOUT ( RTNAME )
      ERRMSG ( " GLOBALLY INVALID&M" )
      GOTO BADCODE
   ENDIF
   OPEN\LOCKED ( C(RCDSEG), RCODENAME :GOTO BADCODE )
   OPEN\WINDOWED ( C(RDBSEG), RDEBUGNAME :GOTO BADDEBUG )
   *
   RTLOADED _ TRUE
   VERSION\CHECK()
   RETURN
   *
  BADFRIEND:
   CLRWINDOW ( GDBWINDOW )
   CLOSE ( C(RDBSEG) )
  BADDEBUG:
   CLOSE ( C(RCDSEG) )
  BADCODE:
   CLOSE ( C(RTXSEG) )
  BADTEXT:
   ERRMSG ( "BAD RUNTIME&M" )
   CLEAR ( RUNTNAME )
   RETURN
END
```

---

```
FUNCTION RUN()
*
* EXINIT INITIALIZES THE LANGUAGE SYSTEM.
*
    *
    *
    * INITIALIZE BUFFERS
    *
    BUFINIT()
    *
    * INITIALIZE LINE-COLLECTOR
    *
    INITLINE()
    *
    * SETUP EDITOR AND DEBUGGER
    *
    LSINIT()
    *
    * GO PROCESS FIRST COMMAND
    *
    COMMANDS()

    RETURN
END
```

```
FUNCTION RUN\SCOM( FN )

* RUN COMPILER FUNCTION 'FN'

   REFERENCE STATEWORD
   SCALAR HDR, NWL, NWA, LBASE, FBASE
   MACRO C(V) _ V$VDCAP
   MACRO PUSH(X) _ ((X)&CODE(4114B))
   MACRO TRAP(N) _ ((N)&CODE(4164B,44106B,1,4100B))

   STATEWORD _ SETREFSIZE ( @DTSCOM[0], SW\SIZE )

   LBASE _ STATEWORD . SWLTOP

   HDR _ BLKHEAD ( CDSCOM, FN )
   NWL _ HDR $ HDRNL - FRSIZE
   NWA _ -( HDR $ HDRNFA + 1 )
   FBASE _ LBASE - NWA

   SPUSH ( 0 ); SPUSH ( 0 )
   SPUSH ( INST1 @ LHALF + FBASE @ RHALF )
   SPUSH ( FN @ LHALF + FCMARKER @ RHALF )
   STATEWORD . SWLBASE _ LBASE
   STATEWORD . SWLTOP _ STATEWORD . SWLTOP + NWL
   STATEWORD . SWFLAGS _ 3000B

   PUSH ( -1 )
   PUSH ( C(RCDSCOM) )
   PUSH ( C(CDSCOM) )
   PUSH ( C(DTSCOM) )
   PUSH ( C(CLSCOM) )

   TRAP ( 26 )

   RETURN
END
```

```
FUNCTION SBPROC()
*
* SET-BREAKPOINT COMMAND
*
*       NOTE: DOES NOT FIX SOURCE
*             DOES NOT HANDLE MULT LINES
*
   UNLESS CMDMOVE ( IPTR1 :FRETURN ) DO RETURN
   *
   LTEGET ( DBSEG, IPTR1.BLOCK, IPTR1.LINE :FRETURN )
   IF LTE $ LTECODEPTR = LTENULL DO
      ERRMSG ( " NOT A STATEMENT-LINE&M" )
   ELSEIF LTE $ LTEBREAK DO
      ERRMSG ( "BREAKPOINT ALREADY SET&M" )
   ELSE DO
      LTE _ INBREAK ( IPTR1.BLOCK, LTE )
      LTE $ LTEBREAK _ TRUE
      LTEPUT ( DBSEG, IPTR1.BLOCK, IPTR1.LINE :CRASH() )
   ENDIF
   RETURN
END
```

---

```
FUNCTION SCAN(PROC MATCH,PROC MOVE,STRING TARGET,REFERENCE PTR)
*
* SCAN LOOKS FOR A LINE CONTAINING THE TARGET STRING, USING THE
*  MATCHING PROCEDURE PASSED BY THE CALLER. IF A LINE IS FOUND,
*  PTR IS LEFT ACTIVE AND POINTING TO THE LINE. IF NO LINE IS
*  FOUND, PTR IS LEFT ACTIVE AND UNCHANGED. IF PTR WAS NULL,
*  THE FUNCTION FAILS.
*  NOTE:   MATCH = AMATCH OR LMATCH
*          MOVE  = FWD OR BACK
*
   REFERENCE SS
   *
   STRING PTARGET [ LSZ ], LTARGET

   UNLESS ACTIVE ( PTR ) DO FRETURN
   SETPOINTER ( SEARCH , PTR )
   MOVE ( SEARCH :RETURN FALSE )
   *
   * MAIN SEARCH LOOP
   *
   SS _ @ ( SEARCH . STR )
   SDCOPY ( LTARGET, TARGET )
   POLISH ( LTARGET, PTARGET ); GCD ( PTARGET )
   UNTIL MATCH ( PTARGET , SS ) DO
      ADVANCE ( MOVE, SEARCH , PTR :RETURN FALSE )
   ENDUNTIL
   *
   * FALL THROUGH MEANS SUCCESS
   *
   SETPOINTER ( PTR , SEARCH )
   RETURN TRUE
END
```

```
FUNCTION SDPROC()
*
* STORE-DATA COMMAND
*
   SCALAR A, L, R, T
   *
   A _ CSN ( CSTRING, 8 :FRETURN )
   IF A > DTSEG $ VDSIZE - 1 DO
      ERRMSG ( "ADDRESS TOO LARGE&M" )
   ELSE DO
      L _ CSN ( CSTRING, 8 :FRETURN )
      R _ CSN ( CSTRING, 8 :T_L & L_0 & T )
      DTSEG [ A ] $ LHALF _ L
      DTSEG [ A ] $ RHALF _ R
      PWORD ( A , DTSEG )
   ENDIF
   RETURN
END
```

```
FUNCTION SED2()

* TEST-MODE FRONT-END

   SCOPY ( PROMPT, "    :" )
   RUN()
   RETURN
END
```

```
FUNCTION SEGINIT(VECTOR SEG)

