UCSD PASCAL P-SYSTEM I.5 SOURCE CODE LISTINGS

September 1978

Table of Contents

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Binder
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Binder

    4    4    0  BINDER              [Main] UCSD Pascal 1.5 Binder
   35   35    1     ERROR
   41   41    1     GETFILE
   89   89    1     MOVECODE
  126  126    1     LINKER
  140  140    2        LINKCODE
  143  143    3           LINK0
  161  161    3           LINKIT

*** End ProcNames: 8 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Binder
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Compiler
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Compiler

   11   11    0  PASCALSYSTEM          [+] UCSD Pascal 1.5 Compiler
   48   48    1     USERPROGRAM
   50   50    2        FILEHANDLER
   53   53    2        DEBUGGER
   56   56    2        PRINTERROR
   59   59    2        INITIALIZE
   62   62    2        GETCMD
   65   65    2        NOTUSED1
   68   68    2        NOTUSED2
   71   71    2        NOTUSED3
   76   76    1     PASCALCOMPILER
  414  414    2        COMPINIT
  416  416    3           ENTSTDTYPES
  451  451    3           ENTSTDNAMES
  527  527    3           ENTUNDECL
  563  563    3           ENTSPCPROCS
  601  601    3           ENTSTDPROCS
  651  651    3           INITSCALARS
  679  679    3           INITSETS
  772  772    2        DECLARATIONPART
  777  777    3           TYP
  782  782    4              SIMPLETYPE
  910  910    4              PACKABLE
  940  940    4              FIELDLIST
  945  945    5                 ALLOCATE
  973  973    5                 VARIANTLIST
 1112 1112    4              POINTERTYPE
 1292 1292    3           USESDECLARATION
 1309 1309    4              GETTEXT
 1453 1453    3           LABELDECLARATION
 1483 1483    3           CONSTDECLARATION
 1509 1509    3           TYPEDECLARATION
 1547 1547    3           VARDECLARATION
 1605 1605    3           PROCDECLARATION
 1611 1611    4              PARAMETERLIST
 1998 1998    2        BODYPART
 2000 2000    3           LINKERREF
 2017 2017    3           GENLDC
 2027 2027    3           GENBIG
 2038 2038    3           GEN0
 2049 2049    3           GEN1
 2095 2095    3           GEN2
 2118 2118    3           GENNR
 2120 2120    4              ASSIGN
 2138 2138    3           GENJMP
 2169 2169    3           GENFJP
 2175 2175    3           GENLABEL
 2181 2181    3           PUTLABEL
 2200 2200    3           LOAD
 2247 2247    3           STORE
 2262 2262    3           LOADADDRESS
 2283 2283    3           SELECTOR
 2481 2481    3           CALL
```

```
2484 2484    4              VARIABLE
2493 2493    4              STRGVAR
2525 2525    4              ROUTINE
2527 2527    5                 NEWSTMT
2574 2574    5                 MOVE
2589 2589    5                 EXIT
2611 2611    5                 UNITIO
2639 2639    5                 CONCAT
2667 2667    5                 COPYDELETE
2705 2705    5                 STR
2723 2723    5                 CLOSE
2749 2749    5                 GETPUTETC
2778 2778    5                 SCAN
2806 2806    5                 BLOCKIO
2831 2831    5                 SIZEOF
2865 2865    4              LOADIDADDR
2876 2876    4              READ
2929 2929    4              WRITE
3021 3021    4              CALLNONSPECIAL
3299 3299    3           EXPRESSION
3305 3305    4              FLOATIT
3316 3316    4              STRETCHIT
3327 3327    4              SIMPLEEXPRESSION
3330 3330    5                 TERM
3333 3333    6                    FACTOR
3614 3614    4              MAKEPA
3739 3739    3           STATEMENT
3743 3743    4              ASSIGNMENT
3826 3826    4              GOTOSTATEMENT
3848 3848    4              COMPOUNDSTATEMENT
3859 3859    4              IFSTATEMENT
3892 3892    4              CASESTATEMENT
3971 3971    4              REPEATSTATEMENT
3986 3986    4              WHILESTATEMENT
3994 3994    4              FORSTATEMENT
4072 4072    4              WITHSTATEMENT
4157 4157    3           BODY
4325 4325    2        WRITELINKERINFO
4354 4354    3           GETREFS
4357 4357    4              GETNEXTBLOCK
4380 4380    3           GLOBALSEARCH
4562 4562    2        UNITPART
4565 4565    3           OPENREFFILE
4571 4571    3           UNITDECLARATION
4693 4693    2        ERROR
4735 4735    2        GETNEXTPAGE
4777 4777    2        PRINTLINE
4804 4804    2        ENTERID
4820 4820    2        INSYMBOL
4824 4824    3           CHECKEND
4850 4850    3           COMMENTER
4853 4853    4              SCANSTRING
4946 4946    3           STRING
4983 4983    3           NUMBER
5206 5206    2        SEARCHSECTION
5214 5214    2        SEARCHID
5243 5243    2        GETBOUNDS
5258 5258    2        SKIP
5262 5262    2        PAOFCHAR
5269 5269    2        STRGTYPE
```

```
 5274 5274    2        DECSIZE
 5277 5277    2        CONSTANT
 5370 5370    2        COMPTYPES
 5447 5447    2        GENBYTE
 5452 5452    2        GENWORD
 5459 5459    2        WRITETEXT
 5468 5468    2        WRITECODE
 5488 5488    2        FINISHSEG
 5503 5503    2        BLOCK
 5507 5507    3           FINDFORW

*** End ProcNames: 123 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Compiler
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Disassembler
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Disassembler

    2    2    0  CODESTAT            [+] UCSD Pascal 1.5 Disassembler
   96   96    1     INIT
  101  101    2        NEWOP
  215  215    1     DISASSEMBLE
  217  217    2        BUFRESET
  229  229    2        LASTBYTE
  245  245    2        GETBYTE
  261  261    2        GETBIG
  275  275    2        GETWORD
  291  291    2        MOSTSIGBIT
  309  309    2        SHORTOP
  351  351    2        ONEOP
  357  357    3           JUMPOPST
  414  414    2        OPTOP
  438  438    2        LOPTOP
  459  459    2        TWOOP
  491  491    2        WORDOP
  506  506    2        WORDSOP
  541  541    2        CMPRSSOP
  555  555    2        CMPRSS2OP
  578  578    2        CHRSOP
  599  599    2        BLKOP
  626  626    2        PROCEJUR
  630  630    3           JUMPINFO
  719  719    2        ALLPROCS
  751  751    2        SEGMINT
  774  774    2        ACTACCESS
  800  800    2        PROCGUIDE
  806  806    3           DATASEGINFO
  828  828    3           PROCLOOK
  914  914    2        SEGMTGUIDE
  954  954    2        LEXGUIDE
 1024 1024    1     GATHER
 1027 1027    2        WRITEHDR
 1045 1045    2        JUMPSTUFF
 1062 1062    2        PROCSTUFF
 1074 1074    2        SHORTSTUFF
 1077 1077    3           SHORT1
 1107 1107    3           SHORT2
 1143 1143    2        SHORTST
 1153 1153    2        ONEST
 1174 1174    2        TWOST
 1208 1208    2        WORDST
 1228 1228    2        LOPTST
 1251 1251    2        WORDSST
 1273 1273    2        CMPRSSST
 1301 1301    2        CMPRSS2ST
 1322 1322    2        GINIT
 1376 1376    1     DATACOUNT
 1387 1387    2        SETORDER
 1390 1390    3           DATASET
 1438 1438    2        DATAHEADER
 1449 1449    2        PRINTDATA
```

```
 1487 1487    1     PROMPT

*** End ProcNames: 54 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Disassembler
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 LibMap
########################################################################

Procedure/Function names for UCSD Pascal 1.5 LibMap

   30   30    0  LIBMAP                 [+] UCSD Pascal 1.5 LibMap
  171  171    1     alphabetic
  185  185    1     phase2
  194  194    2        readlinkinfo
  202  202    3           copyinterface
  257  257    3           getentry
  276  276    3           ref
  386  386    1     getfile

*** End ProcNames: 8 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 LibMap
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Librarian
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Librarian

    4    4    0  PLIBRARIAN            [Main] UCSD Pascal 1.5 Librarian
   65   65    1     LIBRARIAN
   94   94    2        NEWLINKER
  102  102    3           PROMPT
  110  110    3           CHECKIO
  123  123    3           OPENFILE
  137  137    3           DISPLAY
  156  156    3           LINKCODE
  159  159    4              LINKIT
  161  161    5                 COPYLINKINFO
  167  167    6                    GETREC
  215  215    5                 COPYINTERFACE
  300  300    4              CONFIRM

*** End ProcNames: 13 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Librarian
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Linker
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Linker

   31   31    0  systemlevel           [+] UCSD Pascal 1.5 Linker
   69   69    1     linker
  259  259    2        error
  280  280    2        fetchbyte
  285  285    2        fetchword
  293  293    2        storebyte
  298  298    2        storeword
  314  314    2        entersym
  357  357    2        symsrch
  383  383    2        unitsrch
  410  410    2        alphabetic
  428  428    2        getcodep
  459  459    2        phase1
  469  469    3           buildfilelist
  481  481    4              setupfile
  494  494    5                 getfilep
  648  648    3           buildseginfo
  719  719    3           buildseplist
  812  812    2        phase2
  825  825    3           readlinkinfo
  839  839    4              getentry
  865  865    4              addunit
  898  898    4              validate
 1022 1022    3           buildplaces
 1033 1033    4              procsrch
 1166 1166    2        phase3
 1223 1223    3           buildworklists
 1235 1235    4              findprocs
 1244 1244    5                 procsrch
 1285 1285    4              findnewprocs
 1299 1299    5                 findnadd
 1307 1307    6                    procsrch
 1379 1379    4              resolve
 1392 1392    5                 sepsrch
 1422 1422    5                 procinsert
 1589 1589    4              refsrch
 1601 1601    5                 checkrefs
 1693 1693    4              findlocals
 1790 1790    3           readsrcseg
 1807 1807    4              readnsplit
 1920 1920    3           copyinprocs
 1933 1933    4              readsepseg
 1999 1999    3           fixuprefs
 2104 2104    3           writetocode
 2140 2140    3           linksegment
 2147 2147    4              writemap

*** End ProcNames: 46 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Linker
```

```
##########################################################################
### FILE: P.UCSD Pascal 1.5 PIO
##########################################################################

Procedure/Function names for UCSD Pascal 1.5 PIO

   17   17    0  PASCALSYSTEM          [+] UCSD Pascal 1.5 PIO

*** End ProcNames: 1 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 PIO
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 PIO Unit
########################################################################

Procedure/Function names for UCSD Pascal 1.5 PIO Unit

    4    4    0  PASCALIO          [Main] UCSD Pascal 1.5 PIO Unit
   17   17    1     FSEEK
   75   75    1     FREADREAL
  122  122    1     FWRITEREAL
  205  205    1     FWRITEDEC
  212  212    1     FREADDEC

*** End ProcNames: 6 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 PIO Unit
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 Radix
########################################################################

Procedure/Function names for UCSD Pascal 1.5 Radix

    4    4    0  CONVERSION       [Main] UCSD Pascal 1.5 Radix
   55   55    1     PROMPT
   62   62    1     CLEARSCREEN
   68   68    1     INIT
   93   93    1     INITSCREEN
  114  114    1     DECTO
  159  159    1     HEXTO
  194  194    1     OCTTO
  233  233    1     BINTO
  265  265    1     WRITEOUT
  303  303    1     OUTER
  394  394    1     HEADER

*** End ProcNames: 12 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 Radix
```

```
########################################################################
### FILE: P.UCSD Pascal 1.5 System
########################################################################

Procedure/Function names for UCSD Pascal 1.5 System

   14   14    0  PASCALSYSTEM        [Main] UCSD Pascal 1.5 System
  345  345    1     USERPROGRAM
  351  351    1     DEBUGGER
  357  357    1     PRINTERROR
  403  403    1     INITIALIZE
  409  409    2        INITSYSCOM
  443  443    2        INITUNITABLE
  471  471    2        INITFNAMES
  523  523    2        INITCHARSET
  577  577    2        INITHEAP
  597  597    2        INITWORKFILE
  635  635    2        INITFILES
  694  694    1     GETCMD
  701  701    2        ASSOCIATE
  780  780    2        STARTCOMPILE
  834  834    2        FINISHCOMPILE
  860  860    2        EXECUTE
  878  878    2        RUNWORKFILE
  981  981    1     EXECERROR
 1012 1012    1     CHECKDEL
 1052 1052    1     PUTPREFIXED
 1065 1065    1     HOMECURSOR
 1070 1070    1     CLEARSCREEN
 1082 1082    1     CLEARLINE
 1087 1087    1     PROMPT
 1102 1102    1     FGOTOXY
 1114 1114    1     GETCHAR
 1126 1126    1     SPACEWAIT
 1141 1141    1     SCANTITLE
 1223 1223    1     FETCHDIR
 1277 1277    1     WRITEDIR
 1314 1314    1     VOLSEARCH
 1376 1376    1     DIRSEARCH
 1389 1389    1     DELENTRY
 1401 1401    1     INSENTRY
 1413 1413    1     ENTERTEMP
 1418 1418    2        FINDMAX
 1476 1476    1     FINIT
 1494 1494    1     RESETER
 1533 1533    1     FOPEN
 1647 1647    1     FCLOSE
 1724 1724    1     XSEEK
 1730 1730    1     XREADREAL
 1736 1736    1     XWRITEREAL
 1742 1742    1     CANTSTRETCH
 1783 1783    1     FRESET
 1794 1794    1     FBLOCKIO
 1846 1846    1     FGET
 1929 1929    1     FPUT
 1997 1997    1     FEOF
 2002 2002    1     FEOLN
 2005 2005    1     FWRITELN
 2010 2010    1     FWRITECHAR
```

```
 2037 2037    1     FWRITEINT
 2063 2063    1     FWRITESTRING
 2083 2083    1     FREADSTRING
 2105 2105    1     FWRITEBYTES
 2124 2124    1     FREADLN
 2132 2132    1     FREADCHAR
 2143 2143    1     FREADINT
 2177 2177    1     SCONCAT
 2186 2186    1     SINSERT
 2206 2206    1     SCOPY
 2216 2216    1     SDELETE
 2232 2232    1     SPOS
 2261 2261    1     COMMAND

*** End ProcNames: 66 Procedures and Functions

### END OF FILE P.UCSD Pascal 1.5 System
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Binder
########################################################################


 (* UCSD PASCAL I.5 P-SYSTEM "BINDER" *)

 PROGRAM BINDER;
 CONST
   MAXSEG = 15;

 TYPE
   SEGNUM = 0..MAXSEG;
   SEGTBLP = ^SEGTBL;
   SEGTBL = RECORD
              SEGDESC: ARRAY[SEGNUM] OF
                         RECORD
                           DISKADDR: INTEGER;
                           CODELENG: INTEGER
                         END {SEGDESC};
              SEGNAME: ARRAY[SEGNUM] OF
                         PACKED ARRAY[0..7] OF CHAR;
              STUFF: PACKED ARRAY[0..319] OF CHAR
            END {SEGTBL};

    BUFFER = PACKED RECORD CASE INTEGER OF
          1: ( BYTES: PACKED ARRAY[0..10239] OF 0..255);
          2: ( WORDS: ARRAY[0..5119] OF INTEGER)
    END {BUFFERS};
 VAR
   CCH: CHAR;
   ZEROBYTES,USERBYTES: INTEGER;
   TABLE: SEGTBLP;
   ZEROBUF,USERBUF: ^BUFFER;
   HEAPPTR: ^INTEGER;
   SOURCE,INFILE: FILE;


 PROCEDURE ERROR;
 BEGIN
   WRITELN(' ERROR ');
   EXIT(PROGRAM);
 END;

 PROCEDURE GETFILE;
 {$I-}
 CONST
   USERSEG = 0;
 VAR
   BLOCKS: INTEGER;
   TITLE: STRING;
 BEGIN
   WRITELN;
   WRITELN('This program modifies the SYSTEM.PASCAL of your default prefix');
   WRITELN('disk.  If any of the files it expects to be around are missing,');
   WRITELN('i.e. SYSTEM.PASCAL, or enough room (60 blocks) to re-create it,');
   WRITELN('it will terminate with the cryptic message "ERROR"');
   writeln;
   writeln('You also need to execute the program SETUP to get the system to');
```