*INITIALIZE GIVEN SEGMENT

        SSEGSIZE ( SEG, DIRESIZE )
        BLKBASE(SEG,0) _ 0
        BLKSIZE(SEG,0) _ DIRESIZE
        DIRSIZE(SEG) _ 1
        RETURN
END
```

```
FUNCTION SEGNAME ( STRING NAME, STRING TAIL, STRING PROG )
*
* FORM SEGMENT NAME FROM 'PROG' AND 'TAIL', AND
*  RETURN IT IN 'NAME'
*
   SCOPY ( NAME, TAIL )
   APPEND ( NAME, PROG )
   RETURN
END
```

```
FUNCTION SELECTPROGRAM ( STRING PNAME )
*
* SELECTPROGRAM DROPS THE CURRENT PROGRAM AND PICKS UP
*  A NEW PROGRAM NAMED 'PNAME'. IT FAILS IF THERE IS
*  NO SUCH PROGRAM.
*
   MACRO C(V) _ V $ VDCAP
   IF LENGTH ( PNAME ) > PNAMEMAX DO
      ERRMSG ( "NAME TOO LONG&M" )
      RETURN
   ENDIF
   *
   IF RTLOADED AND STREQL ( PNAME , RUNTNAME ) DO
      ERRMSG ( "SAME AS RUNTIME&M" )
      RETURN
   ENDIF
   *
   DROP()
   *
   MAKE\PROGRAM ( PNAME )
   *
   OPEN\WINDOWED ( C(TXSEG), TEXTNAME :CLEAR(PROGNAME)&FRETURN )

   READ\LENGTH ( CODENAME :FRESH\SEGS() )
   READ\LENGTH ( DEBUGNAME :FRESH\SEGS() )

   OPEN\WINDOWED ( C(CDSEG), CODENAME :CRASH() )
   OPEN\WINDOWED ( C(DBSEG), DEBUGNAME :CRASH() )
   *
   NEWPOINTER ( CURRENT , GLOBLK , 1 , 0 )
   OUTWINDOW ( TXWINDOW )
   *
   LOADED _ TRUE
   VERSION\CHECK()
   RETURN
END
```

```
FUNCTION SELWINDOW ( REFERENCE W , VECTOR SEG , SCALAR BLK )
*
* SELECTS CONTENTS OF WINDOW W
*
   SCALAR BUF
   *
   BUF _ W.BUFFER
   W.SEGMENT _ SEG
   W.BLOCK _ BLK
   BUFSL ( SEG , BLK , BUF, 0 )
   W.BARRAY _ BUFBASE [ BUF ]
   RETURN
END
```

```
FUNCTION SETINVALID ( SCALAR BLK , BOOL VAL )
*
* SET INVALID BIT OF BLK TO VAL
*
   ( BLKINFO ( TXSEG, BLK ) ) $ BLKINVALID _ VAL
   RETURN
END
```

---

```
FUNCTION SETTSIZE ( SCALAR N )
*
* SET SIZE OF TEXT BUFFER TO N
*
   SCALAR I
   *
   FOR I _ 0 TO NWINS-1 DO
      OUTWINDOW ( WINADDR ( I ) )
   ENDFOR
   *
   SFREE ( TARRAY )
   TSZ _ N
   TARRAY _ SMAKE ( TSZ )
   TLENGTH _ TLINES _ 0
   *
   RETURN
END
```

```
FUNCTION SETUP\SCOM()

* GET COMPILER READY TO EXECUTE

   MACRO C(V) _ V$VDCAP
   MACRO PUSH(X) _ ((X)&CODE(4114B))
   MACRO TRAP(N) _ ((N)&CODE(4164B,44106B,1,4100B))
   MACRO MYCLIST ( I ) _ ( PUSH ( I ) & TRAP ( 30 ) )
   REFERENCE STATEWORD

   PUSH ( 0 ); TRAP ( 28 )

   OPEN\LOCKED ( C(DTSCOM), "$SCOM2" )
   MYCLIST ( C ( CLSCOM ) )
   OPEN\LOCKED ( C(CDSCOM), "CSCOM2" )
   OPEN\LOCKED ( C(RCDSCOM), "CSRUN" )

   STATEWORD _ SETREFSIZE ( @DTSCOM[0], SW\SIZE )
   STATEWORD . SWLTOP _ 0
   STATEWORD . SWSBASE _ BLKINFO ( CDSCOM, GLOBLK )

   RETURN
END
```

---

```
FUNCTION SETWINDOW ( REFERENCE W , VECTOR NEW )
*
* SETWINDOW MAKES ARRAY NEW BE THE CONTENTS OF WINDOW W.
*
   BUFCLEAR ( W.BUFFER )
   BUFSET ( W.SEGMENT , W.BLOCK , W.BUFFER , NEW )
   W.BARRAY _ NEW
   RETURN
END
```

```
FUNCTION SFREE ( VECTOR OBJECT )
*
* STANDARD FREE FUNCTION
*
   IF OBJECT $ VDBASE # 0 DO
      FREE ( FREEPOOL, OBJECT )
   ENDIF
   RETURN
END
```

```
FUNCTION SMAKE ( SCALAR LENGTH ) RETURNING VECTOR
*
* STANDARD MAKE FUNCTION
*
   VECTOR NULLVEC
   *
   NULLVEC $ VDBASE _ 0
   *
   IF LENGTH > 0 DO
      RETURN MAKE ( FREEPOOL, LENGTH )
   ELSE DO
      RETURN NULLVEC
   ENDIF
END
```

```
FUNCTION SMAKE\STRING ( STRING S, NCHARS )
*
* ALLOCATE 'NCHARS' BYTES OF STORAGE FOR
*  STRING S FROM STANDARD FREEPOOL
*
   VECTOR SVEC
   *
   SVEC _ SMAKE ( ( NCHARS + CPW - 1 ) / CPW )
   SVEC _ BYTEVEC ( SVEC )
   SVEC _ SUBVEC ( SVEC, 0, NCHARS )
   SETSTRING ( S, SVEC )
   CLEAR ( S )
   RETURN
END
```

```
FUNCTION SPANLINE ( REFERENCE POINTER )
*
* SPANLINE TAKES AN ACTIVE POINTER WITH STARTX SET, AND
*  SCANS TO THE END OF THE LINE, SETTING ENDX AND STR.
*
   SCALAR I, J, L, RP
   VECTOR A
   *
   A _ TXWINDOW.BARRAY
   L _ BLENGTH ( TXWINDOW )
   I _ J _ POINTER.STARTX
   WHILE MORELINE ( A [ J ] ) DO
      J _ J + 1
   ENDWHILE
   POINTER.ENDX _ J
   MAKESTRING ( @(POINTER.STR), A, I, J )
   RETURN
END
```

```
FUNCTION SPPROC()
*
* SELECT-PROGRAM COMMAND
*
   XPPROC ( SELECTPROGRAM :FRETURN )
   RETURN
END
```

```
FUNCTION SPUSH ( W )

* PUSH ONE WORD ONTO COMPILER'S STACK

   SCALAR LTOP, SBASE
   REFERENCE STATEWORD

   STATEWORD _ SETREFSIZE ( @DTSCOM[0], SW\SIZE )
   LTOP _ STATEWORD . SWLTOP
   SBASE _ STATEWORD . SWSBASE
   DTSCOM [ LTOP + SBASE ] _ W
   STATEWORD . SWLTOP _ LTOP + 1
   RETURN
END
```

```
FUNCTION SRPROC()
*
* SELECT-RUNTIME-LIBRARY COMMAND
*
   XPPROC ( RTSELECT :FRETURN )
   RETURN
END
```

```
FUNCTION STEFIND ( STRING NAME, SCALAR FD ) RETURNING REFERENCE
*
* STEFIND: FIND ENTRY IN SYMBOL TABLE
*  LOCATES SYMBOL TABLE FOR FUNCTION DESCRIBED BY 'FD'.
*  IT LOOKS UP 'NAME' AND RETURNS A
*  REFERENCE TO THE SYMBOL TABLE ENTRY FOUND.
*
*  FAILS: IF SYMBOL NOT FOUND
*
   SCALAR INDEX
   REFERENCE THISSTE
   *
   SELWINDOW ( GDBWINDOW, DBUGSEG ( FD $ FNRUNT ), FD $ FNBLK )
   INDEX _ LOOKUP ( NAME :GOTO NOSUCH )
   THISSTE _ @ ( GDBWINDOW.BARRAY ) [ INDEX ]
   BCOPY ( STE, THISSTE, SDISP )
   CLRWINDOW ( GDBWINDOW )
   RETURN STE
  NOSUCH:
   CLRWINDOW ( GDBWINDOW )
   FRETURN
END
```

```
FUNCTION STEPPROC()
*
* INSTRUCTION STEP TRAP
*
   MESSAGE ( "INSTRUCTION-STEP TRAP.&M" )
   RETURN
END
```

```
FUNCTION STLK(BUFNO,VECTOR NAME,SIZE)

*LOOK UP SIZE WORD SYMBOL NAME STARTING AT NAME
*IN DEBUGGER BLOCK BUFFER NUMBER BUFNO
*FAILS IF NOT FOUND, SUCCEEDS AND RETURNS SYMBOL TABLE RELATIVE
*DISPLACEMENT OF ENTRY IF FOUND

       DI ENTRY,I
       DV ENAME, BUF
*COMPUTE HASH CODE
       BUF _ BUFBASE[BUFNO]
       ENTRY _ ABS(NAME[0] REM BUFREF[BUFNO].HCSIZE) + HCBASE
*SCAN CHAIN FOR NAME
       FOR ENTRY _ BUF[ENTRY]$SCHAIN WHILE ENTRY # 0 DO
*COMPARE SIZE OF NAMES FOR EQUALITY
          IF BUF[ENTRY]$SSIZE # SIZE THEN GOTO STLK1
*COMPARE NAMES FOR EQUALITY
          ENAME _ SUBVEC(BUF,ENTRY+SDISP,SIZE)
          FOR I _ SIZE-1 BY -1 TO 0 DO
             IF NAME[I] # ENAME[I] THEN GOTO STLK1
          ENDFOR
          RETURN ENTRY
STLK1: ENDFOR
       FRETURN
END
```

```
FUNCTION STOVPROC()
*
* STACK-OVERFLOW TRAP
*
   MESSAGE ( "STACK OVERFLOW TRAP.&M" )
   RETURN
END
```

```
FUNCTION STPROC()
*
* DUMP SYMBOL-TABLE-ENTRY
*
   REFERENCE STE
   SCALAR FD
   STRING S [ IDLENGTH ]
   BOOL FOUND
   *
   CMDARG ( S )
   *
   FOUND _ FALSE
   FD _ GETFN ( ENV . SFCALLDATA )
   IF FD > 0 DO
      FOUND _ TRUE
      STE _ STEFIND ( S, FD :FOUND _ FALSE & LEAVE )
   ENDIF
   *
   IF NOT FOUND DO
      FOUND _ TRUE
      STE _ STEFIND ( S, GLOBLK :FOUND _ FALSE & LEAVE )
   ENDIF
   *
   IF FOUND DO
      IF STE . SPD DO SOUT ( "SYSTEM " )
      IF STE . SLG DO
         SOUT ( "LOCAL, ")
      ELSE DO
         SOUT ( "GLOBAL, " )
      ENDIF
      LVOUT ( "STYPE = ", STE.STYPE )
      LVOUT ( ", SVTYPE = ", STE.SVTYPE )
      LVOUT ( ", SRTYPE = ", STE.SRTYPE )
      LVOUT ( ", SVAL1 = ", STE.SVAL1 )
      LVOUT ( ", SVAL2 = ", STE.SVAL2 )
      CRLF ( 1 )
   ELSE DO
      SOUT ( S )
      ERRMSG ( " NOT FOUND&M" )
   ENDIF
   RETURN
END
```

```
FUNCTION STREQL ( STRING S1 , STRING S2 )
*
* STREQL COMPARES TWO STRINGS FOR EQUALITY
*
   STRING T1, T2
   *
   IF LENGTH ( S1 ) = LENGTH ( S2 ) DO
      SDCOPY ( T1, S1 ); SDCOPY ( T2, S2 )
      WHILE GCI ( T1 :RETURN TRUE ) = GCI ( T2 :RETURN TRUE ) DO
      ENDWHILE
   ENDIF
   RETURN FALSE
END
```