```
  writeln('work intelligently with your terminal.');
  WRITELN;
  REPEAT
    WRITE(' File with GOTOXY(X,Y: INTEGER) procedure:');
    READLN(TITLE);
    IF LENGTH(TITLE) = 0 THEN EXIT(PROGRAM);
    OPENOLD(INFILE,TITLE);
    IF IORESULT <> 0 THEN
      OPENOLD(INFILE,CONCAT(TITLE,'.CODE'));
  UNTIL IORESULT = 0;
  OPENOLD(SOURCE,'SYSTEM.PASCAL');
  IF IORESULT <> 0 THEN ERROR;
  {$I+}
  {read in SYSTEM.PASCALs segtable}
  {read in SYSTEM.PASCALs segment 0}
  {read in named files segtable}
  {read in named files segment 1}
  IF BLOCKREAD(SOURCE,TABLE^,1,0) <> 1 THEN ERROR;
  WITH TABLE^.SEGDESC[0] DO
    BEGIN
      ZEROBYTES := CODELENG;
      BLOCKS := (CODELENG + 511) DIV 512;
      IF BLOCKREAD(SOURCE,ZEROBUF^,BLOCKS,DISKADDR) <> BLOCKS THEN ERROR;
    END;
  IF BLOCKREAD(INFILE,TABLE^,1,0) <> 1 THEN ERROR;
  WITH TABLE^.SEGDESC[USERSEG] DO
    BEGIN
      USERBYTES := CODELENG;
      BLOCKS := (CODELENG + 511) DIV 512;
      IF BLOCKREAD(INFILE,USERBUF^,BLOCKS,DISKADDR) <> BLOCKS THEN ERROR;
    END;
END {GETFILE};

PROCEDURE MOVECODE;
  CONST
    INPNUM = 2;
    OUTPNUM = 29;
  {Move procedure #2 from buffer USERBUF^ to
      procedure #29 buffer ZEROBUF^, and
      point LINKER at it.}
VAR
  CODESIZE,CODEAT,ENTERIC,CODEBASE: INTEGER;
  INPPOINT: INTEGER;
BEGIN
  {set inppoint to location of proc offset in source}
  INPPOINT := (INPNUM*2+2);
  {set codebase to where proc 'starts' in source}
  CODEBASE := USERBUF^.WORDS[(USERBYTES DIV 2) -(INPNUM+1)];
  {get enteric from source}
  ENTERIC := USERBUF^.WORDS[(USERBYTES-CODEBASE-INPPOINT) DIV 2 -1];
  {set procedure to appropriate number}
  USERBUF^.BYTES[USERBYTES-CODEBASE-INPPOINT] := OUTPNUM;
  {set lex level to zero}
  USERBUF^.BYTES[USERBYTES-CODEBASE-(INPPOINT-1)] := 0;
  {number of bytes of code is enteric + 4 more bytes}
  CODESIZE := ENTERIC + 4;
  {code is located at ... }
  CODEAT := USERBYTES - CODEBASE - CODESIZE - INPNUM*2;
  {make room for the code coming in}
  MOVERIGHT(ZEROBUF^.BYTES[0],ZEROBUF^.BYTES[CODESIZE],ZEROBYTES);
```

```
  {put the frigging code in}
  MOVELEFT(USERBUF^.BYTES[CODEAT],ZEROBUF^.BYTES[0],CODESIZE);
  {make a note of the fact that you have stretched the segment}
  ZEROBYTES := ZEROBYTES + CODESIZE;
  {point the appropriate word at the appropriate byte}
  ZEROBUF^.WORDS[(ZEROBYTES DIV 2)-(OUTPNUM+1)] :=
                                              ZEROBYTES - CODESIZE-(OUTPNUM*2);
END;


PROCEDURE LINKER(NTITLE,TITLE: STRING);

CONST
  WINDOW = 2;
  MARKCODE = 15;
  MARKIN = 5;


VAR LENCODE,NBLOCKS,RSLT,OUTBLOCK: INTEGER;
    INTBL,BUF: SEGTBLP;
    SEG: SEGNUM;
    CODETABLE: SEGTBLP;
    CODE: FILE;

PROCEDURE LINKCODE;
  VAR NBLOCKS: INTEGER;

  PROCEDURE LINK0;
    BEGIN
      WITH INTBL^,SEGDESC[0] DO
        BEGIN
            NBLOCKS := (ZEROBYTES + 511) DIV 512;
            IF BLOCKWRITE(CODE,ZEROBUF^,NBLOCKS,OUTBLOCK) <> NBLOCKS THEN
              ERROR
            ELSE
              BEGIN
                CODETABLE^.SEGNAME[0] := 'PASCALSY';
                CODETABLE^.SEGDESC[0].CODELENG := ZEROBYTES;
                CODETABLE^.SEGDESC[0].DISKADDR := OUTBLOCK;
                LENCODE := LENCODE + NBLOCKS;
                OUTBLOCK := OUTBLOCK + NBLOCKS
              END
        END;
    END;

  PROCEDURE LINKIT;
    BEGIN
      WITH INTBL^,SEGDESC[SEG] DO
        BEGIN
          NBLOCKS := (CODELENG+511) DIV 512;
          IF BLOCKREAD(INFILE,BUF^,NBLOCKS,DISKADDR) <> NBLOCKS THEN
            ERROR
          ELSE
            IF BLOCKWRITE(CODE,BUF^,NBLOCKS,OUTBLOCK) <> NBLOCKS THEN
              ERROR
            ELSE
              BEGIN
                CODETABLE^.SEGNAME[SEG] := SEGNAME[SEG];
                CODETABLE^.SEGDESC[SEG].CODELENG := CODELENG;
                CODETABLE^.SEGDESC[SEG].DISKADDR := OUTBLOCK;
```

```
                LENCODE := LENCODE + NBLOCKS;
                OUTBLOCK := OUTBLOCK + NBLOCKS
              END
          END;
      END;

BEGIN
  IF LENGTH(NTITLE)>0 THEN
    IF BLOCKREAD(INFILE,INTBL^,1,0) = 1 THEN
    ELSE
      ERROR;
    LINK0;
    FOR SEG := 1 TO 15 DO
      IF (INTBL^.SEGDESC[SEG].CODELENG > 0) THEN LINKIT;
  CLOSE(INFILE)
END {LINKCODE} ;

BEGIN
  LENCODE := 0;
  NEW(CODETABLE);
  NEW(INTBL);
  OPENNEW(CODE,TITLE);
  OUTBLOCK := 1; NEW(BUF);
  WITH CODETABLE^ DO
    FOR SEG := 0 TO MAXSEG DO
      BEGIN  SEGNAME[SEG] := '        ';
        SEGDESC[SEG].CODELENG := 0;
        SEGDESC[SEG].DISKADDR := 0
      END;
  OPENOLD(INFILE,NTITLE);
  LINKCODE;
  IF BLOCKWRITE(CODE,CODETABLE^,1,0) = 1 THEN CLOSE(CODE,LOCK)
  ELSE
    WRITELN(OUTPUT,'Code file write error ')
END;

BEGIN
  NEW(ZEROBUF);
  MARK(HEAPPTR);
  NEW(TABLE);
  NEW(USERBUF);
  GETFILE;
  WRITELN;
  WRITELN(' Moving procedures around ');
  MOVECODE;
  RELEASE(HEAPPTR);
  USERBUF := NIL;
  TABLE := NIL;
  CLOSE(INFILE);
  WRITELN;
  WRITELN(' Calling system linker to create new SYSTEM.PASCAL');
  LINKER('SYSTEM.PASCAL','SYSTEM.PASCAL[60]');
END {BINDER}.
```

```
{ +-------------------------------------------------------------+
  |                                                             |
  |                    F    I    N    I    S                   |
  |                                                             |
  +-------------------------------------------------------------+ }
```

END OF FILE UCSD Pascal 1.5 Binder

```
########################################################################
### FILE: UCSD Pascal 1.5 Compiler
########################################################################


 (* SWAPPING PASCAL COMPILER INCLUDE FILES *)
 (*$C COPYRIGHT (C) 1978 REGENTS UCSD I.5.A.1*)

 (*$T+*) (*$S+*)

 (* $I COMPGLBLS.TEXT*)

 (*$U-*)

 PROGRAM PASCALSYSTEM;  (* VERSION I.5 (Unit Compiler)  9-01-78 *)


 (***********************************************)
 (*                                             *)
 (*        UCSD  PASCAL  COMPILER               *)
 (*                                             *)
 (*    BASED ON ZURICH P2 PORTABLE              *)
 (*    COMPILER, EXTENSIVLY                     *)
 (*    MODIFIED BY ROGER T. SUMNER              *)
 (*    SHAWN FANNING AND ALBERT A. HOFFMAN      *)
 (*    1976..1978                               *)
 (*                                             *)
 (*    RELEASE LEVEL: I.3 AUGUST, 1977          *)
 (*                   I.4 JANUARY, 1978         *)
 (*                   I.5 SEPTEMBER, 1978       *)
 (*                                             *)
 (*    INSTITUTE FOR INFORMATION SYSTEMS        *)
 (*    UC SAN DIEGO, LA JOLLA, CA 92093         *)
 (*                                             *)
 (*    KENNETH L. BOWLES, DIRECTOR              *)
 (*                                             *)
 (*    COPYRIGHT (C) 1978, REGENTS OF THE       *)
 (*    UNIVERSITY OF CALIFORNIA, SAN DIEGO      *)
 (*                                             *)
 (***********************************************)


 TYPE PHYLE = FILE;
 INFOREC = RECORD
             WORKSYM,WORKCODE: ^PHYLE;
             ERRSYM,ERRBLK,ERRNUM: INTEGER;
             SLOWTERM,STUPID: BOOLEAN;
             ALTMODE: CHAR
           END;


 SEGMENT PROCEDURE USERPROGRAM;

   SEGMENT PROCEDURE FILEHANDLER;
   BEGIN END;

   SEGMENT PROCEDURE DEBUGGER;
   BEGIN END;
```

```
  SEGMENT PROCEDURE PRINTERROR;
  BEGIN END;

  SEGMENT PROCEDURE INITIALIZE;
  BEGIN END;

  SEGMENT PROCEDURE GETCMD;
  BEGIN END;

  SEGMENT PROCEDURE NOTUSED1;
  BEGIN END;

  SEGMENT PROCEDURE NOTUSED2;
  BEGIN END;

  SEGMENT PROCEDURE NOTUSED3;
  BEGIN END;

BEGIN END; (* USERPROGRAM *)

SEGMENT PROCEDURE PASCALCOMPILER(VAR USERINFO: INFOREC);

CONST DISPLIMIT = 12; MAXLEVEL = 8; MAXADDR = 28000;
      INTSIZE = 1; REALSIZE = 2; BITSPERWD = 16;
      CHARSIZE = 1; BOOLSIZE = 1; PTRSIZE = 1;
      FILESIZE = 300; NILFILESIZE = 40; BITSPERCHR = 8; CHRSPERWD = 2;
      STRINGSIZE = 0; STRGLGTH = 255; MAXINT = 32767; MAXDEC = 36;
      DEFSTRGLGTH = 80; LCAFTERMARKSTACK = 1; REFSPERBLK = 128;
      EOL = 13; MAXCURSOR = 1023; MAXCODE = 1299;
      MAXJTAB = 24; MAXSEG = 15; MAXPROCNUM = 149;

TYPE
                  (*BASIC SYMBOLS, MUST MATCH ORDER IN IDSEARCH*)

     SYMBOL =  (IDENT,COMMA,COLON,SEMICOLON,LPARENT,RPARENT,DOSY,TOSY,
                DOWNTOSY,ENDSY,UNTILSY,OFSY,THENSY,ELSESY,BECOMES,LBRACK,
                RBRACK,ARROW,PERIOD,BEGINSY,IFSY,CASESY,REPEATSY,WHILESY,
                FORSY,WITHSY,GOTOSY,LABELSY,CONSTSY,TYPESY,VARSY,PROCSY,
                FUNCSY,PROGSY,FORWARDSY,INTCONST,REALCONST,STRINGCONST,
                NOTSY,MULOP,ADDOP,RELOP,SETSY,PACKEDSY,ARRAYSY,RECORDSY,
                FILESY,OTHERSY,LONGCONST,USESSY,UNITSY,INTERSY,IMPLESY,
                EXTERNLSY,SEPARATSY);


     OPERATOR = (MUL,RDIV,ANDOP,IDIV,IMOD,PLUS,MINUS,OROP,LTOP,LEOP,
                 GEOP,GTOP,NEOP,EQOP,INOP,NOOP);

     SETOFSYS = SET OF SYMBOL;

     NONRESIDENT = (SEEK,FREADREAL,FWRITEREAL,FREADDEC,FWRITEDEC,DECOPS);
     NONRESPFLIST = ARRAY[NONRESIDENT] OF INTEGER;

                                              (*CONSTANTS*)
     CSTCLASS = (REEL,PSET,STRG,TRIX,LONG);
     CSP = ^ CONSTREC;
     CONSTREC = RECORD CASE CCLASS: CSTCLASS OF
                         LONG: (LLENG,LLAST: INTEGER;
                                LONGVAL: ARRAY[1..9] OF INTEGER);
                         TRIX: (CSTVAL: ARRAY [1..8] OF INTEGER);
                               (*MUST COMPLETELY OVERLAP FOLLOWING FIELDS*)
```

```
                    REEL: (RVAL: REAL);
                    PSET: (PVAL: SET OF 0..127);
                    STRG: (SLGTH: 0..STRGLGTH;
                           SVAL: PACKED ARRAY [1..STRGLGTH] OF CHAR)
                  END;

VALU = RECORD CASE BOOLEAN OF
                TRUE:  (IVAL: INTEGER);
                FALSE: (VALP: CSP)
              END;

                                                  (*DATA STRUCTURES*)
BITRANGE = 0..BITSPERWD; OPRANGE = 0..80;
CURSRANGE = 0..MAXCURSOR; PROCRANGE = 0..MAXPROCNUM;
LEVRANGE = 0..MAXLEVEL; ADDRRANGE = 0..MAXADDR;
JTABRANGE = 0..MAXJTAB; SEGRANGE = 0..MAXSEG;
DISPRANGE = 0..DISPLIMIT;

STRUCTFORM = (SCALAR,SUBRANGE,POINTER,LONGINT,POWER,ARRAYS,
              RECORDS,FILES,TAGFLD,VARIANT);

DECLKIND = (STANDARD,DECLARED,SPECIAL);

STP = ^ STRUCTURE; CTP = ^ IDENTIFIER;

STRUCTURE = RECORD
              SIZE: ADDRRANGE;
              CASE FORM: STRUCTFORM OF
                SCALAR:   (CASE SCALKIND: DECLKIND OF
                             DECLARED: (FCONST: CTP));
                SUBRANGE: (RANGETYPE: STP; MIN,MAX: VALU);
                POINTER:  (ELTYPE: STP);
                POWER:    (ELSET: STP);
                ARRAYS:   (AELTYPE,INXTYPE: STP;
                           CASE AISPACKD:BOOLEAN OF
                             TRUE: (ELSPERWD,ELWIDTH: BITRANGE;
                                    CASE AISSTRNG: BOOLEAN OF
                                     TRUE:(MAXLENG: 1..STRGLGTH)));
                RECORDS:  (FSTFLD: CTP; RECVAR: STP);
                FILES:    (FILTYPE: STP);
                TAGFLD:   (TAGFIELDP: CTP; FSTVAR: STP);
                VARIANT:  (NXTVAR,SUBVAR: STP; VARVAL: VALU)
              END;

                                                        (*NAMES*)
IDCLASS = (TYPES,KONST,FORMALVARS,ACTUALVARS,FIELD,
           PROC,FUNC,MODULE);
SETOFIDS = SET OF IDCLASS;
IDKIND = (ACTUAL,FORMAL);
ALPHA = PACKED ARRAY [1..8] OF CHAR;

IDENTIFIER = RECORD
               NAME: ALPHA; LLINK, RLINK: CTP;
               IDTYPE: STP; NEXT: CTP;
               CASE KLASS: IDCLASS OF
                 KONST: (VALUES: VALU);
             FORMALVARS,
             ACTUALVARS: (VLEV: LEVRANGE;
                          VADDR: ADDRRANGE;
                          CASE BOOLEAN OF
```

```
                        TRUE: (PUBLIC: BOOLEAN));
                FIELD: (FLDADDR: ADDRRANGE;
                        CASE FISPACKD: BOOLEAN OF
                          TRUE: (FLDRBIT,FLDWIDTH: BITRANGE));
                PROC,
                FUNC:  (CASE PFDECKIND: DECLKIND OF
                          SPECIAL:  (KEY: INTEGER);
                          STANDARD: (CSPNUM: INTEGER);
                          DECLARED: (PFLEV: LEVRANGE;
                                     PFNAME: PROCRANGE;
                                     PFSEG: SEGRANGE;
                                     CASE PFKIND: IDKIND OF
                                      ACTUAL: (LOCALLC: ADDRRANGE;
                                               FORWDECL: BOOLEAN;
                                               EXTURNAL: BOOLEAN;
                                               INSCOPE: BOOLEAN;
                                               CASE BOOLEAN OF
                                                TRUE: (IMPORTED:BOOLEAN))));
              MODULE: (SEGID: INTEGER)
              END;


WHERE = (BLCK,CREC,VREC,REC);

                                          (*EXPRESSIONS*)
ATTRKIND = (CST,VARBL,EXPR);
VACCESS = (DRCT,INDRCT,PACKD,MULTI,BYTE);

ATTR = RECORD TYPTR: STP;
         CASE KIND: ATTRKIND OF
           CST:   (CVAL: VALU);
           VARBL: (CASE ACCESS: VACCESS OF
                     DRCT:   (VLEVEL: LEVRANGE; DPLMT: ADDRRANGE);
                     INDRCT: (IDPLMT: ADDRRANGE))
       END;

TESTP = ^ TESTPOINTER;
TESTPOINTER = RECORD
                ELT1,ELT2 : STP;
                LASTTESTP : TESTP
              END;

                                                (*LABELS*)
LBP = ^ CODELABEL;
CODELABEL = RECORD
              CASE DEFINED: BOOLEAN OF
                FALSE: (REFLIST: ADDRRANGE);
                TRUE:  (OCCURIC: ADDRRANGE; JTABINX: JTABRANGE)
            END;

LABELP = ^ USERLABEL;
USERLABEL = RECORD
              LABVAL: INTEGER;
              NEXTLAB: LABELP;
              CODELBP: LBP
            END;

REFARRAY = ARRAY[1..REFSPERBLK] OF
             RECORD
               KEY,OFFSET: INTEGER
```

```
                END;

     CODEARRAY = PACKED ARRAY [0..MAXCODE] OF CHAR;
     SYMBUFARRAY = PACKED ARRAY [CURSRANGE] OF CHAR;

     UNITFILE = (WORKCODE,SYSLIBRARY);

     LEXSTKREC = RECORD
                   DOLDTOP: DISPRANGE;
                   DOLDLEV: 0..MAXLEVEL;
                   POLDPROC,SOLDPROC: PROCRANGE;
                   DOLDSEG: SEGRANGE;
                   DLLC: ADDRRANGE;
                   BFSY: SYMBOL;
                   DFPROCP: CTP;
                   DMARKP: ^INTEGER;
                   ISSEGMENT: BOOLEAN;
                   PREVLEXSTACKP: ^LEXSTKREC
                 END;


(*-------------------------------------------------------------------*)

VAR

    CODEP: ^ CODEARRAY;              (*CODE BUFFER UNTIL WRITEOUT*)
    SYMBUFP: ^ SYMBUFARRAY;          (*SYMBOLIC BUFFER...ASCII OR CODED*)

    GATTR: ATTR;                     (*DESCRIBES CURRENT EXPRESSION*)

    TOP: DISPRANGE;                  (*TOP OF DISPLAY*)
    LC,IC: ADDRRANGE;                (*LOCATION AND INSTRUCT COUNTERS*)
    TEST: BOOLEAN;
    INTPTR: STP;                     (*POINTER TO STANDARD INTEGER TYPE*)
    SEG: SEGRANGE;                   (*CURRENT SEGMENT NO.*)
                                     (*SCANNER GLOBALS...NEXT FOUR VARS*)
                                     (*MUST BE IN THIS ORDER FOR IDSEARCH*)
    SYMCURSOR: CURSRANGE;            (*CURRENT SCANNING INDEX IN SYMBUFP^*)
    SY: SYMBOL;                      (*SYMBOL FOUND BY INSYMBOL*)
    OP: OPERATOR;                    (*CLASSIFICATION OF LAST SYMBOL*)
    ID: ALPHA;                       (*LAST IDENTIFIER FOUND*)

    LGTH: INTEGER;                   (*LENGTH OF LAST STRING CONSTANT IN CHARS
                                       OR LEN OF LAST LONG INTEGER CONSTANT
                                        IN DIGITS*)
    VAL: VALU;                       (*VALUE OF LAST CONSTANT*)
    DISX: DISPRANGE;                 (*LEVEL OF LAST ID SEARCHED*)

    LCMAX: ADDRRANGE;                (*TEMPORARIES LOCATION COUNTER*)

                                     (*SWITCHES:*)

    PRTERR,GOTOOK,RANGECHECK,DEBUGGING,
    NOISY,CODEINSEG,IOCHECK,BPTONLINE,
    CLINKERINFO,DLINKERINFO,LIST,TINY,LSEPPROC,
    DP,INCLUDING,USING,NOSWAP,SEPPROC,
    STARTINGUP,INMODULE,ININTERFACE,
    LIBNOTOPEN,SYSCOMP,PUBLICPROCS,GETSTMTLEV: BOOLEAN;

                                     (*POINTERS:*)
```

```
(*INTPTR,*)REALPTR,LONGINTPTR,
CHARPTR,BOOLPTR,
TEXTPTR,NILPTR,
INTRACTVPTR,STRGPTR: STP;       (*POINTERS TO STANDARD IDS*)

UTYPPTR,UCSTPTR,UVARPTR,
UFLDPTR,UPRCPTR,UFCTPTR,        (*POINTERS TO UNDECLARED IDS*)
MODPTR,INPUTPTR,OUTPUTPTR,
OUTERBLOCK,FWPTR,USINGLIST: CTP;

GLOBTESTP: TESTP;               (*LAST TESTPOINTER*)

LEVEL: LEVRANGE;                (*CURRENT STATIC LEVEL*)
BEGSTMTLEV,STMTLEV: INTEGER;    (*CURRENT STATEMENT NESTING LEVEL*)
MARKP: ^INTEGER;                (*FOR MARKING HEAP*)
TOS: ^LEXSTKREC;                (*TOP OF LEX STACK*)
GLEV: DISPRANGE;                (*GLOBAL LEVEL OF DISPLAY*)
NEWBLOCK: BOOLEAN;              (*INDICATES NEED TO PUSH LEX STACK*)

NEXTSEG: SEGRANGE;              (*NEXT SEGMENT #*)
SEGINX: INTEGER;                (*CURRENT INDEX IN SEGMENT*)
SCONST: CSP;                    (*INSYMBOL STRING RESULTS*)

LOWTIME,LINEINFO,SCREENDOTS,STARTDOTS,SYMBLK,SMALLESTSPACE: INTEGER;
LINESTART: CURSRANGE;

CURPROC,NEXTPROC: PROCRANGE;     (*PROCEDURE NUMBER ASSIGNMENT*)

CONSTBEGSYS,SIMPTYPEBEGSYS,TYPEBEGSYS,
BLOCKBEGSYS,SELECTSYS,FACBEGSYS,STATBEGSYS,TYPEDELS: SETOFSYS;
VARS: SETOFIDS;

DISPLAY: ARRAY [DISPRANGE] OF
            RECORD
              FNAME: CTP;
              CASE OCCUR: WHERE OF
                BLCK: (FFILE: CTP; FLABEL: LABELP);
                CREC: (CLEV: LEVRANGE; CDSPL: ADDRRANGE);
                VREC: (VDSPL: ADDRRANGE)
              END;

PFNUMOF: NONRESPFLIST;

PROCTABLE: ARRAY [PROCRANGE] OF INTEGER;

SEGTABLE: ARRAY [SEGRANGE] OF
            RECORD
              DISKADDR,CODELENG: INTEGER;
              SEGNAME: ALPHA;
              SEGKIND,
              TEXTADDR: INTEGER
            END (*SEGTABLE*) ;

COMMENT: ^STRING;
SYSTEMLIB: STRING[40];
NEXTJTAB: JTABRANGE;
JTAB: ARRAY [JTABRANGE] OF INTEGER;

REFFILE: FILE;
NREFS,REFBLK: INTEGER;
```

```
    REFLIST: ^REFARRAY;
    OLDSYMBLK,PREVSYMBLK: INTEGER;
    OLDSYMCURSOR,OLDLINESTART,PREVSYMCURSOR,PREVLINESTART: CURSRANGE;
    USEFILE: UNITFILE;
    INCLFILE,LIBRARY: FILE;
    LP: TEXT;

    CURBYTE, CURBLK: INTEGER;
    DISKBUF: PACKED ARRAY [0..511] OF CHAR;

(*-----------------------------------------------------------------*)

(* FORWARD DECLARED PROCEDURES NEEDED BY COMPINIT *)

PROCEDURE ERROR(ERRORNUM: INTEGER);
  FORWARD;
PROCEDURE GETNEXTPAGE;
  FORWARD;
PROCEDURE PRINTLINE;
  FORWARD;
PROCEDURE ENTERID(FCP: CTP);
  FORWARD;
PROCEDURE INSYMBOL;
  FORWARD;

(* FORWARD DECLARED PROCEDURES USED IN BOTH DECLARATIONPART AND BODYPART *)

PROCEDURE SEARCHSECTION(FCP:CTP; VAR FCP1: CTP);
  FORWARD;
PROCEDURE SEARCHID(FIDCLS: SETOFIDS; VAR FCP: CTP);
  FORWARD;
PROCEDURE GETBOUNDS(FSP: STP; VAR FMIN,FMAX: INTEGER);
  FORWARD;
PROCEDURE SKIP(FSYS: SETOFSYS);
  FORWARD;
FUNCTION PAOFCHAR(FSP: STP): BOOLEAN;
  FORWARD;
FUNCTION STRGTYPE(FSP: STP): BOOLEAN;
  FORWARD;
FUNCTION DECSIZE(I: INTEGER): INTEGER;
  FORWARD;
PROCEDURE CONSTANT(FSYS: SETOFSYS; VAR FSP: STP; VAR FVALU: VALU);
  FORWARD;
FUNCTION COMPTYPES(FSP1,FSP2: STP): BOOLEAN;
  FORWARD;
PROCEDURE GENBYTE(FBYTE: INTEGER);
  FORWARD;
PROCEDURE GENWORD(FWORD: INTEGER);
  FORWARD;
PROCEDURE WRITETEXT;
  FORWARD;
PROCEDURE WRITECODE(FORCEBUF: BOOLEAN);
  FORWARD;
PROCEDURE BLOCK(FSYS: SETOFSYS);
  FORWARD;

(* $I COMPINIT.TEXT*)

SEGMENT PROCEDURE COMPINIT;
```

```
PROCEDURE ENTSTDTYPES;
BEGIN
  NEW(INTPTR,SCALAR,STANDARD);
  WITH INTPTR^ DO
    BEGIN SIZE := INTSIZE; FORM := SCALAR; SCALKIND := STANDARD END;
  NEW(REALPTR,SCALAR,STANDARD);
  WITH REALPTR^ DO
    BEGIN SIZE := REALSIZE; FORM := SCALAR; SCALKIND := STANDARD END;
  NEW(LONGINTPTR,LONGINT);
  WITH LONGINTPTR^ DO
    BEGIN SIZE := INTSIZE; FORM := LONGINT END;
  NEW(CHARPTR,SCALAR,STANDARD);
  WITH CHARPTR^ DO
    BEGIN SIZE := CHARSIZE; FORM := SCALAR; SCALKIND := STANDARD END;
  NEW(BOOLPTR,SCALAR,DECLARED);
  WITH BOOLPTR^ DO
    BEGIN SIZE := BOOLSIZE; FORM := SCALAR; SCALKIND := DECLARED END;
  NEW(NILPTR,POINTER);
  WITH NILPTR^ DO
    BEGIN SIZE := PTRSIZE; FORM := POINTER; ELTYPE := NIL END;
  NEW(TEXTPTR,FILES);
  WITH TEXTPTR^ DO
    BEGIN SIZE := FILESIZE+CHARSIZE; FORM := FILES; FILTYPE := CHARPTR END;
  NEW(INTRACTVPTR,FILES);
  WITH INTRACTVPTR^ DO
    BEGIN SIZE := FILESIZE+CHARSIZE; FORM := FILES; FILTYPE := CHARPTR END;
  NEW(STRGPTR,ARRAYS,TRUE,TRUE);
  WITH STRGPTR^ DO
    BEGIN FORM := ARRAYS; SIZE := (DEFSTRGLGTH + CHRSPERWD) DIV CHRSPERWD;
      AISPACKD := TRUE; AISSTRNG := TRUE; INXTYPE := INTPTR;
      ELWIDTH := BITSPERCHR; ELSPERWD := CHRSPERWD;
      AELTYPE := CHARPTR; MAXLENG := DEFSTRGLGTH;
    END
END (*ENTSTDTYPES*) ;

PROCEDURE ENTSTDNAMES;
  VAR CP,CP1: CTP; I: INTEGER;
BEGIN
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'INTEGER '; IDTYPE := INTPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'REAL    '; IDTYPE := REALPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'CHAR    '; IDTYPE := CHARPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'BOOLEAN '; IDTYPE := BOOLPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'STRING  '; IDTYPE := STRGPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
```

```
    BEGIN NAME := 'TEXT    '; IDTYPE := TEXTPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(CP,TYPES);
  WITH CP^ DO
    BEGIN NAME := 'INTERACT'; IDTYPE := INTRACTVPTR; KLASS := TYPES END;
  ENTERID(CP);
  NEW(INPUTPTR,FORMALVARS,FALSE);
  WITH INPUTPTR^ DO
    BEGIN NAME := 'INPUT   '; IDTYPE := TEXTPTR; KLASS := FORMALVARS;
      VLEV := 0; VADDR := 2
    END;
  ENTERID(INPUTPTR);
  NEW(OUTPUTPTR,FORMALVARS,FALSE);
  WITH OUTPUTPTR^ DO
    BEGIN NAME := 'OUTPUT  '; IDTYPE := TEXTPTR; KLASS := FORMALVARS;
      VLEV := 0; VADDR := 3
    END;
  ENTERID(OUTPUTPTR);
  NEW(CP,FORMALVARS,FALSE);
  WITH CP^ DO
    BEGIN NAME := 'KEYBOARD'; IDTYPE := TEXTPTR; KLASS := FORMALVARS;
      VLEV := 0; VADDR := 4
    END;
  ENTERID(CP);
  CP1 := NIL;
  FOR I := 0 TO 1 DO
    BEGIN NEW(CP,KONST);
      WITH CP^ DO
        BEGIN IDTYPE := BOOLPTR;
          IF I = 0 THEN NAME := 'FALSE   '
          ELSE NAME := 'TRUE    ';
          NEXT := CP1; VALUES.IVAL := I; KLASS := KONST
        END;
      ENTERID(CP); CP1 := CP
    END;
  BOOLPTR^.FCONST := CP;
  NEW(CP,KONST);
  WITH CP^ DO
    BEGIN NAME := 'NIL     '; IDTYPE := NILPTR;
      NEXT := NIL; VALUES.IVAL := 0; KLASS := KONST
    END;
  ENTERID(CP);
  NEW(CP,KONST);
  WITH CP^ DO
    BEGIN
      NAME := 'MAXINT  '; IDTYPE := INTPTR;
      KLASS := KONST; VALUES.IVAL := MAXINT
    END;
  ENTERID(CP);
END (*ENTSTDNAMES*) ;

PROCEDURE ENTUNDECL;
BEGIN
  NEW(UTYPPTR,TYPES);
  WITH UTYPPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL; KLASS := TYPES END;
  NEW(UCSTPTR,KONST);
  WITH UCSTPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL; NEXT := NIL;
      VALUES.IVAL := 0; KLASS := KONST
```

```
    END;
  NEW(UVARPTR,ACTUALVARS,FALSE);
  WITH UVARPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL;
      NEXT := NIL; VLEV := 0; VADDR := 0; KLASS := ACTUALVARS
    END;
  NEW(UFLDPTR,FIELD);
  WITH UFLDPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL; NEXT := NIL;
      FLDADDR := 0; KLASS := FIELD
    END;
  NEW(UPRCPTR,PROC,DECLARED,ACTUAL,FALSE);
  WITH UPRCPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL; FORWDECL := FALSE;
      NEXT := NIL; INSCOPE := FALSE; LOCALLC := 0; EXTURNAL := FALSE;
      PFLEV := 0; PFNAME := 0; PFSEG := 0;
      KLASS := PROC; PFDECKIND := DECLARED; PFKIND := ACTUAL
    END;
  NEW(UFCTPTR,FUNC,DECLARED,ACTUAL,FALSE);
  WITH UFCTPTR^ DO
    BEGIN NAME := '        '; IDTYPE := NIL; NEXT := NIL;
      FORWDECL := FALSE; EXTURNAL := FALSE; INSCOPE := FALSE; LOCALLC := 0;
      PFLEV := 0; PFNAME := 0; PFSEG := 0;
      KLASS := FUNC; PFDECKIND := DECLARED; PFKIND := ACTUAL
    END
END (*ENTUNDECL*) ;