```
FUNCTION TBFILL ( PROC GETLINE )
*
* FILL TEXT BUFFER USING GETLINE
*
   SCALAR TL
   VECTOR LNVEC [ LSZW ]
   SCALAR LNLENGTH
   STRING LNSTR
   BOOL ESCAPE
   *
   TLENGTH _ TLINES _ 0
   *
   MAKESTRING ( LNSTR, LNVEC, 0, LSZW-1 )
   ESCAPE _ FALSE
   *
   UNTIL ESCAPE DO
      ESCAPE _ GETLINE ( LNSTR :CRASH() )
      UNLESS ( ESCAPE AND GCNI(LNSTR) = CARRET ) DO
         PADLINE ( LNSTR )
         LNLENGTH _ LENGTH ( LNSTR ) / CPW
         TL _ TLENGTH + LNLENGTH
         IF TL > TSZ DO
            ERRMSG ( "FULL!&M" )
            RETURN
         ELSEIF TL > TSZ - LSZW DO
            ERRMSG ( "ALMOST FULL&M" )
         ENDIF
         BCOPY ( @TARRAY[TLENGTH], @LNVEC[0], LNLENGTH )
         TLENGTH _ TL
         TLINES _ TLINES + 1
      ENDUNLESS
   ENDUNTIL
   RETURN
END
```

```
FUNCTION TRAPS()
*
* HANDLE USER PROGRAM TRAP OR USER TERMINAL PANIC
*
   CONSTANT NTRAPS _ 13
   REFERENCE TOP
   SCALAR LN, NB, FB, I, FN
   PROC P
   *
   * TRAP TABLE
   *
   *
   VECTOR TRAPPROC [ NTRAPS ] _ ..
      ATTNPROC, QOVPROC, PCTRPROC, BKPTPROC, ..
      STEPPROC, WINPROC, STOVPROC, NOGOPROC, ..
      FTRCPROC, BCOPPROC, BADOPPROC, ARITHPROC, ..
      PANICPROC
   *
   TOP _ FIND\FRAME ( STATEWORD . SWLBASE )
   ENV _ TOP
   FN _ GETFN ( TOP . SFCALLDATA )
   NB _ DIRSIZE ( PROCSEG ( FN $ FNRUNT ) ) - 1
   FB _ FN $ FNBLK
   IF FB < 2 OR NB < FB DO
      ERRMSG ( "BAD STACK&M" )
   ELSE DO
      *
      * SET ACTIVE-POINTER ("*") HERE
      *
      LN _ LTEFIND ( FN, TOP.SFPCTR-1 :CRASH() )
      * NOTE DISGUSTING '-1' ABOVE
      *
      CRLF ( 1 )
      PLADDR ( FN, LN ); SOUT ( ": " )
      UNLESS FN $ FNRUNT DO
      CURRENT.SEGMENT _ TXSEG
         CURRENT.BLOCK      _ FB
         CURRENT.LINE       _ 1
         CURRENT.STARTX  _ 0

         UNLESS ACTIVE ( CURRENT ) DO CRASH()
         SPANLINE ( CURRENT )
         FOR I _ 1 TO LN-1 DO
            FWD ( CURRENT :CRASH() )
         ENDFOR
      ENDUNLESS
   ENDIF
   CSFR.DOSAVEDCODE _ FALSE;          *(TENTATIVELY ASSUME NO PATCH)
   *
   P _ TRAPPROC [ STATEWORD . SWTRAPCLASS ]
   P()
   *
   USERTRAPPED _ FALSE
   *
```

```
    RETURN
END
```

```
FUNCTION TTPROC()
*
* TTPROC TYPES OUT THE CONTENTS OF THE TEXT BUFFER
*
   SCALAR I , J , L
   STRING TSTRING
   *
   I _ L _ 0
   UNTIL ( L = TLINES ) DO
      J _ I
      WHILE ( MORELINE ( TARRAY[J] )) DO
         J _ J + 1
      ENDWHILE
      MAKESTRING ( TSTRING, TARRAY, I, J )
      UNPAD ( TSTRING )
      *
      OUTLINE ( TSTRING :CRASH() )
      *
      PADLINE ( TSTRING )
      I _ J + 1
      L _ L + 1
   ENDUNTIL
   RETURN
END
```

```
FUNCTION UNPAD ( STRING L )
*
* UNPAD REMOVES NUL PADDING FROM END OF LINE L.
*
   SCALAR CHAR
   *
   GCD ( L )
   WHILE ( ( CHAR _ GCD ( L :GOTO EMPTY )) = NUL ) DO
   ENDWHILE
   WCI ( CHAR , L )
  EMPTY:
   WCI ( CARRET , L )
   RETURN
END
```

```
FUNCTION UPPTRS ( SCALAR B )
*
* UPPTRS ADJUSTS POINTERS DUE TO INSERTION OF NEW FUNCTION
*   BLOCK B.
*
   REFERENCE P
   SCALAR I
   *
   FOR I _ 0 TO NPTRS - 1 DO
      P _ PTRADDR ( I )
      IF P.BLOCK >= B DO
         P.BLOCK _ P.BLOCK + 1
      ENDIF
   ENDFOR
   RETURN
END
```

```
FUNCTION VERSION\CHECK()

* COMPARE VERSION NUMBERS OF CURRENT RUNTIME AND RUNTIME
*  USED IN LAST GLOBAL RECOMPILATION

   SCALAR UV, RV, RGLOBLK

   RGLOBLK _ TRUE @ FNRUNT + GLOBLK @ FNBLK
   IF LOADED AND RTLOADED AND NOT INVALID ( GLOBLK ) DO
      UV _ STEFIND ( "%VERSION", GLOBLK :GOTO FUSS ) . SVAL
      RV _ STEFIND ( "%VERSION", RGLOBLK :GOTO FUSS ) . SVAL
      UNLESS UV = RV DO
         ERRMSG ( "WARNING: RUNTIME VERSION MISMATCH&M" )
      ENDUNLESS
   ENDIF
   RETURN

  FUSS:
   ERRMSG ( "%VERSION UNDEFINED&M" )
   RETURN
END
```

```
FUNCTION WHPROC()

* WHERE COMMAND

   REFERENCE P

   P _ CMDPTR ( :FRETURN )
   IF NULL ( P ) DO
      MESSAGE ( "NULL&M" )
   ELSE DO
      PLADDR ( P.BLOCK, P.LINE )
      CRLF()
   ENDIF
   RETURN
END
```