PROCEDURE ENTSPCPROCS;
  LABEL 1;
  VAR LCP: CTP; I: INTEGER; ISFUNC: BOOLEAN;
      NA: ARRAY [1..43] OF ALPHA;

BEGIN
  NA[ 1] := 'READ    '; NA[ 2] := 'READLN  '; NA[ 3] := 'WRITE   ';
  NA[ 4] := 'WRITELN '; NA[ 5] := 'EOF     '; NA[ 6] := 'EOLN    ';
  NA[ 7] := 'PRED    '; NA[ 8] := 'SUCC    '; NA[ 9] := 'ORD     ';
  NA[10] := 'SQR     '; NA[11] := 'ABS     '; NA[12] := 'NEW     ';
  NA[13] := 'UNITREAD'; NA[14] := 'UNITWRIT'; NA[15] := 'CONCAT  ';
  NA[16] := 'LENGTH  '; NA[17] := 'INSERT  '; NA[18] := 'DELETE  ';
  NA[19] := 'COPY    '; NA[20] := 'POS     '; NA[21] := 'MOVELEFT';
  NA[22] := 'MOVERIGH'; NA[23] := 'EXIT    '; NA[24] := 'IDSEARCH';
  NA[25] := 'TREESEAR'; NA[26] := 'TIME    '; NA[27] := 'FILLCHAR';
  NA[28] := 'OPENNEW '; NA[29] := 'OPENOLD '; NA[30] := 'REWRITE ';
  NA[31] := 'CLOSE   '; NA[32] := 'SEEK    '; NA[33] := 'RESET   ';
  NA[34] := 'GET     '; NA[35] := 'PUT     '; NA[36] := 'SCAN    ';
  NA[37] := 'BLOCKREA'; NA[38] := 'BLOCKWRI'; NA[39] := 'TRUNC   ';
  NA[40] := 'PAGE    '; NA[41] := 'SIZEOF  '; NA[42] := 'STR     ';
  NA[43] := 'GOTOXY  ';
  FOR I := 1 TO 43 DO
    BEGIN
      IF TINY THEN
          IF I IN [2,7,8,10,13,17,18,19,20,32,34,35,40,42,43] THEN
            GOTO 1;
      ISFUNC := I IN [5,6,7,8,9,10,11,15,16,19,20,25,36,37,38,39,41];
      IF ISFUNC THEN NEW(LCP,FUNC,SPECIAL)
      ELSE NEW(LCP,PROC,SPECIAL);
      WITH LCP^ DO
        BEGIN NAME := NA[I]; NEXT := NIL; IDTYPE := NIL;
          IF ISFUNC THEN KLASS := FUNC ELSE KLASS := PROC;
          PFDECKIND := SPECIAL; KEY := I
```

```
          END;
        ENTERID(LCP);
1:    END
    END (*ENTSPCPROCS*) ;

  PROCEDURE ENTSTDPROCS;
    VAR LCP,PARAM: CTP; LSP,FTYPE: STP; I: INTEGER; ISPROC: BOOLEAN;
        NA: ARRAY [1..19] OF ALPHA;
  BEGIN
    NA[ 1] := 'ODD     '; NA[ 2] := 'CHR     '; NA[ 3] := 'MEMAVAIL';
    NA[ 4] := 'ROUND   '; NA[ 5] := 'SIN     '; NA[ 6] := 'COS     ';
    NA[ 7] := 'LOG     '; NA[ 8] := 'ATAN    '; NA[ 9] := 'LN      ';
    NA[10] := 'EXP     '; NA[11] := 'SQRT    '; NA[12] := 'MARK    ';
    NA[13] := 'RELEASE '; NA[14] := 'IORESULT'; NA[15] := 'UNITBUSY';
    NA[16] := 'PWROFTEN'; NA[17] := 'UNITWAIT'; NA[18] := 'UNITCLEA';
    NA[19] := 'HALT    ';
    FOR I := 1 TO 19 DO
      BEGIN ISPROC := I IN [12,13,17,18,19];
        CASE I OF
          1:  BEGIN FTYPE := BOOLPTR; NEW(PARAM,ACTUALVARS,FALSE);
                WITH PARAM^ DO
                  BEGIN IDTYPE := INTPTR; KLASS := ACTUALVARS END
              END;
          2:  FTYPE := CHARPTR;
          3:  BEGIN FTYPE := INTPTR; PARAM := NIL END;
          4:  BEGIN FTYPE := INTPTR; NEW(PARAM,ACTUALVARS,FALSE);
                WITH PARAM^ DO BEGIN IDTYPE := REALPTR; KLASS := ACTUALVARS END
              END;
          5:  FTYPE := REALPTR;
         12:  BEGIN FTYPE := NIL; NEW(PARAM,FORMALVARS,FALSE); NEW(LSP,POINTER);
                WITH LSP^ DO
                  BEGIN SIZE := PTRSIZE; FORM := POINTER; ELTYPE := NIL END;
                WITH PARAM^ DO BEGIN IDTYPE := LSP; KLASS := FORMALVARS END
              END;
         14:  BEGIN FTYPE := INTPTR; PARAM := NIL END;
         15:  BEGIN FTYPE := BOOLPTR; NEW(PARAM,ACTUALVARS,FALSE);
              WITH PARAM^ DO
                BEGIN IDTYPE := INTPTR; KLASS := ACTUALVARS END;
              END;
         16:  FTYPE := REALPTR;
         17:  FTYPE := NIL;
         19:  BEGIN FTYPE := NIL; PARAM := NIL END
        END (*PARAM AND TYPE CASES*) ;
        IF ISPROC THEN NEW(LCP,PROC,STANDARD)
        ELSE NEW(LCP,FUNC,STANDARD);
        WITH LCP^ DO
          BEGIN NAME := NA[I]; PFDECKIND := STANDARD; CSPNUM := I + 20;
            IF ISPROC THEN KLASS := PROC ELSE KLASS := FUNC;
            IF PARAM <> NIL THEN PARAM^.NEXT := NIL;
            IDTYPE := FTYPE; NEXT := PARAM
          END;
        ENTERID(LCP)
      END
    END (*ENTSTDPROCS*) ;

  PROCEDURE INITSCALARS;
   VAR I: NONRESIDENT;
  BEGIN
    FWPTR := NIL; MODPTR := NIL; GLOBTESTP := NIL;
    LINESTART := 0; LINEINFO := LCAFTERMARKSTACK; LIST := FALSE;
```

```
    SYMBLK := 2; SCREENDOTS := 0; STARTDOTS := 0;
    FOR SEG := 0 TO MAXSEG DO
      WITH SEGTABLE[SEG] DO
        BEGIN DISKADDR := 0; CODELENG := 0; SEGNAME := '        ';
          SEGKIND := 0; TEXTADDR := 0
        END;
    USINGLIST := NIL;
    IF USERINFO.STUPID THEN SYSTEMLIB := '*SYSTEM.PASCAL'
    ELSE SYSTEMLIB := '*SYSTEM.LIBRARY';
    LC := LCAFTERMARKSTACK; IOCHECK := TRUE; DP := TRUE;
    SEGINX := 0; NEXTJTAB := 1; NEXTPROC := 2; CURPROC := 1;
    NEW(SCONST); NEW(SYMBUFP); NEW(CODEP);
    CLINKERINFO := FALSE; DLINKERINFO := FALSE;
    SEG := 1; NEXTSEG := 10; CURBLK := 1; CURBYTE := 0; LSEPPROC := FALSE;
    STARTINGUP := TRUE; NOISY := NOT USERINFO.SLOWTERM; SEPPROC := FALSE;
    NOSWAP := TRUE; DEBUGGING := FALSE; BPTONLINE := FALSE; INMODULE := FALSE;
    GOTOOK := FALSE; RANGECHECK := TRUE; SYSCOMP := FALSE; TINY := FALSE;
    CODEINSEG := FALSE; PRTERR := TRUE; INCLUDING := FALSE; USING := FALSE;
    FOR I := SEEK TO DECOPS DO PFNUMOF[I] := 0;
    COMMENT := NIL; LIBNOTOPEN := TRUE;
    GETSTMTLEV := TRUE; BEGSTMTLEV := 0
  END (*INITSCALARS*) ;

  PROCEDURE INITSETS;

  BEGIN
    CONSTBEGSYS := [ADDOP,INTCONST,REALCONST,STRINGCONST,IDENT];
    SIMPTYPEBEGSYS := [LPARENT] + CONSTBEGSYS;
    TYPEBEGSYS := [ARROW,PACKEDSY,ARRAYSY,RECORDSY,SETSY,FILESY]
                  + SIMPTYPEBEGSYS;
    TYPEDELS := [ARRAYSY,RECORDSY,SETSY,FILESY];
    BLOCKBEGSYS := [USESSY,LABELSY,CONSTSY,TYPESY,VARSY,
                    PROCSY,FUNCSY,PROGSY,BEGINSY];
    SELECTSYS := [ARROW,PERIOD,LBRACK];
    FACBEGSYS := [INTCONST,REALCONST,LONGCONST,STRINGCONST,IDENT,
                  LPARENT,LBRACK,NOTSY];
    STATBEGSYS := [BEGINSY,GOTOSY,IFSY,WHILESY,REPEATSY,FORSY,WITHSY,CASESY];
    VARS := [FORMALVARS,ACTUALVARS]
  END (*INITSETS*) ;

BEGIN (*COMPINIT*)
  INITSCALARS; INITSETS;
  LEVEL := 0; TOP := 0;
  IF NOISY THEN
    BEGIN
      FOR IC := 1 TO 7 DO WRITELN(OUTPUT);
      WRITELN(OUTPUT,'PASCAL Compiler [I.5] (Unit Compiler)');
      WRITE(OUTPUT,'<   0>')
    END;
  WITH DISPLAY[0] DO
    BEGIN FNAME := NIL; FFILE := NIL; FLABEL := NIL; OCCUR := BLCK END;
  SMALLESTSPACE:=MEMAVAIL;
  GETNEXTPAGE;
  INSYMBOL;
  ENTSTDTYPES;   ENTSTDNAMES;   ENTUNDECL;
  ENTSPCPROCS;   ENTSTDPROCS;
  IF SYSCOMP THEN
    BEGIN OUTERBLOCK := NIL; SEG := 0; NEXTSEG := 1;
      GLEV :=1; BLOCKBEGSYS := BLOCKBEGSYS + [UNITSY,SEPARATSY]
    END
```

```
  ELSE
    BEGIN TOP := 1; LEVEL := 1;
      WITH DISPLAY[1] DO
        BEGIN FNAME := NIL; FFILE := NIL;
          FLABEL := NIL; OCCUR := BLCK
        END;
      LC := LC+2; GLEV := 3; (*KEEP STACK STRAIGHT FOR NOW*)
      NEW(OUTERBLOCK,PROC,DECLARED,ACTUAL,FALSE);
      WITH OUTERBLOCK^ DO
        BEGIN NEXT := NIL; LOCALLC := LC;
          NAME := 'PROGRAM '; IDTYPE := NIL; KLASS := PROC;
          PFDECKIND := DECLARED; PFLEV := 0; PFNAME := 1; PFSEG := SEG;
          PFKIND := ACTUAL; FORWDECL := FALSE; EXTURNAL := FALSE;
          INSCOPE := TRUE
        END
    END;
  IF SY = PROGSY THEN
    BEGIN INSYMBOL;
      IF SY = IDENT THEN
        BEGIN SEGTABLE[SEG].SEGNAME := ID;
          IF OUTERBLOCK <> NIL THEN
            BEGIN
              OUTERBLOCK^.NAME := ID;
              ENTERID(OUTERBLOCK) (*ALLOWS EXIT ON PROGRAM NAME*)
            END
        END
      ELSE ERROR(2); INSYMBOL;
      IF SY = LPARENT THEN
        BEGIN
          REPEAT INSYMBOL
          UNTIL SY IN [RPARENT,SEMICOLON]+BLOCKBEGSYS;
          IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
        END;
      IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14)
    END;
  MARK(MARKP);
  NEW(TOS);
  WITH TOS^ DO  (*MAKE LEXSTKREC FOR OUTERBLOCK*)
    BEGIN
      PREVLEXSTACKP:=NIL;
      BFSY:=PERIOD;
      DFPROCP:=OUTERBLOCK;
      DLLC:=LC;
      DOLDLEV:=LEVEL;
      DOLDTOP:=TOP;
      POLDPROC:=CURPROC;
      ISSEGMENT:=FALSE;
      DMARKP:=MARKP;
    END;
END (*COMPINIT*) ;

(* $I DECPART.A.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

SEGMENT PROCEDURE DECLARATIONPART(FSYS: SETOFSYS);
VAR LSY: SYMBOL;
    NOTDONE: BOOLEAN;
    DUMMYVAR: ARRAY[0..0] OF INTEGER; (*FOR PRETTY DISPLAY OF STACK AND HEAP *)
```

```
PROCEDURE TYP(FSYS: SETOFSYS; VAR FSP: STP; VAR FSIZE: ADDRRANGE);
  VAR LSP,LSP1,LSP2: STP; OLDTOP: DISPRANGE; LCP: CTP;
      LSIZE,DISPL: ADDRRANGE; LMIN,LMAX: INTEGER;
      PACKING: BOOLEAN; NEXTBIT,NUMBITS: BITRANGE;

  PROCEDURE SIMPLETYPE(FSYS:SETOFSYS; VAR FSP:STP; VAR FSIZE:ADDRRANGE);
    VAR LSP,LSP1: STP; LCP,LCP1: CTP; TTOP: DISPRANGE;
        LCNT: INTEGER; LVALU: VALU;
  BEGIN FSIZE := 1;
    IF NOT (SY IN SIMPTYPEBEGSYS) THEN
      BEGIN ERROR(1); SKIP(FSYS + SIMPTYPEBEGSYS) END;
    IF SY IN SIMPTYPEBEGSYS THEN
      BEGIN
        IF SY = LPARENT THEN
          BEGIN TTOP := TOP;
            WHILE DISPLAY[TOP].OCCUR <> BLCK DO TOP := TOP - 1;
            NEW(LSP,SCALAR,DECLARED);
            WITH LSP^ DO
              BEGIN SIZE := INTSIZE; FORM := SCALAR;
                SCALKIND := DECLARED
              END;
            LCP1 := NIL; LCNT := 0;
            REPEAT INSYMBOL;
              IF SY = IDENT THEN
                BEGIN NEW(LCP,KONST);
                  WITH LCP^ DO
                    BEGIN NAME := ID; IDTYPE := LSP; NEXT := LCP1;
                      VALUES.IVAL := LCNT; KLASS := KONST
                    END;
                  ENTERID(LCP);
                  LCNT := LCNT + 1;
                  LCP1 := LCP; INSYMBOL
                END
              ELSE ERROR(2);
              IF NOT (SY IN FSYS + [COMMA,RPARENT]) THEN
                BEGIN ERROR(6); SKIP(FSYS + [COMMA,RPARENT]) END
            UNTIL SY <> COMMA;
            LSP^.FCONST := LCP1; TOP := TTOP;
            IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
          END
        ELSE
          BEGIN
            IF SY = IDENT THEN
              BEGIN SEARCHID([TYPES,KONST],LCP);
                INSYMBOL;
                IF LCP^.KLASS = KONST THEN
                  BEGIN NEW(LSP,SUBRANGE);
                    WITH LSP^, LCP^ DO
                      BEGIN RANGETYPE := IDTYPE; FORM := SUBRANGE;
                        IF STRGTYPE(RANGETYPE) THEN
                          BEGIN ERROR(148); RANGETYPE := NIL END;
                        MIN := VALUES; SIZE := INTSIZE
                      END;
                    IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
                    CONSTANT(FSYS,LSP1,LVALU);
                    LSP^.MAX := LVALU;
                    IF LSP^.RANGETYPE <> LSP1 THEN ERROR(107)
                  END
                ELSE
```