```
FUNCTION WINPROC()
*
* WINDOW TRAP
*
   MESSAGE ( "WINDOW TRAP.&M" )
   RETURN
END
```

```
FUNCTION XPPROC ( PROC GETPROGRAM )
*
* XPPROC EXTRACTS PROGRAM NAME FROM COMMAND LINE,
*  AND GETS THAT PROGRAM USING THE FUNCTION PASSED,
*  WHICH SHOULD BE EITHER NEWPROGRAM, SELECTPROGRAM, OR
*  RTSELECT.
*
   SCALAR CHAR
   *
   CMDARG ( TARGET )
   GETPROGRAM ( TARGET :ERRMSG ( "BAD PROGRAM NAME&M" ) )
   RETURN
END
```

```
FUNCTION XTPROC( PROC DOTBUF )
*
* LOAD/DELETE TEXT
*
   IF CMDMOVE ( IPTR1 :FRETURN ) DO
      SETPOINTER ( CURRENT, IPTR1 )
      IF CMDMOVE ( IPTR2 :FRETURN ) DO
         ACTIVE ( CURRENT ); BACK ( CURRENT )
         IF DOTBUF ( IPTR1 , IPTR2 :GOTO BAD ) DO
            RETURN TRUE
         ELSE DO
            ERRMSG ( "TEXT BUFFER WOULD OVERFLOW&M" )
         ENDIF
      ENDIF
   ENDIF
   RETURN FALSE
   *
  BAD:
   ACTIVE ( CURRENT ); FWD ( CURRENT )
   ERRMSG ( "INVALID POINTER ARRANGEMENT&M" )
   RETURN FALSE
END
```

```
FUNCTION ZAPPTRS ( SCALAR BLK , SCALAR START , SCALAR FIN )
*
* ZAPPOINTERS SETS TO NULL ALL TEXT POINTERS WHICH POINT INTO
*  THE DELETED TEXT IN BLK BETWEEN WORDS START AND FIN-1,
*  INCLUSIVE.
*
   SCALAR I
   REFERENCE P
   *
   FOR I _ 0 TO NPTRS-1 DO
      P _ PTRADDR ( I )
      IF P.BLOCK = BLK DO
         IF START <= P.STARTX AND P.STARTX < FIN DO
            MAKENULL ( P )
         ENDIF
      ENDIF
   ENDFOR
   RETURN
END
```

```
FUNCTION INITLINE()
*
* INITIALIZE THE LINE COLLECTOR
*
   SCALAR I
   *
   LCINIT _ TRUE
   *
   FOR I _ 0 TO LSZ - 1 DO
      TABS [ I ] _ FALSE
   ENDFOR
   *
   FOR I _ 3 BY 3 TO LSZ-1 DO
      TABS [ I ] _ TRUE
   ENDFOR
   *
   OLDLINE _ LINE1
   NEWLINE _ LINE2
   *
   RETURN
   *
END
```

```
FUNCTION OUTLINE ( REFERENCE PL )
*
* OUTLINE ACCEPTS SYSTEXT LINE 'L' FROM THE CALLER AND
*  OUTPUTS IT TO THE TERMINAL.
*  OUTLINE FAILS IF:
*     1) 'L' DOES NOT END WITH CR.
*     2) 'L' IS TOO LONG.
*     3) LINE COLLECTOR NEVER INITIALIZED
*

   STRING L

   UNLESS LCINIT DO FRETURN
   SDCOPY ( L, PL )
   UNPOLISH ( L , HOLD    :FRETURN )
*
   SOUT ( HOLD ); CRLF()
   SRETURN
END
```

```
FUNCTION EDITLINE ( REFERENCE L )
*
* EDITLINE ACCEPTS A SYSTEXT LINE FROM THE CALLER
*  AND MAKES IT THE OLD LINE. A NEWLINE IS THEN
*  COLLECTED AND RETURNED. EDITLINE FAILS IF:
*     1) 'L' DOES NOT END WITH CR.
*     2) 'L' IS  TOO LONG.
*     3) NEW LINE IS TOO LONG TO FIT IN STORAGE OF 'L'
*     4) LINE COLLECTOR NEVER INITIALIZED
*
   UNLESS LCINIT DO FRETURN
   *
   UNPOLISH ( L , OLDLINE :FRETURN )
   *
   SRETURN INLINE ( L :FRETURN )
   *
END
```

```
FUNCTION INLINE ( REFERENCE L )
*
* INLINE ASSEMBLES LINE 'L' FROM THE TERMINAL, UP
*  TO A CARRIAGE-RETURN OR CARRIAGE-RETURN-ESCAPE,
*  AND RETURNS FALSE OR TRUE, RESPECTIVELY.
*  'L' SHOULD LOCATE A STRING DESCRIPTOR POINTING TO STORAGE
*  FOR THE NEW LINE.
*  INLINE FAILS IF:
*     1) NEW LINE IS TOO LONG TO FIT IN STORAGE OF 'L'
*     2) LINE COLLECTOR NEVER INITIALIZED.
*
   VECTOR CTLKEY [ 33 ] _ ..
       NOOP,NOOP, ..
       CCOPY,TSKIP,NSKIP,NCOPY,NBACK,ICOPY, ..
       TCOPY,ECOPY,TAB,NOOP,TABSET,INSERT,  ..
       EOL,ESKIP,CPA,CA,CBACK,IBACK,WCOPY,  ..
       TBACK,EOL,ISKIP,WBACK,WSKIP,EBACK,   ..
       CSKIP,TYPESTATE,CRE,NOOP,NOOP,NOOP
   *
   UNLESS LCINIT DO FRETURN
   *
   INSMODE _ FALSE
   TCHAR _ 0
   CLEAR ( NEWLINE )
   RPRESET ( OLDLINE )
   *
   WHILE TCHAR = 0 DO
      CHAR _ CIN()
      IF CHAR<RUBOUT OR CHAR>177B THEN; * CHAR NE CONTROL-BYTE
         IF CHAR=201B THEN CHAR_'"'; * UMLAUT BECOMES DOUBLE QUOTE
         UNLESS TRIVIAL() DO COMPLAIN
      ELSE DO
         ( CTLKEY [ CHAR-RUBOUT ] ) ( )
      ENDIF
   ENDWHILE
   *
   POLISH ( NEWLINE , L :FRETURN )
   *
   EXCHANGE ( OLDLINE , NEWLINE )
   IF TCHAR = CRESC DO
      SRETURN TRUE
   ELSE DO
      SRETURN FALSE
   ENDIF
END
```