```
          BEGIN LSP := LCP^.IDTYPE;
            IF (LSP = STRGPTR) AND (SY = LBRACK) THEN
              BEGIN INSYMBOL;
                CONSTANT(FSYS + [RBRACK],LSP1,LVALU);
                IF LSP1 = INTPTR THEN
                  BEGIN
                    IF (LVALU.IVAL <= 0) OR
                       (LVALU.IVAL > STRGLGTH) THEN
                      BEGIN ERROR(203);
                        LVALU.IVAL := DEFSTRGLGTH
                      END;
                    IF LVALU.IVAL <> DEFSTRGLGTH THEN
                      BEGIN NEW(LSP,ARRAYS,TRUE,TRUE);
                        LSP^ := STRGPTR^;
                        WITH LSP^,LVALU DO
                          BEGIN MAXLENG := IVAL;
                            SIZE := (IVAL+CHRSPERWD) DIV CHRSPERWD
                          END
                      END
                  END
                ELSE ERROR(15);
                IF SY = RBRACK THEN INSYMBOL ELSE ERROR(12)
              END
            ELSE
              IF LSP = INTPTR THEN
                IF SY = LBRACK THEN
                  BEGIN INSYMBOL;
                    NEW(LSP,LONGINT);
                    LSP^ := LONGINTPTR^;
                    CONSTANT(FSYS + [RBRACK],LSP1,LVALU);
                    IF LSP1 = INTPTR THEN
                      IF (LVALU.IVAL <= 0) OR
                         (LVALU.IVAL > MAXDEC) THEN ERROR(203)
                      ELSE
                        LSP^.SIZE := DECSIZE(LVALU.IVAL)
                    ELSE ERROR(15);
                    IF SY = RBRACK THEN INSYMBOL ELSE ERROR(12);
                  END
                ELSE
                  IF LSP^.FORM = FILES THEN
                    IF INMODULE THEN
                      IF NOT ININTERFACE THEN
                        ERROR(191); (*NO PRIVATE FILES*)
            IF LSP <> NIL THEN FSIZE := LSP^.SIZE
          END
        END (*SY = IDENT*)
      ELSE
        BEGIN NEW(LSP,SUBRANGE); LSP^.FORM := SUBRANGE;
          CONSTANT(FSYS + [COLON],LSP1,LVALU);
          IF STRGTYPE(LSP1) THEN
            BEGIN ERROR(148); LSP1 := NIL END;
          WITH LSP^ DO
            BEGIN RANGETYPE:=LSP1; MIN:=LVALU; SIZE:=INTSIZE END;
          IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
          CONSTANT(FSYS,LSP1,LVALU);
          LSP^.MAX := LVALU;
          IF LSP^.RANGETYPE <> LSP1 THEN ERROR(107)
        END;
      IF LSP <> NIL THEN
        WITH LSP^ DO
```

```
              IF FORM = SUBRANGE THEN
                IF RANGETYPE <> NIL THEN
                  IF RANGETYPE = REALPTR THEN ERROR(399)
                  ELSE
                    IF MIN.IVAL > MAX.IVAL THEN
                      BEGIN ERROR(102); MAX.IVAL := MIN.IVAL END
        END;
      FSP := LSP;
      IF NOT (SY IN FSYS) THEN
        BEGIN ERROR(6); SKIP(FSYS) END
    END
      ELSE FSP := NIL
END (*SIMPLETYPE*) ;

FUNCTION PACKABLE(FSP: STP): BOOLEAN;
  VAR LMIN,LMAX: INTEGER;
BEGIN PACKABLE := FALSE;
  IF (FSP <> NIL) AND PACKING THEN
    WITH FSP^ DO
      CASE FORM OF
        SUBRANGE,
        SCALAR:  IF (FSP <> INTPTR) AND (FSP <> REALPTR) THEN
                   BEGIN GETBOUNDS(FSP,LMIN,LMAX);
                     IF LMIN >= 0 THEN
                       BEGIN PACKABLE := TRUE;
                         NUMBITS := 1; LMIN := 1;
                         WHILE LMIN < LMAX DO
                           BEGIN LMIN := LMIN + 1;
                             LMIN := LMIN + LMIN - 1;
                             NUMBITS := NUMBITS + 1
                           END
                       END;
        POWER:   IF PACKABLE(ELSET) THEN
                   BEGIN GETBOUNDS(ELSET,LMIN,LMAX);
                     LMAX := LMAX + 1;
                     IF LMAX < BITSPERWD THEN
                       BEGIN PACKABLE := TRUE;
                         NUMBITS := LMAX
                       END
                   END
      END (* CASES *);
END (*PACKABLE*) ;

PROCEDURE FIELDLIST(FSYS: SETOFSYS; VAR FRECVAR: STP);
  VAR LCP,LCP1,NXT,NXT1,LAST: CTP; LSP,LSP1,LSP2,LSP3,LSP4: STP;
      MINSIZE,MAXSIZE,LSIZE: ADDRRANGE; LVALU: VALU;
      MAXBIT,MINBIT: BITRANGE;

  PROCEDURE ALLOCATE(FCP: CTP);
    VAR ONBOUND: BOOLEAN;
  BEGIN ONBOUND := FALSE;
    WITH FCP^ DO
      IF PACKABLE(IDTYPE) THEN
        BEGIN
          IF (NUMBITS + NEXTBIT) > BITSPERWD THEN
            BEGIN DISPL := DISPL + 1; NEXTBIT := 0; ONBOUND := TRUE END;
          FLDADDR := DISPL; FISPACKD := TRUE;
          FLDWIDTH := NUMBITS; FLDRBIT := NEXTBIT;
          NEXTBIT := NEXTBIT + NUMBITS
```

```
      END
    ELSE
      BEGIN DISPL := DISPL + ORD(NEXTBIT > 0);
        NEXTBIT := 0; ONBOUND := TRUE;
        FISPACKD := FALSE; FLDADDR := DISPL;
        IF IDTYPE <> NIL THEN
          DISPL := DISPL + IDTYPE^.SIZE
      END;
  IF ONBOUND AND (LAST <> NIL) THEN
    WITH LAST^ DO
      IF FISPACKD THEN
        IF FLDRBIT = 0 THEN FISPACKD := FALSE
        ELSE
          IF (FLDWIDTH <= 8) AND (FLDRBIT <= 8) THEN
            BEGIN FLDWIDTH := 8; FLDRBIT := 8 END
END (*ALLOCATE*) ;

PROCEDURE VARIANTLIST;
  VAR GOTTAGNAME: BOOLEAN;
BEGIN NEW(LSP,TAGFLD);
  WITH LSP^ DO
    BEGIN TAGFIELDP := NIL; FSTVAR := NIL; FORM := TAGFLD END;
  FRECVAR := LSP;
  INSYMBOL;
  IF SY = IDENT THEN
    BEGIN
      IF PACKING THEN NEW(LCP,FIELD,TRUE)
      ELSE NEW(LCP,FIELD,FALSE);
      WITH LCP^ DO
        BEGIN IDTYPE := NIL; KLASS:=FIELD;
          NEXT := NIL; FISPACKD := FALSE
        END;
      GOTTAGNAME := FALSE; PRTERR := FALSE;
      SEARCHID([TYPES],LCP1); PRTERR := TRUE;
      IF LCP1 = NIL THEN
        BEGIN GOTTAGNAME := TRUE;
          LCP^.NAME := ID; ENTERID(LCP); INSYMBOL;
          IF SY = COLON THEN INSYMBOL ELSE ERROR(5)
        END;
      IF SY = IDENT THEN
        BEGIN SEARCHID([TYPES],LCP1);
          LSP1 := LCP1^.IDTYPE;
          IF LSP1 <> NIL THEN
            BEGIN
              IF LSP1^.FORM <= SUBRANGE THEN
                BEGIN
                  IF COMPTYPES(REALPTR,LSP1) THEN ERROR(109);
                  LCP^.IDTYPE := LSP1; LSP^.TAGFIELDP := LCP;
                  IF GOTTAGNAME THEN ALLOCATE(LCP)
                END
              ELSE ERROR(110)
            END;
          INSYMBOL
        END
      ELSE BEGIN ERROR(2); SKIP(FSYS + [OFSY,LPARENT]) END
    END
  ELSE BEGIN ERROR(2); SKIP(FSYS + [OFSY,LPARENT]) END;
  LSP^.SIZE := DISPL + ORD(NEXTBIT > 0);
  IF SY = OFSY THEN INSYMBOL ELSE ERROR(8);
  LSP1 := NIL; MINSIZE := DISPL; MAXSIZE := DISPL;
```

```
      MINBIT := NEXTBIT; MAXBIT := NEXTBIT;
      REPEAT LSP2 := NIL;
        REPEAT CONSTANT(FSYS + [COMMA,COLON,LPARENT],LSP3,LVALU);
          IF LSP^.TAGFIELDP <> NIL THEN
            IF NOT COMPTYPES(LSP^.TAGFIELDP^.IDTYPE,LSP3) THEN
              ERROR(111);
          NEW(LSP3,VARIANT);
          WITH LSP3^ DO
            BEGIN NXTVAR := LSP1; SUBVAR := LSP2;
              VARVAL := LVALU; FORM := VARIANT
            END;
          LSP1 := LSP3; LSP2 := LSP3;
          TEST := SY <> COMMA;
          IF NOT TEST THEN INSYMBOL
        UNTIL TEST;
        IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
        IF SY = LPARENT THEN INSYMBOL ELSE ERROR(9);
        IF SY = RPARENT THEN LSP2 := NIL
        ELSE
          FIELDLIST(FSYS + [RPARENT,SEMICOLON],LSP2);
        IF DISPL > MAXSIZE THEN
          BEGIN MAXSIZE := DISPL; MAXBIT := NEXTBIT END
        ELSE
          IF (DISPL = MAXSIZE) AND (NEXTBIT > MAXBIT) THEN
            MAXBIT := NEXTBIT;
        WHILE LSP3 <> NIL DO
          BEGIN LSP4 := LSP3^.SUBVAR; LSP3^.SUBVAR := LSP2;
            LSP3^.SIZE := DISPL + ORD(NEXTBIT > 0);
            LSP3 := LSP4
          END;
        IF SY = RPARENT THEN
          BEGIN INSYMBOL;
            IF NOT (SY IN FSYS + [SEMICOLON]) THEN
              BEGIN ERROR(6); SKIP(FSYS + [SEMICOLON]) END
          END
        ELSE ERROR(4);
        TEST := SY <> SEMICOLON;
        IF NOT TEST THEN
          BEGIN INSYMBOL;
            DISPL := MINSIZE; NEXTBIT := MINBIT
          END
      UNTIL (TEST) OR (SY = ENDSY); (* <<<< SMF 2-28-78 *)
      DISPL := MAXSIZE; NEXTBIT := MAXBIT;
      LSP^.FSTVAR := LSP1
    END (*VARIANTLIST*) ;

  BEGIN (*FIELDLIST*)
    NXT1 := NIL; LSP := NIL; LAST := NIL;
    IF NOT (SY IN [IDENT,CASESY]) THEN
      BEGIN ERROR(19); SKIP(FSYS + [IDENT,CASESY]) END;
    WHILE SY = IDENT DO
      BEGIN NXT := NXT1;
        REPEAT
          IF SY = IDENT THEN
            BEGIN
              IF PACKING THEN NEW(LCP,FIELD,TRUE)
              ELSE NEW(LCP,FIELD,FALSE);
              WITH LCP^ DO
                BEGIN NAME := ID; IDTYPE := NIL; NEXT := NXT;
                  KLASS := FIELD; FISPACKD := FALSE
```

```
              END;
            NXT := LCP;
            ENTERID(LCP);
            INSYMBOL
          END
        ELSE ERROR(2);
        IF NOT (SY IN [COMMA,COLON]) THEN
          BEGIN ERROR(6); SKIP(FSYS + [COMMA,COLON,SEMICOLON,CASESY]) END;
        TEST := SY <> COMMA;
        IF NOT TEST  THEN INSYMBOL
      UNTIL TEST;
      IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
      TYP(FSYS + [CASESY,SEMICOLON],LSP,LSIZE);
      IF LSP <> NIL THEN
        IF LSP^.FORM = FILES THEN ERROR(108);
      WHILE NXT <> NXT1 DO
        WITH NXT^ DO
          BEGIN IDTYPE := LSP; ALLOCATE(NXT);
            IF NEXT = NXT1 THEN LAST := NXT;
            NXT := NEXT
          END;
      NXT1 := LCP;
      IF SY = SEMICOLON THEN
        BEGIN INSYMBOL;
          IF NOT (SY IN [IDENT,ENDSY,CASESY]) THEN (* <<<< SMF 2-28-78 *)
            BEGIN ERROR(19); SKIP(FSYS + [IDENT,CASESY]) END
        END
    END (*WHILE*);
  NXT := NIL;
  WHILE NXT1 <> NIL DO
    WITH NXT1^ DO
      BEGIN LCP := NEXT; NEXT := NXT; NXT := NXT1; NXT1 := LCP END;
  IF SY = CASESY THEN VARIANTLIST
  ELSE FRECVAR := NIL
END (*FIELDLIST*) ;

PROCEDURE POINTERTYPE;
BEGIN NEW(LSP,POINTER); FSP := LSP;
  WITH LSP^ DO
    BEGIN ELTYPE := NIL; SIZE := PTRSIZE; FORM := POINTER END;
  INSYMBOL;
  IF SY = IDENT THEN
    BEGIN PRTERR := FALSE;
      SEARCHID([TYPES],LCP); PRTERR := TRUE;
      IF LCP = NIL THEN   (*FORWARD REFERENCED TYPE ID*)
        BEGIN NEW(LCP,TYPES);
          WITH LCP^ DO
            BEGIN NAME := ID; IDTYPE := LSP;
              NEXT := FWPTR; KLASS := TYPES
            END;
          FWPTR := LCP
        END
      ELSE
        BEGIN
          IF LCP^.IDTYPE <> NIL THEN
            IF (LCP^.IDTYPE^.FORM <> FILES) OR SYSCOMP THEN
              LSP^.ELTYPE := LCP^.IDTYPE
            ELSE ERROR(108)
        END;
      INSYMBOL;
```

```
      END
    ELSE ERROR(2)
  END (*POINTERTYPE*) ;

BEGIN (*TYP*)
  PACKING := FALSE;
  IF NOT (SY IN TYPEBEGSYS) THEN
     BEGIN ERROR(10); SKIP(FSYS + TYPEBEGSYS) END;
  IF SY IN TYPEBEGSYS THEN
    BEGIN
      IF SY IN SIMPTYPEBEGSYS THEN SIMPLETYPE(FSYS,FSP,FSIZE)
      ELSE
(*^*)   IF SY = ARROW THEN POINTERTYPE
        ELSE
          BEGIN
            IF SY = PACKEDSY THEN
              BEGIN INSYMBOL; PACKING := TRUE;
                IF NOT (SY IN TYPEDELS) THEN
                  BEGIN ERROR(10); SKIP(FSYS + TYPEDELS) END
              END;
(*ARRAY*)   IF SY = ARRAYSY THEN
              BEGIN INSYMBOL;
                IF SY = LBRACK THEN INSYMBOL ELSE ERROR(11);
                LSP1 := NIL;
                REPEAT
                  IF PACKING THEN NEW(LSP,ARRAYS,TRUE,FALSE)
                  ELSE NEW(LSP,ARRAYS,FALSE);
                  WITH LSP^ DO
                    BEGIN AELTYPE := LSP1; INXTYPE := NIL;
                      IF PACKING THEN AISSTRNG := FALSE;
                      AISPACKD := FALSE;  FORM := ARRAYS
                    END;
                  LSP1 := LSP;
                  SIMPLETYPE(FSYS + [COMMA,RBRACK,OFSY],LSP2,LSIZE);
                  LSP1^.SIZE := LSIZE;
                  IF LSP2 <> NIL THEN
                    IF LSP2^.FORM <= SUBRANGE THEN
                      BEGIN
                        IF LSP2 = REALPTR THEN
                          BEGIN ERROR(109); LSP2 := NIL END
                        ELSE
                          IF LSP2 = INTPTR THEN
                            BEGIN ERROR(149); LSP2 := NIL END;
                        LSP^.INXTYPE := LSP2
                      END
                    ELSE BEGIN ERROR(113); LSP2 := NIL END;
                  TEST := SY <> COMMA;
                  IF NOT TEST THEN INSYMBOL
                UNTIL TEST;
                IF SY = RBRACK THEN INSYMBOL ELSE ERROR(12);
                IF SY = OFSY THEN INSYMBOL ELSE ERROR(8);
                TYP(FSYS,LSP,LSIZE);
                IF LSP <> NIL THEN
                  IF LSP^.FORM = FILES THEN ERROR(108);
                IF PACKABLE(LSP) THEN
                  IF NUMBITS + NUMBITS <= BITSPERWD THEN
                    WITH LSP1^ DO
                      BEGIN AISPACKD := TRUE;
                        ELSPERWD := BITSPERWD DIV NUMBITS;
                        ELWIDTH := NUMBITS
```