```
FUNCTION TRIVIAL ()
*
* TRIVIAL HANDLES INSERTION OF TYPED-IN TEXT CHARACTERS
*  INTO NEW LINE.   RETURNS TRUE IF  CHAR FITS
*  INTO NEWLINE, FALSE IF NEWLINE OVERFLOWS.
*
   WCI ( CHAR , NEWLINE :RETURN FALSE )
   UNLESS INSMODE DO GCI ( OLDLINE :0 )
   RETURN TRUE
END
```

```
FUNCTION POLISH ( REFERENCE NEWLINE , REFERENCE L )
*
* POLISH PRODUCES A FINAL SYSTEXT LINE 'L', WITH CARRIAGE RETURN
*  AND COMPRESSED BLANKS. FAILS IF NEWLINE IS TOO LONG TO FIT
*  IN 'L'.
*
   SCALAR NBLANKS, CHAR
   MACRO PUT ( C ) _ WCI ( C, L :FRETURN )
   *
   CLEAR ( L )
   *
   REPEAT
      CHAR _ GCI ( NEWLINE :GOTO NEWEND )
      IF CHAR = BLANK DO
         CHAR _ GCI ( NEWLINE :PUT(BLANK) & GOTO NEWEND )
         IF CHAR = BLANK DO
            PUT ( BLANKS )
            NBLANKS _ 1
            WHILE CHAR = BLANK DO
               NBLANKS _ NBLANKS + 1
               CHAR _ GCI ( NEWLINE :PUT(NBLANKS) & GOTO NEWEND )
            ENDWHILE
            PUT ( NBLANKS )
         ELSE DO
            PUT ( BLANK )
         ENDIF
      ENDIF
      PUT ( CHAR )
   ENDREPEAT
   *
   * END OF NEW LINE
   *
  NEWEND:
   PUT ( CARRET )
   *
   SRETURN
END
```

```
FUNCTION UNPOLISH ( REFERENCE L , REFERENCE OLDLINE )
*
* UNPOLISH ACCEPTS A SYSTEXT LINE, REMOVES THE
*  CARRIAGE RETURN, AND UNPACKS ANY COMPRESSED BLANKS.
*  UNPOLISH FAILS IF LINE IS TOO LONG OR DOES NOT
*  END WITH A CR.
*
   SCALAR NBLANKS, LTEMP, I, CHAR
   *
   RPSAVE ( L , LTEMP )
   CLEAR ( OLDLINE )
   *
   UNTIL ( CARRET = ( CHAR _ GCI(L :GOTO FAIL) ) ) DO
      IF CHAR = BLANKS DO
         NBLANKS _ GCI ( L :GOTO FAIL )
         FOR I _ 1 TO NBLANKS DO
            WCI ( BLANK , OLDLINE :GOTO FAIL )
         ENDFOR
      ELSE DO
         WCI ( CHAR , OLDLINE :GOTO FAIL )
      ENDIF
   ENDUNTIL
   RPRESTORE ( L , LTEMP )
   SRETURN
  FAIL:
   RPRESTORE ( L , LTEMP )
   FRETURN
END
```

```
FUNCTION NOOP()
*
* UNKNOWN CONTROL CHAR
*
   COMPLAIN
   RETURN
END
```

```
FUNCTION EOL()
*
* END-OF-LINE ( CARRIAGE RETURN WITH/WITHOUT ESCAPE )
*
   TCHAR _ CHAR
   CRLF()
   RETURN
END
```

```
FUNCTION CA()
*
* CONCATENATE-AND-ACCEPT
*
   CONCX( FALSE )
   COUT('"'); CRLF()
   RETURN
END
```

```
FUNCTION CPA()
*
* CONCATENATE-PRINT-AND-ACCEPT
*
   CONCX(TRUE)
   CRLF()
   RETURN
END
```

```
FUNCTION CONCX ( BOOL PRINT )
*
* CONCATENATE-AND-ACCEPT, WITH OPTIONAL PRINT
*
   NEWSAVE
   APPEND ( NEWLINE, OLDLINE :GOTO NEWEND )
   TCHAR _ CARRET
   IF PRINT DO
      SOUT ( OLDLINE )
   ENDIF
   RETURN
   *
   * END-OF-LINE
   *
  NEWEND:
   NEWRESTORE
   COMPLAIN
   RETURN
END
```

```
FUNCTION INSERT()
*
* TOGGLE INSERT MODE
*
   IF INSMODE DO
      INSMODE _ FALSE; COUT ( '>' )
   ELSE DO
      INSMODE _ TRUE; COUT ( '<' )
   ENDIF
   RETURN
END
```

```
FUNCTION TABSET()
*
* TOGGLE TAB SETTING AT CURRENT NEWLINE POSITION
*
   SCALAR L
   *
   L _ LENGTH ( NEWLINE )
   TABS [ L ] _ NOT TABS [ L ]
   RETURN
END
```

```
FUNCTION TAB()
*
* MOVE TO NEXT TABSTOP IN NEW LINE
*
   SCALAR D
   *
   OLDSAVE; NEWSAVE; CLEAR ( HOLD )
   *
   WCI ( BLANK , HOLD ); UNLESS INSMODE DO GCI ( OLDLINE :0 )
   D _ LENGTH ( NEWLINE ) + 1
   UNTIL TABS [ D ] DO
      D _ D + 1
      WCI ( BLANK , HOLD );
      UNLESS INSMODE DO GCI ( OLDLINE :0 )
      UNLESS ( D < LSZ ) DO GOTO NEWEND
   ENDUNTIL
   APPEND ( NEWLINE , HOLD :GOTO NEWEND )
   SOUT ( HOLD )
   RETURN
   *
   * END-OF-LINE
   *
  NEWEND:
   NEWRESTORE; OLDRESTORE
   COMPLAIN
   RETURN
END
```

```
FUNCTION TYPESTATE()
*
* TYPE STATE OF OLD AND NEW LINES
*
   SCALAR I,L
   *
   CLEAR ( HOLD )
   L _ LENGTH ( NEWLINE )
   FOR I _ 1 TO L DO
      WCI ( BLANK , HOLD )
   ENDFOR
   *
   CRLF(); SOUT ( HOLD ); SOUT ( OLDLINE )
   CRLF(); SOUT ( NEWLINE )
   *
   INSMODE _ FALSE
   RETURN
END
```

```
FUNCTION CRE()
*
* CONCATENATE-AND-RE-EDIT
*
    NEWSAVE
    APPEND ( NEWLINE , OLDLINE :GOTO NEWEND )
    EXCHANGE ( NEWLINE , OLDLINE )
    CLEAR ( NEWLINE )
    CRLF()
    INSMODE _ FALSE
    RETURN
    *
    * END-OF-LINE
    *
  NEWEND:
    NEWRESTORE
    COMPLAIN
    RETURN
END
```

```
FUNCTION CCOPY()
*
* COPY ONE CHARACTER
*
        CHAR _ GCI ( OLDLINE :GOTO OLDEND )
        WCI ( CHAR , NEWLINE :GOTO NEWEND )
        COUT ( CHAR )
        RETURN
       NEWEND:
        WCD ( CHAR , OLDLINE )
       OLDEND:
        COMPLAIN
        RETURN
END
```

```
FUNCTION WCOPY()
*
* COPY ONE WORD
*
        OLDSAVE; NEWSAVE; CLEAR ( HOLD )
        *
        UNTIL ALPHANUMERIC ( CHAR _ GCI ( OLDLINE :GOTO OLD1END )) DO
                WCI ( CHAR , HOLD )
        ENDUNTIL
        WCI ( CHAR , HOLD )
        WHILE ALPHANUMERIC ( CHAR _ GCI ( OLDLINE :GOTO OLD2END )) DO
                WCI ( CHAR , HOLD )
        ENDWHILE
        WCD ( CHAR , OLDLINE )
       OLD2END:
        APPEND ( NEWLINE , HOLD :GOTO NEWEND )
        SOUT ( HOLD )
        RETURN
        *
        * END-OF-LINE AT UNACCEPTABLE TIME
        *
       NEWEND:
        NEWRESTORE
       OLD1END:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION NCOPY()
*
* COPY TO NEXT CHARACTER TYPED
*
        XCOPY ( FALSE )
        RETURN
END
```

```
FUNCTION ICOPY()
*
* COPY THROUGH NEXT CHARACTER TYPED, INCLUSIVE
*
        XCOPY ( TRUE )
        RETURN
END
```

```
FUNCTION XCOPY ( BOOL INCLUSIVE )
*
* COPY UP TO THE NEXT CHARACTER TYPED, OPTIONALLY INCLUSIVE
*
        OLDSAVE; NEWSAVE; CLEAR ( HOLD )
        *
        SGETCHAR()
        *
        IF NOT INCLUSIVE DO
               WCI ( GCI ( OLDLINE :GOTO OLDEND ), HOLD )
        ENDIF
        *
        UNTIL ( SCHAR = ( CHAR _ GCI ( OLDLINE :GOTO OLDEND )) ) DO
              WCI ( CHAR , HOLD )
        ENDUNTIL
        *
        IF INCLUSIVE DO
                WCI ( CHAR , HOLD )
        ELSE DO
                WCD ( CHAR , OLDLINE )
        ENDIF
        APPEND ( NEWLINE , HOLD :GOTO NEWEND )
        SOUT ( HOLD )
        RETURN
        *
        * END-OF-LINE
        *
       NEWEND:
        NEWRESTORE
       OLDEND:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION TCOPY()
*
* COPY TO TAB
*
        SCALAR D
        *
        OLDSAVE; NEWSAVE; CLEAR ( HOLD )
        WCI ( GCI ( OLDLINE :GOTO OLDEND ) , HOLD )
        D _ DISTANCE ( OLDLINE )
        UNTIL TABS [ D ] DO
                WCI ( GCI ( OLDLINE :GOTO OLDEND ) , HOLD )
                D _ D + 1
        ENDUNTIL
        *
        APPEND ( NEWLINE , HOLD :GOTO NEWEND )
        SOUT ( HOLD )
        RETURN
        *
        * END-OF-LINE
        *
       NEWEND:
        NEWRESTORE
       OLDEND:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION ECOPY()
*
* COPY TO EDGE
*
        OLDSAVE; NEWSAVE; CLEAR ( HOLD )
        REPEAT
                CHAR _ GCI ( OLDLINE :GOTO OLDEND )
                WCI ( CHAR , HOLD )
        ENDREPEAT
       OLDEND:
        APPEND ( NEWLINE , HOLD :GOTO NEWEND )
        SOUT ( HOLD )
        RETURN
       NEWEND:
        NEWRESTORE; OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION CSKIP()
*
* SKIP ONE CHARACTER
*
        GCI ( OLDLINE :GOTO OLDEND )
        COUT ( BLOT )
        RETURN
       OLDEND:
        COMPLAIN
        RETURN
END
```

```
FUNCTION WSKIP()
*
* SKIP ONE WORD
*
        OLDSAVE; CLEAR ( HOLD )
        *
        UNTIL ALPHANUMERIC ( GCI ( OLDLINE :GOTO OLD1END )) DO
                WCI ( BLOT , HOLD )
        ENDUNTIL
        WCI ( BLOT , HOLD )
        WHILE ALPHANUMERIC ( CHAR _ GCI ( OLDLINE :GOTO OLD2END )) DO
                WCI ( BLOT , HOLD )
        ENDWHILE
        *
        WCD ( CHAR , OLDLINE )
       OLD2END:
        SOUT ( HOLD )
        RETURN
       OLD1END:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION NSKIP()
*
* SKIP TO NEXT CHARACTER TYPED
*
        XSKIP ( FALSE )
        RETURN
END
```