```
                    END;
                REPEAT
                  WITH LSP1^ DO
                    BEGIN LSP2 := AELTYPE; AELTYPE := LSP;
                      IF INXTYPE <> NIL THEN
                        BEGIN GETBOUNDS(INXTYPE,LMIN,LMAX);
                          IF AISPACKD THEN
                            LSIZE := (LMAX-LMIN+ELSPERWD)
                                                DIV ELSPERWD
                          ELSE
                            LSIZE := LSIZE*(LMAX - LMIN + 1);
                          IF LSIZE <= 0 THEN
                            BEGIN ERROR(398); LSIZE := 1 END;
                          SIZE := LSIZE
                        END
                    END;
                  LSP := LSP1; LSP1 := LSP2
                UNTIL LSP1 = NIL
              END
            ELSE
(*RECORD*)    IF SY = RECORDSY THEN
                BEGIN INSYMBOL;
                  OLDTOP := TOP;
                  IF TOP < DISPLIMIT THEN
                    BEGIN TOP := TOP + 1;
                      WITH DISPLAY[TOP] DO
                        BEGIN FNAME := NIL; OCCUR := REC END
                    END
                  ELSE ERROR(250);
                  DISPL := 0; NEXTBIT := 0;
                  FIELDLIST(FSYS-[SEMICOLON]+[ENDSY],LSP1);
                  DISPL := DISPL + ORD(NEXTBIT > 0);
                  NEW(LSP,RECORDS);
                  WITH LSP^ DO
                    BEGIN FSTFLD := DISPLAY[TOP].FNAME;
                      RECVAR := LSP1; SIZE := DISPL;
                      FORM := RECORDS
                    END;
                  TOP := OLDTOP;
                  IF SY = ENDSY THEN INSYMBOL ELSE ERROR(13)
                END
              ELSE
(*SET*)         IF SY = SETSY THEN
                  BEGIN INSYMBOL;
                    IF SY = OFSY THEN INSYMBOL ELSE ERROR(8);
                    SIMPLETYPE(FSYS,LSP1,LSIZE);
                    IF LSP1 <> NIL THEN
                      IF (LSP1^.FORM > SUBRANGE) OR
                              (LSP1 = INTPTR) OR (LSP1 = REALPTR) THEN
                        BEGIN ERROR(115); LSP1 := NIL END
                      ELSE
                        IF LSP1 = REALPTR THEN
                          BEGIN ERROR(114); LSP1 := NIL END;
                    NEW(LSP,POWER);
                    WITH LSP^ DO
                      BEGIN ELSET := LSP1; FORM := POWER;
                        IF LSP1 <> NIL THEN
                          BEGIN GETBOUNDS(LSP1,LMIN,LMAX);
                            SIZE := (LMAX + BITSPERWD) DIV BITSPERWD;
                            IF SIZE > 255 THEN
```

```
                              BEGIN ERROR(169); SIZE := 1 END
                          END
                        ELSE SIZE := 0
                      END
                  END
                ELSE
  (*FILE*)        IF SY = FILESY THEN
                    BEGIN
                      IF INMODULE THEN
                        IF NOT ININTERFACE THEN
                          ERROR(191); (*NO PRIVATE FILES*)
                      INSYMBOL; NEW(LSP,FILES);
                      WITH LSP^ DO
                        BEGIN FORM := FILES; FILTYPE := NIL END;
                      IF SY = OFSY THEN
                        BEGIN INSYMBOL; TYP(FSYS,LSP1,LSIZE) END
                      ELSE LSP1 := NIL;
                      LSP^.FILTYPE := LSP1;
                      IF LSP1 <> NIL THEN
                        LSP^.SIZE := FILESIZE + LSP1^.SIZE
                      ELSE LSP^.SIZE := NILFILESIZE
                    END;
              FSP := LSP
            END;
        IF NOT (SY IN FSYS) THEN
          BEGIN ERROR(6); SKIP(FSYS) END
      END
    ELSE FSP := NIL;
    IF FSP = NIL THEN FSIZE := 1 ELSE FSIZE := FSP^.SIZE
  END (*TYP*) ;

(* $I DECPART.B.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

    PROCEDURE USESDECLARATION(MAGIC: BOOLEAN);
      LABEL 1;
      TYPE DCREC = RECORD
                     DISKADDR: INTEGER;
                     CODELENG: INTEGER
                   END;
      VAR SEGDICT: RECORD
                       DANDC: ARRAY[SEGRANGE] OF DCREC;
                       SEGNAME: ARRAY[SEGRANGE] OF ALPHA;
                       SEGKIND: ARRAY[SEGRANGE] OF INTEGER;
                       TEXTADDR: ARRAY[SEGRANGE] OF INTEGER;
                       FILLER: ARRAY[0..127] OF INTEGER
                   END;
          FOUND: BOOLEAN; BEGADDR: INTEGER;
          LCP: CTP; LLEXSTK: LEXSTKREC; LNAME: ALPHA;
          LSY: SYMBOL; LOP: OPERATOR; LID: ALPHA;

      PROCEDURE GETTEXT(VAR FOUND: BOOLEAN);
        VAR LCP: CTP; SEGINDEX: INTEGER;

      BEGIN FOUND := FALSE;
        LCP := MODPTR;
        WHILE (LCP <> NIL) AND NOT FOUND DO
          IF LCP^.NAME = ID THEN FOUND := TRUE ELSE LCP := LCP^.NEXT;
```

```
    IF FOUND THEN
      BEGIN
        LSEPPROC := SEGTABLE[LCP^.SEGID].SEGKIND = 4;
        IF NOT LSEPPROC THEN
          BEGIN SEG := LCP^.SEGID; NEXTPROC := 1 END;
        BEGADDR := SEGTABLE[LCP^.SEGID].TEXTADDR;
        USEFILE := WORKCODE;
      END
    ELSE
      BEGIN FOUND := TRUE;
        IF LIBNOTOPEN THEN
          BEGIN RESET(LIBRARY,SYSTEMLIB);
            IF IORESULT <> 0 THEN BEGIN ERROR(187); FOUND := FALSE END
            ELSE
              IF BLOCKREAD(LIBRARY,SEGDICT,1,0) <> 1 THEN
                BEGIN ERROR(187); FOUND := FALSE END;
          END;
        IF FOUND THEN
          BEGIN LIBNOTOPEN := FALSE;
            SEGINDEX := 0; FOUND := FALSE;
            WHILE (SEGINDEX <= MAXSEG) AND (NOT FOUND) DO
              IF MAGIC THEN
                  IF SEGDICT.SEGNAME[SEGINDEX] = LNAME THEN FOUND := TRUE
                  ELSE SEGINDEX := SEGINDEX + 1
              ELSE
                IF SEGDICT.SEGNAME[SEGINDEX] = ID THEN FOUND := TRUE
                ELSE SEGINDEX := SEGINDEX + 1;
            IF FOUND THEN
              BEGIN USEFILE := SYSLIBRARY;
                BEGADDR := SEGDICT.TEXTADDR[SEGINDEX];
                LSEPPROC := SEGDICT.SEGKIND[SEGINDEX] = 4;
                IF NOT LSEPPROC THEN
                  BEGIN
                    IF MAGIC THEN SEG := 6
                    ELSE
                      BEGIN SEG := NEXTSEG;
                        NEXTSEG := NEXTSEG + 1;
                        IF NEXTSEG > MAXSEG THEN ERROR(250)
                      END;
                    WITH SEGTABLE[SEG] DO
                      BEGIN DISKADDR := 0; CODELENG := 0;
                        SEGNAME := SEGDICT.SEGNAME[SEGINDEX];
                        IF INMODULE OR MAGIC THEN SEGKIND := 0
                        ELSE SEGKIND := SEGDICT.SEGKIND[SEGINDEX];
                        TEXTADDR := 0
                      END;
                    NEXTPROC := 1
                  END
              END
            ELSE ERROR(190) (*NOT IN LIBRARY*)
          END
      END;
    IF BEGADDR = 0 THEN BEGIN ERROR(195); FOUND := FALSE END;
    IF FOUND THEN
      BEGIN
        USING := TRUE;
        PREVSYMCURSOR := SYMCURSOR;
        PREVLINESTART := LINESTART;
        PREVSYMBLK := SYMBLK - 2;
        SYMBLK := BEGADDR; GETNEXTPAGE;
```

```
        INSYMBOL
      END
  END (*GETTEXT*) ;

BEGIN (*USESDECLARATION*)
  IF LEVEL <> 1 THEN ERROR(189);
  IF INMODULE AND NOT ININTERFACE THEN ERROR(192);
  IF NOT MAGIC THEN DLINKERINFO := TRUE;
  IF NOT USING THEN USINGLIST := NIL;
  REPEAT
    IF (NOT MAGIC) AND (SY <> IDENT) THEN ERROR(2)
    ELSE
      IF USING THEN
        BEGIN LCP := USINGLIST;
          WHILE LCP <> NIL DO
            IF LCP^.NAME = ID THEN GOTO 1
            ELSE LCP := LCP^.NEXT;
          ERROR(188)(*UNIT MUST BE PREDECLARED IN MAIN PROG*);
      1:
        END
      ELSE
        BEGIN
          IF MAGIC THEN
            BEGIN LNAME := 'TURTLE  ';
              LSY := SY; LOP := OP; LID := ID
            END
          ELSE
            BEGIN LNAME := ID;
              WRITELN(OUTPUT); WRITELN(OUTPUT,ID,' [',MEMAVAIL:5,' words]');
              WRITE(OUTPUT,'<',SCREENDOTS:4,'>')
            END;
          WITH LLEXSTK DO
            BEGIN DOLDSEG := SEG; SOLDPROC := NEXTPROC END;
          GETTEXT(FOUND);
          IF FOUND THEN
            BEGIN
              NEW(LCP,MODULE);
              WITH LCP^ DO
                BEGIN NAME := LNAME; NEXT := USINGLIST;
                  IDTYPE := NIL; KLASS := MODULE;
                  IF LSEPPROC THEN SEGID := -1 (*NO SEG*) ELSE SEGID := SEG
                END;
              ENTERID(LCP);
              USINGLIST := LCP;
              DECLARATIONPART(FSYS + [ENDSY]);
              IF NEXTPROC=1 (*NO PROCS DECLARED*) THEN
                LCP^.SEGID := -1; (*NO SEG*)
              SYMBLK := 9999; (*FORCE RETURN TO SOURCEFILE*)
              GETNEXTPAGE
            END;
          IF NOT LSEPPROC THEN
            WITH LLEXSTK DO
              BEGIN SEG := DOLDSEG;
                NEXTPROC := SOLDPROC
              END;
          LSEPPROC := FALSE;
        END;
    IF NOT MAGIC THEN
      BEGIN INSYMBOL;
        TEST := SY <> COMMA;
```

```
        IF TEST THEN
          IF SY <> SEMICOLON THEN ERROR(20)
          ELSE
        ELSE INSYMBOL
      END
  UNTIL TEST OR MAGIC;
  IF NOT MAGIC THEN
    IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14)
  ELSE BEGIN SY := LSY; OP := LOP; ID := LID END;
  IF NOT USING THEN
    BEGIN
      IF INMODULE THEN USINGLIST := NIL;
      CLOSE(LIBRARY,LOCK);
      LIBNOTOPEN := TRUE
    END
END (*USESDECLARATION*) ;

PROCEDURE LABELDECLARATION;
  VAR LLP: LABELP; REDEF: BOOLEAN;
BEGIN
  REPEAT
    IF SY = INTCONST THEN
      WITH DISPLAY[TOP] DO
        BEGIN LLP := FLABEL; REDEF := FALSE;
          WHILE (LLP <> NIL) AND NOT REDEF DO
            IF LLP^.LABVAL <> VAL.IVAL THEN
              LLP := LLP^.NEXTLAB
            ELSE BEGIN REDEF := TRUE; ERROR(166) END;
          IF NOT REDEF THEN
            BEGIN NEW(LLP);
              WITH LLP^ DO
                BEGIN LABVAL := VAL.IVAL;
                  CODELBP := NIL; NEXTLAB := FLABEL
                END;
              FLABEL := LLP
            END;
          INSYMBOL
        END
    ELSE ERROR(15);
    IF NOT ( SY IN FSYS + [COMMA, SEMICOLON] ) THEN
      BEGIN ERROR(6); SKIP(FSYS+[COMMA,SEMICOLON]) END;
    TEST := SY <> COMMA;
    IF NOT TEST THEN INSYMBOL
  UNTIL TEST;
  IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14)
END (* LABELDECLARATION *) ;

PROCEDURE CONSTDECLARATION;
  VAR LCP: CTP; LSP: STP; LVALU: VALU;
BEGIN
  IF SY <> IDENT THEN
    BEGIN ERROR(2); SKIP(FSYS + [IDENT]) END;
  WHILE SY = IDENT DO
    BEGIN NEW(LCP,KONST);
      WITH LCP^ DO
        BEGIN NAME := ID; IDTYPE := NIL;
          NEXT := NIL; KLASS := KONST
        END;
      INSYMBOL;
      IF (SY = RELOP) AND (OP = EQOP) THEN INSYMBOL ELSE ERROR(16);
```

```
      CONSTANT(FSYS + [SEMICOLON],LSP,LVALU);
      ENTERID(LCP);
      LCP^.IDTYPE := LSP; LCP^.VALUES := LVALU;
      IF SY = SEMICOLON THEN
        BEGIN INSYMBOL;
          IF NOT (SY IN FSYS + [IDENT]) THEN
            BEGIN ERROR(6); SKIP(FSYS + [IDENT]) END
        END
      ELSE
        IF NOT ((SY = ENDSY) AND (INMODULE)) THEN ERROR(14)
    END
END (*CONSTDECLARATION*) ;

PROCEDURE TYPEDECLARATION;
  VAR LCP,LCP1,LCP2: CTP; LSP: STP; LSIZE: ADDRRANGE;
BEGIN
  IF SY <> IDENT THEN
    BEGIN ERROR(2); SKIP(FSYS + [IDENT]) END;
  WHILE SY = IDENT DO
    BEGIN NEW(LCP,TYPES);
      WITH LCP^ DO
        BEGIN NAME := ID; IDTYPE := NIL; KLASS := TYPES END;
      INSYMBOL;
      IF (SY = RELOP) AND (OP = EQOP) THEN INSYMBOL ELSE ERROR(16);
      TYP(FSYS + [SEMICOLON],LSP,LSIZE);
      ENTERID(LCP);
      LCP^.IDTYPE := LSP;
      LCP1 := FWPTR;
      WHILE LCP1 <> NIL DO
        BEGIN
          IF LCP1^.NAME = LCP^.NAME THEN
            BEGIN
              LCP1^.IDTYPE^.ELTYPE := LCP^.IDTYPE;
              IF LCP1 <> FWPTR THEN
                LCP2^.NEXT := LCP1^.NEXT
              ELSE FWPTR := LCP1^.NEXT;
            END;
          LCP2 := LCP1; LCP1 := LCP1^.NEXT
        END;
      IF SY = SEMICOLON THEN
        BEGIN INSYMBOL;
          IF NOT (SY IN FSYS + [IDENT]) THEN
            BEGIN ERROR(6); SKIP(FSYS + [IDENT]) END
        END
      ELSE
        IF NOT ((SY = ENDSY) AND (INMODULE)) THEN ERROR(14)
    END;
  IF FWPTR <> NIL THEN
    BEGIN ERROR(117); FWPTR := NIL END
END (*TYPEDECLARATION*) ;

PROCEDURE VARDECLARATION;
  VAR LCP,NXT,IDLIST: CTP; LSP: STP; LSIZE: ADDRRANGE;
BEGIN NXT := NIL;
  REPEAT
    REPEAT
      IF SY = IDENT THEN
        BEGIN
          IF INMODULE THEN NEW(LCP,ACTUALVARS,TRUE)
          ELSE NEW(LCP,ACTUALVARS,FALSE);
```