```
FUNCTION ISKIP()
*
* SKIP THROUGH NEXT CHARACTER TYPED, INCLUSIVE
*
        XSKIP ( TRUE )
        RETURN
END
```

```
FUNCTION XSKIP ( BOOL INCLUSIVE )
*
* SKIP UP TO NEXT CHARACTER TYPED, OPTIONALLY INCLUSIVE
*
        OLDSAVE; CLEAR ( HOLD )
        SGETCHAR()
        *
        IF NOT INCLUSIVE DO
             GCI ( OLDLINE :GOTO OLDEND )
             WCI ( BLOT, HOLD )
        ENDIF
        *
        UNTIL ( SCHAR = ( CHAR _ GCI ( OLDLINE :GOTO OLDEND )) ) DO
                WCI ( BLOT , HOLD )
        ENDUNTIL
        *
        IF INCLUSIVE DO
                WCI ( BLOT , HOLD )
        ELSE DO
                WCD ( CHAR , OLDLINE )
        ENDIF
        *
        SOUT ( HOLD )
        RETURN
        *
        * END-OF-LINE
        *
       OLDEND:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION TSKIP()
*
* SKIP TO TAB
*
        SCALAR D
        *
        OLDSAVE; CLEAR ( HOLD )
        GCI ( OLDLINE :GOTO  OLDEND )
        WCI ( BLOT , HOLD )
        D _ DISTANCE ( OLDLINE )
        UNTIL TABS [ D ] DO
                CHAR _ GCI ( OLDLINE :GOTO OLDEND )
                WCI ( BLOT , HOLD )
                D _ D + 1
        ENDUNTIL
        WCD ( CHAR , OLDLINE )
        SOUT ( HOLD )
        RETURN
        *
        * END-OF-LINE
        *
       OLDEND:
        OLDRESTORE
        COMPLAIN
        RETURN
END
```

```
FUNCTION ESKIP()
*
* SKIP TO EDGE
*
        CLEAR ( HOLD )
        REPEAT
                GCI ( OLDLINE :GOTO OLDEND )
                WCI ( BLOT , HOLD )
        ENDREPEAT
       OLDEND:
        SOUT ( HOLD )
        RETURN
END
```

```
FUNCTION CBACK()
*
* BACKUP ONE CHARACTER
*
        GCD ( NEWLINE :GOTO NEWEND )
        UNLESS INSMODE DO KCD ( OLDLINE )
        IF LENGTH ( NEWLINE ) = 0 DO
                EBACK()
        ELSE DO
                COUT ('{')
        ENDIF
        RETURN
       NEWEND:
        COMPLAIN
        RETURN
END
```

```
FUNCTION WBACK()
*
* BACK UP ONE WORD
*
   OLDSAVE; NEWSAVE
   *
   UNTIL ALPHANUMERIC ( GCD ( NEWLINE :GOTO NEW1END )) DO
      UNLESS INSMODE DO KCD ( OLDLINE )
   ENDUNTIL
   UNLESS INSMODE DO KCD ( OLDLINE )
   WHILE ALPHANUMERIC ( CHAR _ GCD ( NEWLINE :GOTO NEW2END )) DO
      UNLESS INSMODE DO KCD ( OLDLINE )
   ENDWHILE
   *
   WCI ( CHAR , NEWLINE )
  NEW2END:
   IF LENGTH ( NEWLINE ) = 0 DO
      EBACK()
      RETURN
   ENDIF
   COUT ('\')
   RETURN
   *
   * END-OF-LINE
   *
  NEW1END:
   NEWRESTORE; OLDRESTORE
   COMPLAIN
   RETURN
END
```

```
FUNCTION NBACK()
*
* BACK UP TO NEXT CHARACTER TYPED
*
   XBACK ( FALSE )
   RETURN
END
```

```
FUNCTION IBACK()
*
* BACK UP THROUGH NEXT CHARACTER TYPED, INCLUSIVE
*
   XBACK ( TRUE )
   RETURN
END
```

```
FUNCTION XBACK ( BOOL INCLUSIVE )
*
* BACK UP TO NEXT CHARACTER TYPED, OPTIONALL INCLUSIVE
*
   OLDSAVE; NEWSAVE
   SGETCHAR()
   *
   IF NOT INCLUSIVE DO
      GCD ( NEWLINE :GOTO NEWEND )
      UNLESS INSMODE DO KCD ( OLDLINE )
   ENDIF
   *
   UNTIL ( SCHAR = ( CHAR _ GCD ( NEWLINE :GOTO NEWEND )) ) DO
      UNLESS INSMODE DO KCD ( OLDLINE )
   ENDUNTIL
   *
   IF INCLUSIVE DO
      UNLESS INSMODE DO KCD ( OLDLINE )
      IF LENGTH ( NEWLINE ) = 0 DO
         EBACK()
         RETURN
      ENDIF
   ELSE DO
      WCI ( CHAR , NEWLINE )
   ENDIF
   COUT ( '\' )
   RETURN
   *
   * END-OF-LINE
   *
  NEWEND:
   NEWRESTORE; OLDRESTORE
   COMPLAIN
   RETURN
END
```

```
FUNCTION TBACK()
*
* BACK UP TO TAB
*
   SCALAR D
   *
   OLDSAVE; NEWSAVE
   UNLESS INSMODE DO KCD ( OLDLINE )
   GCD ( NEWLINE : NEWEND )
   D _ LENGTH ( NEWLINE )
   UNTIL TABS [ D ] DO
      UNLESS INSMODE DO KCD ( OLDLINE )
   GCD ( NEWLINE :GOTO NEWEND )
   D _ D - 1
   ENDUNTIL
   COUT ( '\' )
   RETURN
  NEWEND:
   NEWRESTORE; OLDRESTORE
   COMPLAIN
   RETURN
END
```

```
FUNCTION EBACK()
*
* BACK UP TO EDGE
*
   RPRESET ( OLDLINE ); CLEAR ( NEWLINE )
   INSMODE _ FALSE
   COUT ( '&135' ); CRLF()
   RETURN
END
```

```
FUNCTION KCD ( REFERENCE S )
*
* KEEP CHARACTER AND DECREMENT RDR PTR
*
   SCALAR RP
   *
   IF ( RP _ S.SDFPTR ) > 0 DO
      S.SDFPTR _ RP - 1
   ENDIF
   RETURN
END
```

```
FUNCTION SGETCHAR()
*
* GET SEARCH CHARACTER, WITHOUT ECHO
*
   SET\ECHO ( , FALSE )
   SCHAR _ CIN()
   SET\ECHO ( , TRUE )
   RETURN
END
```