```
            WITH LCP^ DO
             BEGIN NAME := ID; NEXT := NXT; KLASS := ACTUALVARS;
               IDTYPE := NIL; VLEV := LEVEL;
               IF INMODULE THEN
                 IF ININTERFACE THEN PUBLIC := TRUE
                 ELSE PUBLIC := FALSE
             END;
            ENTERID(LCP);
            NXT := LCP;
            INSYMBOL;
          END
        ELSE ERROR(2);
        IF NOT (SY IN FSYS + [COMMA,COLON] + TYPEDELS) THEN
          BEGIN ERROR(6); SKIP(FSYS+[COMMA,COLON,SEMICOLON]+TYPEDELS) END;
        TEST := SY <> COMMA;
        IF NOT TEST THEN INSYMBOL
      UNTIL TEST;
      IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
      IDLIST := NXT;
      TYP(FSYS + [SEMICOLON] + TYPEDELS,LSP,LSIZE);
      WHILE NXT <> NIL DO
        WITH  NXT^ DO
          BEGIN IDTYPE := LSP; VADDR := LC;
            LC := LC + LSIZE; NXT := NEXT;
            IF NEXT = NIL THEN
              IF LSP <> NIL THEN
                IF LSP^.FORM = FILES THEN
                    BEGIN (*PUT IDLIST INTO LOCAL FILE LIST*)
                      NEXT := DISPLAY[TOP].FFILE;
                      DISPLAY[TOP].FFILE := IDLIST
                    END
          END;
      IF SY = SEMICOLON THEN
        BEGIN INSYMBOL;
          IF NOT (SY IN FSYS + [IDENT]) THEN
            BEGIN ERROR(6); SKIP(FSYS + [IDENT]) END
        END
      ELSE
          IF NOT ((SY = ENDSY) AND (INMODULE)) THEN ERROR(14)
    UNTIL (SY <> IDENT) AND NOT (SY IN TYPEDELS);
  IF FWPTR <> NIL THEN
      BEGIN ERROR(117); FWPTR := NIL END
  END (*VARDECLARATION*) ;

(* $I DECPART.C.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

  PROCEDURE PROCDECLARATION(FSY: SYMBOL; SEGDEC: BOOLEAN);
    VAR LSY: SYMBOL; LCP,LCP1: CTP; LSP: STP;
        EXTONLY,FORW: BOOLEAN;
        LCM: ADDRRANGE;
        LLEXSTK: LEXSTKREC;

    PROCEDURE PARAMETERLIST(FSY: SETOFSYS; VAR FPAR: CTP; FCP: CTP);
      VAR LCP,LCP1,LCP2,LCP3: CTP; LSP: STP; LKIND: IDKIND;
        LLC,LEN : ADDRRANGE; COUNT : INTEGER;
    BEGIN LCP1 := NIL; LLC := LC;
      IF NOT (SY IN FSY + [LPARENT]) THEN
```

```
    BEGIN ERROR(7); SKIP(FSYS + FSY + [LPARENT]) END;
  IF SY = LPARENT THEN
    BEGIN IF FORW THEN ERROR(119);
      INSYMBOL;
      IF NOT (SY IN [IDENT,VARSY]) THEN
        BEGIN ERROR(7); SKIP(FSYS + [IDENT,RPARENT]) END;
      WHILE SY IN [IDENT,VARSY] DO
        BEGIN
          IF SY = VARSY THEN
            BEGIN LKIND := FORMAL; INSYMBOL END
          ELSE LKIND := ACTUAL;
          LCP2 := NIL;
          COUNT := 0;
          REPEAT
            IF SY <> IDENT THEN ERROR(2)
            ELSE
              BEGIN
                NEW(LCP,FORMALVARS,FALSE); (*MAY BE ACTUAL(SAME SIZE)*)
                WITH LCP^ DO
                  BEGIN NAME := ID; IDTYPE := NIL; NEXT := LCP2;
                    IF LKIND = FORMAL THEN KLASS := FORMALVARS
                    ELSE KLASS := ACTUALVARS; VLEV := LEVEL
                  END;
                ENTERID(LCP);
                LCP2 := LCP; COUNT := COUNT + 1;
                INSYMBOL
              END;
            IF NOT (SY IN FSYS + [COMMA,SEMICOLON,COLON]) THEN
              BEGIN ERROR(7);
                SKIP(FSYS + [COMMA,SEMICOLON,RPARENT,COLON])
              END;
            TEST := SY <> COMMA;
            IF NOT TEST THEN INSYMBOL
          UNTIL TEST;
          LSP := NIL;
          IF SY = COLON THEN
            BEGIN INSYMBOL;
              IF SY = IDENT THEN
                BEGIN
                  SEARCHID([TYPES],LCP);
                  INSYMBOL;
                  LSP := LCP^.IDTYPE;
                  LEN := PTRSIZE;
                  IF LSP <> NIL THEN
                    IF LKIND = ACTUAL THEN
                      IF LSP^.FORM = FILES THEN ERROR(121)
                      ELSE
                        IF LSP^.FORM <= POWER THEN LEN := LSP^.SIZE;
                  LC := LC + COUNT * LEN
                END
              ELSE ERROR(2)
            END
          ELSE
            IF LKIND = FORMAL THEN
              EXTONLY := TRUE
            ELSE ERROR(5);
          IF NOT (SY IN FSYS + [SEMICOLON,RPARENT]) THEN
            BEGIN ERROR(7); SKIP(FSYS + [SEMICOLON,RPARENT]) END;
          LCP3 := LCP2; LCP := NIL;
          WHILE LCP2 <> NIL DO
```

```
              BEGIN LCP := LCP2;
                WITH LCP2^ DO
                  BEGIN IDTYPE := LSP;
                    LCP2 := NEXT
                  END
              END;
            IF LCP <> NIL THEN
              BEGIN LCP^.NEXT := LCP1; LCP1 := LCP3 END;
            IF SY = SEMICOLON THEN
              BEGIN INSYMBOL;
                IF NOT (SY IN FSYS + [IDENT,VARSY]) THEN
                  BEGIN ERROR(7); SKIP(FSYS + [IDENT,RPARENT]) END
              END
          END (*WHILE*) ;
        IF SY = RPARENT THEN
          BEGIN INSYMBOL;
            IF NOT (SY IN FSY + FSYS) THEN
              BEGIN ERROR(6); SKIP(FSY + FSYS) END
          END
        ELSE ERROR(4);
        FCP^.LOCALLC := LC; LCP3 := NIL;
        WHILE LCP1 <> NIL DO
          WITH LCP1^ DO
            BEGIN LCP2 := NEXT; NEXT := LCP3;
              IF (IDTYPE <> NIL) THEN
                IF KLASS = FORMALVARS THEN
                  BEGIN VADDR := LLC; LLC := LLC + PTRSIZE END
                ELSE
                  IF KLASS = ACTUALVARS THEN
                    IF (IDTYPE^.FORM <= POWER) THEN
                      BEGIN VADDR := LLC; LLC := LLC + IDTYPE^.SIZE END
                    ELSE
                      BEGIN VADDR := LC;
                        LC := LC + IDTYPE^.SIZE;
                        LLC := LLC + PTRSIZE
                      END;
              LCP3 := LCP1; LCP1 := LCP2
            END;
          FPAR := LCP3
        END
          ELSE FPAR := NIL
END (*PARAMETERLIST*) ;

BEGIN (*PROCDECLARATION*)
  IF SEGDEC THEN (* SEGMENT DECLARATION *)
    BEGIN
      IF CODEINSEG THEN
        BEGIN ERROR(399); SEGINX:=0; CURBYTE:=0; END;
      WITH LLEXSTK DO
        BEGIN
          DOLDSEG:=SEG;
          SEG:=NEXTSEG;
          SOLDPROC:=NEXTPROC;
        END;
      NEXTPROC:=1;
      LSY:=SY;
      IF SY IN [PROCSY,FUNCSY] THEN INSYMBOL
      ELSE BEGIN ERROR(399); LSY:=PROCSY END;
      FSY:=LSY;
    END;
```

```
LLEXSTK.DLLC := LC; LC := LCAFTERMARKSTACK;
IF FSY = FUNCSY THEN LC := LC + REALSIZE;
LINEINFO := LC; DP := TRUE; EXTONLY := FALSE;
IF SY = IDENT THEN
  BEGIN
    IF USING OR INMODULE AND ININTERFACE THEN FORW := FALSE
    ELSE
      BEGIN SEARCHSECTION(DISPLAY[TOP].FNAME,LCP);
        IF LCP <> NIL THEN
          BEGIN
            IF LCP^.KLASS = PROC THEN
              FORW := LCP^.FORWDECL AND (FSY = PROCSY)
                      AND (LCP^.PFKIND = ACTUAL)
            ELSE
              IF LCP^.KLASS = FUNC THEN
                FORW := LCP^.FORWDECL AND (FSY = FUNCSY)
                        AND (LCP^.PFKIND = ACTUAL)
              ELSE FORW := FALSE;
            IF NOT FORW THEN ERROR(160)
          END
        ELSE FORW := FALSE
      END;
    IF NOT FORW THEN
      BEGIN
        IF FSY = PROCSY THEN
          IF INMODULE THEN NEW(LCP,PROC,DECLARED,ACTUAL,TRUE)
          ELSE NEW(LCP,PROC,DECLARED,ACTUAL,FALSE)
        ELSE
          IF INMODULE THEN NEW(LCP,FUNC,DECLARED,ACTUAL,TRUE)
          ELSE NEW(LCP,FUNC,DECLARED,ACTUAL,FALSE);
        WITH LCP^ DO
          BEGIN NAME := ID; IDTYPE := NIL; LOCALLC := LC;
            PFDECKIND := DECLARED; PFKIND := ACTUAL;
            INSCOPE := FALSE; PFLEV := LEVEL;
            PFNAME := NEXTPROC; PFSEG := SEG;
            IF USING THEN PROCTABLE[NEXTPROC] := 0;
            IF INMODULE THEN
              IF USING THEN IMPORTED := TRUE
              ELSE IMPORTED := FALSE;
            IF SEGDEC THEN
              BEGIN
                IF NEXTSEG > MAXSEG THEN ERROR(250);
                NEXTSEG := NEXTSEG+1;
                SEGTABLE[SEG].SEGNAME := ID
              END;
            IF NEXTPROC = MAXPROCNUM THEN ERROR(251)
            ELSE NEXTPROC := NEXTPROC + 1;
            IF FSY = PROCSY THEN KLASS := PROC
            ELSE KLASS := FUNC
          END;
        ENTERID(LCP)
      END
    ELSE
      BEGIN LCP1 := LCP^.NEXT;
        WHILE LCP1 <> NIL DO
          BEGIN
            WITH LCP1^ DO
              IF IDTYPE = NIL THEN
                EXTONLY := TRUE
              ELSE
```

```
                IF KLASS = FORMALVARS THEN
                  BEGIN
                    LCM := VADDR + PTRSIZE;
                    IF LCM > LC THEN LC := LCM
                  END
                ELSE
                  IF KLASS = ACTUALVARS THEN
                    BEGIN
                      LCM := VADDR + IDTYPE^.SIZE;
                      IF LCM > LC THEN LC := LCM
                    END;
              LCP1 := LCP1^.NEXT
            END;
          IF SEG <> LCP^.PFSEG THEN
            BEGIN
              SEG := LCP^.PFSEG; NEXTPROC := 2;
              IF NOT SEGDEC THEN ERROR(399)
            END
        END;
      INSYMBOL
    END
  ELSE
    BEGIN ERROR(2); LCP := UPRCPTR END;
  WITH LLEXSTK DO
    BEGIN DOLDLEV:=LEVEL;
      DOLDTOP:=TOP;
      POLDPROC:=CURPROC;
      DFPROCP:=LCP;
    END;
  CURPROC := LCP^.PFNAME;
  IF LEVEL < MAXLEVEL THEN LEVEL := LEVEL + 1 ELSE ERROR(251);
  IF TOP < DISPLIMIT THEN
    BEGIN TOP := TOP + 1;
      WITH DISPLAY[TOP] DO
        BEGIN
          IF FORW THEN FNAME := LCP^.NEXT
          ELSE FNAME := NIL;
          FLABEL := NIL; FFILE := NIL; OCCUR := BLCK
        END
    END
  ELSE ERROR(250);
  IF FSY = PROCSY THEN
    BEGIN PARAMETERLIST([SEMICOLON],LCP1,LCP);
      IF NOT FORW THEN LCP^.NEXT := LCP1
    END
  ELSE
    BEGIN PARAMETERLIST([SEMICOLON,COLON],LCP1,LCP);
      IF NOT FORW THEN LCP^.NEXT := LCP1;
      IF SY = COLON THEN
        BEGIN INSYMBOL;
          IF SY = IDENT THEN
            BEGIN IF FORW THEN ERROR(122);
              SEARCHID([TYPES],LCP1);
              LSP := LCP1^.IDTYPE;
              LCP^.IDTYPE := LSP;
              IF LSP <> NIL THEN
                IF NOT (LSP^.FORM IN [SCALAR,SUBRANGE,POINTER]) THEN
                  BEGIN ERROR(120); LCP^.IDTYPE := NIL END;
              INSYMBOL
            END
```

```
        ELSE BEGIN ERROR(2); SKIP(FSYS + [SEMICOLON]) END
      END
    ELSE
      IF NOT FORW THEN ERROR(123)
  END;
IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14);
LCP^.EXTURNAL := FALSE;
IF (SY = EXTERNLSY)
   OR ((USING) AND (LSEPPROC)) THEN
  BEGIN
    IF LEVEL <> 2 THEN
      ERROR(183) (*EXTERNAL PROCS MUST BE IN OUTERMOST BLOCK*);
    IF INMODULE THEN
      IF ININTERFACE AND NOT USING THEN
        ERROR(184); (*NO EXTERNAL DECL IN INTERFACE*)
    IF SEGDEC THEN ERROR(399);
    WITH LCP^ DO
      BEGIN EXTURNAL := TRUE; FORWDECL := FALSE;
        WRITELN(OUTPUT); WRITELN(OUTPUT,NAME,' [',MEMAVAIL:5,' words]');
        WRITE(OUTPUT,'<',SCREENDOTS:4,'>')
      END;
    PROCTABLE[CURPROC] := 0;
    DLINKERINFO := TRUE;
    IF SY = EXTERNLSY THEN
      BEGIN INSYMBOL;
        IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14);
        IF NOT (SY IN FSYS) THEN
          BEGIN ERROR(6); SKIP(FSYS) END
      END
  END
ELSE
  IF USING THEN
    BEGIN LCP^.FORWDECL := FALSE;
    END
  ELSE
    IF (SY = FORWARDSY) OR INMODULE AND ININTERFACE THEN
      BEGIN
        IF FORW THEN ERROR(161)
        ELSE LCP^.FORWDECL := TRUE;
        IF SY = FORWARDSY THEN
          BEGIN INSYMBOL;
            IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14)
          END;
        IF NOT (SY IN FSYS) THEN
          BEGIN ERROR(6); SKIP(FSYS) END
      END
    ELSE
      BEGIN
        IF EXTONLY THEN
          ERROR(7);
        NEWBLOCK:=TRUE;
        NOTDONE:=TRUE;
        WITH LLEXSTK DO
          BEGIN
            MARK(DMARKP);
            WITH LCP^ DO
              BEGIN FORWDECL := FALSE; INSCOPE := TRUE;
                EXTURNAL := FALSE END;
            BFSY:=SEMICOLON;
            ISSEGMENT:=SEGDEC;
```

```
                PREVLEXSTACKP:=TOS;
               END;
            NEW(TOS);
            TOS^:=LLEXSTK;
            EXIT(PROCDECLARATION);
          END;
    WITH LLEXSTK DO  (* FORWARD OR EXTERNAL DECLARATION, SO RESTORE STATE *)
      BEGIN
        LEVEL:=DOLDLEV;
        TOP:=DOLDTOP;
        LC:=DLLC;
        CURPROC:=POLDPROC;
        IF SEGDEC THEN
          BEGIN
            NEXTPROC:=SOLDPROC;
            SEG:=DOLDSEG;
          END;
      END;
   END; (* PROCDECLARATION *)


BEGIN (*DECLARATIONPART*)
  IF (NOSWAP) AND (STARTINGUP) THEN
    BEGIN
      STARTINGUP:=FALSE; (* ALL SEGMENTS ARE IN BY THIS TIME *)
      BLOCK(FSYS);
      EXIT(DECLARATIONPART);
    END;
  IF NOISY THEN
    UNITWRITE(3,DUMMYVAR[-1600],35); (*ADJUST DISPLAY OF STACK AND HEAP*)
  REPEAT
    NOTDONE:=FALSE;
    IF USERINFO.STUPID THEN
      IF NOT CODEINSEG THEN
        IF (LEVEL = 1) AND (NEXTSEG = 10) THEN
          IF NOT(INMODULE OR USING) THEN USESDECLARATION(TRUE);
          (*To get turtle graphics*)
    IF SY = USESSY THEN
      BEGIN INSYMBOL; USESDECLARATION(FALSE) END;
    IF SY = LABELSY THEN
      BEGIN
        IF INMODULE AND ININTERFACE THEN
          BEGIN ERROR(186); SKIP(FSYS - [LABELSY]) END
        ELSE INSYMBOL; LABELDECLARATION END;
    IF SY = CONSTSY THEN
      BEGIN INSYMBOL; CONSTDECLARATION END;
    IF SY = TYPESY THEN
      BEGIN INSYMBOL; TYPEDECLARATION END;
    IF SY = VARSY THEN
      BEGIN INSYMBOL; VARDECLARATION END;
    IF LEVEL = 1 THEN GLEV := TOP;
    IF SY IN [PROCSY,FUNCSY,PROGSY] THEN
      BEGIN
        IF INMODULE THEN
          IF ININTERFACE AND NOT USING THEN PUBLICPROCS := TRUE;
        REPEAT
          LSY := SY; INSYMBOL;
          IF LSY = PROGSY THEN
            IF INMODULE THEN
              BEGIN ERROR(185 (*SEG DEC NOT ALLOWED IN UNIT*));
```

```
                PROCDECLARATION(PROCSY,FALSE)
              END
            ELSE PROCDECLARATION(LSY,TRUE)
          ELSE PROCDECLARATION(LSY,FALSE);
        UNTIL NOT (SY IN [PROCSY,FUNCSY,PROGSY])
      END;
    IF (SY <> BEGINSY) THEN
     IF NOT ((USING OR INMODULE) AND (SY IN [IMPLESY,ENDSY]))
        AND NOT( SY IN [SEPARATSY,UNITSY]) THEN
       IF (NOT (INCLUDING OR NOTDONE))
          OR
          NOT(SY IN BLOCKBEGSYS) THEN
         BEGIN ERROR(18); SKIP(FSYS - [UNITSY,INTERSY]); END;
  UNTIL (SY IN (STATBEGSYS + [SEPARATSY,UNITSY,IMPLESY,ENDSY]));
  NEWBLOCK:=FALSE;
END (*DECLARATIONPART*) ;

(* $I BODYPART.A.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

SEGMENT PROCEDURE BODYPART(FSYS: SETOFSYS; FPROCP: CTP);

  PROCEDURE LINKERREF(KLASS: IDCLASS; ID,ADDR: INTEGER);
  BEGIN
    IF NREFS > REFSPERBLK THEN (*WRITE BUFFER*)
      BEGIN
        IF BLOCKWRITE(REFFILE,REFLIST^,1,REFBLK) <> 1 THEN ERROR(402);
        REFBLK := REFBLK + 1;
        NREFS := 1
      END;
    WITH REFLIST^[NREFS] DO
      BEGIN
        IF KLASS IN VARS THEN KEY := ID + 32
        ELSE (*PROC*) KEY := ID;
        OFFSET := SEGINX + ADDR
      END;
    NREFS := NREFS + 1
  END (*LINKERREF*) ;

  PROCEDURE GENLDC(IVAL: INTEGER);
  BEGIN
    IF (IVAL >= 0) AND (IVAL <= 127) THEN GENBYTE(IVAL)
    ELSE
      BEGIN GENBYTE(51(*LDC*)+148);
        MOVELEFT(IVAL,CODEP^[IC],2);
        IC := IC+2
      END
  END (*GENLDC*) ;

  PROCEDURE GENBIG(IVAL: INTEGER);
    VAR LOWORDER: CHAR;
  BEGIN
    IF IVAL <= 127 THEN GENBYTE(IVAL)
    ELSE
      BEGIN MOVELEFT(IVAL,CODEP^[IC],2); LOWORDER := CODEP^[IC];
        CODEP^[IC] := CHR(ORD(CODEP^[IC+1])+128);
        CODEP^[IC+1] := LOWORDER; IC := IC+2
      END
```

```
END (*GENBIG*) ;

PROCEDURE GEN0(FOP: OPRANGE);
  VAR I: INTEGER;
BEGIN
  GENBYTE(FOP+128);
  IF FOP = 38(*LCA*) THEN
    WITH GATTR.CVAL.VALP^ DO
      BEGIN GENBYTE(SLGTH);
        FOR I := 1 TO SLGTH DO GENBYTE(ORD(SVAL[I]))
      END
END (*GEN0*) ;

PROCEDURE GEN1(FOP: OPRANGE; FP2: INTEGER);
  LABEL 1;
  VAR I,J: INTEGER;
BEGIN
  GENBYTE(FOP+128);
  IF FOP = 51(*LDC*) THEN
    BEGIN
      IF FP2 = 2 THEN I := REALSIZE
      ELSE
        BEGIN I := 8;
          WHILE I > 0 DO
            IF GATTR.CVAL.VALP^.CSTVAL[I] <> 0 THEN GOTO 1
            ELSE I := I - 1;
    1:  END;
      GATTR.TYPTR^.SIZE := I;
      IF I > 1 THEN
        BEGIN GENBYTE(I);
          FOR J := I DOWNTO 1 DO GENWORD(GATTR.CVAL.VALP^.CSTVAL[J])
        END
      ELSE
        BEGIN IC := IC - 1;
          IF I = 1 THEN GENLDC(GATTR.CVAL.VALP^.CSTVAL[1])
        END
    END
  ELSE
    IF FOP IN [30(*CSP*),32(*ADJ*),45(*RNP*),
               46(*CIP*),60(*LDM*),61(*STM*),
               65(*RBP*),66(*CBP*),78(*CLP*),
               42(*SAS*),79(*CGP*)] THEN GENBYTE(FP2)
    ELSE
      IF INMODULE AND (FOP IN [37(*LAO*),39(*LDO*),43(*SRO*)]) THEN
        BEGIN LINKERREF(ACTUALVARS,FP2,IC); GENBYTE(128); GENBYTE(0) END
      ELSE
        IF ((FOP = 74(*LDL*)) OR (FOP = 39(*LDO*)))
            AND (FP2 <= 16) THEN
          BEGIN IC := IC-1;
            IF FOP = 39(*LDO*) THEN GENBYTE(231+FP2)
            ELSE GENBYTE(215+FP2)
          END
        ELSE
          IF (FOP = 35(*IND*)) AND (FP2 <= 7) THEN
            BEGIN IC := IC-1; GENBYTE(248+FP2) END
          ELSE
            GENBIG(FP2)
END (*GEN1*) ;

PROCEDURE GEN2(FOP: OPRANGE; FP1,FP2: INTEGER);
```

```
BEGIN
  IF (FOP = 64(*IXP*)) OR (FOP = 77(*CXP*)) THEN
    BEGIN GENBYTE(FOP+128); GENBYTE(FP1); GENBYTE(FP2);
    END
  ELSE
    IF FOP IN [47(*EQU*),48(*GEQ*),49(*GRT*),
               52(*LEQ*),53(*LES*),55(*NEQ*)] THEN
      IF FP1 = 0 THEN GEN0(FOP+20)
      ELSE
        BEGIN GEN1(FOP,FP1+FP1);
          IF FP1 > 4 THEN GENBIG(FP2)
        END
    ELSE
      BEGIN (*LDA,LOD,STR*)
        IF FP1 = 0 THEN GEN1(FOP+20,FP2)
        ELSE
          BEGIN
            GENBYTE(FOP+128); GENBYTE(FP1); GENBIG(FP2)
          END
      END;
END (*GEN2*) ;

PROCEDURE GENNR(EXTPROC: NONRESIDENT);

 PROCEDURE ASSIGN(EXTPROC: NONRESIDENT);
 BEGIN
   PROCTABLE[NEXTPROC] := 0;
   PFNUMOF[EXTPROC] := NEXTPROC; NEXTPROC := NEXTPROC + 1;
   IF NEXTPROC > MAXPROCNUM THEN ERROR(193);(*NOT ENOUGH ROOM FOR THIS*)
   CLINKERINFO := TRUE                                    (*OPERATION*)
 END (*ASSIGN*) ;

BEGIN (*GENNR*)
  IF PFNUMOF[EXTPROC] = 0 THEN ASSIGN(EXTPROC);
  IF SEPPROC THEN
    BEGIN
      GEN1(79(*CGP*),0); LINKERREF(PROC,-PFNUMOF[EXTPROC],IC-1)
    END
  ELSE
    GEN1(79(*CGP*),PFNUMOF[EXTPROC]);
END (*GENNR*) ;

PROCEDURE GENJMP(FOP: OPRANGE; FLBP: LBP);
  VAR DISP: INTEGER;
BEGIN
  WITH FLBP^ DO
    IF DEFINED THEN
      BEGIN
        GENBYTE(FOP+128);
        DISP := OCCURIC-IC-1;
        IF (DISP >= 0) AND (DISP <= 127) THEN GENBYTE(DISP)
        ELSE
          BEGIN
            IF JTABINX = 0 THEN
              BEGIN JTABINX := NEXTJTAB;
                IF NEXTJTAB = MAXJTAB THEN ERROR(253)
                ELSE NEXTJTAB := NEXTJTAB + 1;
                JTAB[JTABINX] := OCCURIC
              END;
            DISP := -JTABINX;
```

```
            GENBYTE(248-JTABINX-JTABINX)
          END;
      END
    ELSE
      BEGIN MOVELEFT(REFLIST,CODEP^[IC],2);
        IF FOP = 57(*UJP*) THEN DISP := IC + 4096
        ELSE DISP := IC;
        REFLIST := DISP; IC := IC+2
      END;
END (*GENJMP*) ;

PROCEDURE LOAD; FORWARD;

PROCEDURE GENFJP(FLBP: LBP);
BEGIN LOAD;
  IF GATTR.TYPTR <> BOOLPTR THEN ERROR(135);
  GENJMP(33(*FJP*),FLBP)
END (*GENFJP*) ;

PROCEDURE GENLABEL(VAR FLBP: LBP);
BEGIN NEW(FLBP);
  WITH FLBP^ DO
    BEGIN DEFINED := FALSE; REFLIST := MAXADDR END
END (*GENLABEL*) ;

PROCEDURE PUTLABEL(FLBP: LBP);
  VAR LREF: INTEGER; LOP: OPRANGE;
BEGIN
  WITH FLBP^ DO
    BEGIN LREF := REFLIST;
      DEFINED := TRUE; OCCURIC := IC; JTABINX := 0;
      WHILE LREF < MAXADDR DO
        BEGIN
          IF LREF >= 4096 THEN
            BEGIN LREF := LREF - 4096; LOP := 57(*UJP*) END
          ELSE LOP := 33(*FJP*);
          IC := LREF;
          MOVELEFT(CODEP^[IC],LREF,2);
          GENJMP(LOP,FLBP)
        END;
      IC := OCCURIC
    END
END (*PUTLABEL*) ;

PROCEDURE LOAD;
VAR J,M: INTEGER;
BEGIN
  WITH GATTR DO
    IF TYPTR <> NIL THEN
      BEGIN
        CASE KIND OF
          CST:   IF TYPTR^.FORM = LONGINT THEN
                   WITH GATTR.CVAL.VALP^ DO
                     BEGIN
                       M := 10000;
                       GENLDC(LONGVAL[1]); GENLDC(1);
                       FOR J := 2 TO LLENG DO
                         BEGIN
                           IF J = LLENG THEN M := TRUNC(PWROFTEN(LLAST));
                           GENLDC(M); GENLDC(1);
```

```
                        GENLDC(8(*DMP*)); GENNR(DECOPS);
                        GENLDC(LONGVAL[J]); GENLDC(1);
                        GENLDC(2(*DAD*)); GENNR(DECOPS)
                      END
                  END
                ELSE
                  IF (TYPTR^.FORM = SCALAR) AND (TYPTR <> REALPTR) THEN
                    GENLDC(CVAL.IVAL)
                  ELSE
                    IF TYPTR = NILPTR THEN GEN0(31(*LDCN*))
                    ELSE
                      IF TYPTR = REALPTR THEN GEN1(51(*LDC*),2)
                      ELSE GEN1(51(*LDC*),5);
        VARBL: CASE ACCESS OF
                  DRCT:   IF VLEVEL = 1 THEN GEN1(39(*LDO*),DPLMT)
                          ELSE GEN2(54(*LOD*),LEVEL-VLEVEL,DPLMT);
                  INDRCT: GEN1(35(*IND*),IDPLMT);
                  PACKD:  GEN0(58(*LDP*));
                  MULTI:  GEN1(60(*LDM*),TYPTR^.SIZE);
                  BYTE:   GEN0(62(*LDB*))
                END;
        EXPR:
      END;
      WITH TYPTR^ DO
        IF ((FORM = POWER) OR
           (FORM = LONGINT) AND (KIND <> CST))
           AND (KIND <> EXPR) THEN GENLDC(TYPTR^.SIZE);
      KIND := EXPR
    END
END (*LOAD*) ;

PROCEDURE STORE(VAR FATTR: ATTR);
BEGIN
  WITH FATTR DO
    IF TYPTR <> NIL THEN
      CASE ACCESS OF
        DRCT:   IF VLEVEL = 1 THEN GEN1(43(*SRO*),DPLMT)
                ELSE GEN2(56(*STR*),LEVEL-VLEVEL,DPLMT);
        INDRCT: IF IDPLMT <> 0 THEN ERROR(400)
                ELSE GEN0(26(*STO*));
        PACKD:  GEN0(59(*STP*));
        MULTI:  GEN1(61(*STM*),TYPTR^.SIZE);
        BYTE:   GEN0(63(*STB*))
      END
END (*STORE*) ;

PROCEDURE LOADADDRESS;
BEGIN
  WITH GATTR DO
    IF TYPTR <> NIL THEN
      BEGIN
        CASE KIND OF
          CST:   IF STRGTYPE(TYPTR) THEN GEN0(38(*LCA*))
                 ELSE ERROR(400);
          VARBL: CASE ACCESS OF
                    DRCT:   IF VLEVEL = 1 THEN GEN1(37(*LAO*),DPLMT)
                            ELSE GEN2(50(*LDA*),LEVEL-VLEVEL,DPLMT);
                    INDRCT: IF IDPLMT <> 0 THEN GEN1(34(*INC*),IDPLMT+IDPLMT);
                    PACKD:  ERROR(103)
                  END
```

```
        END;
        KIND := VARBL; ACCESS := INDRCT; IDPLMT := 0
      END
END (*LOADADDRESS*) ;

PROCEDURE EXPRESSION(FSYS: SETOFSYS); FORWARD;

PROCEDURE SELECTOR(FSYS: SETOFSYS; FCP: CTP);
  VAR LATTR: ATTR; LCP: CTP; LMIN,LMAX: INTEGER;
BEGIN
  WITH FCP^, GATTR DO
    BEGIN TYPTR := IDTYPE; KIND := VARBL;
      CASE KLASS OF
        ACTUALVARS:
          BEGIN VLEVEL := VLEV; DPLMT := VADDR; ACCESS := DRCT;
            IF INMODULE THEN
              IF TYPTR <> NIL THEN
                IF (VLEV = 1) AND (TYPTR^.FORM = RECORDS) THEN LOADADDRESS
          END;
        FORMALVARS:
          BEGIN
            IF VLEV = 1 THEN GEN1(39(*LDO*),VADDR)
            ELSE GEN2(54(*LOD*),LEVEL-VLEV,VADDR);
            ACCESS := INDRCT; IDPLMT := 0
          END;
        FIELD:
          WITH DISPLAY[DISX] DO
           BEGIN
            IF OCCUR = CREC THEN
              BEGIN ACCESS := DRCT; VLEVEL := CLEV;
                DPLMT := CDSPL + FLDADDR
              END
            ELSE
              BEGIN
                IF LEVEL = 1 THEN GEN1(39(*LDO*),VDSPL)
                ELSE GEN2(54(*LOD*),0,VDSPL);
                ACCESS := INDRCT; IDPLMT := FLDADDR
              END;
            IF FISPACKD THEN
              BEGIN LOADADDRESS;
                IF ((FLDRBIT = 0) OR (FLDRBIT = 8))
                      AND (FLDWIDTH = 8) THEN
                  BEGIN ACCESS := BYTE;
                    IF FLDRBIT = 8 THEN GEN1(34(*INC*),1)
                  END
                ELSE
                  BEGIN ACCESS := PACKD;
                    GENLDC(FLDWIDTH); GENLDC(FLDRBIT)
                  END
              END
           END;
        FUNC:
          IF PFDECKIND <> DECLARED THEN ERROR(150)
          ELSE
            IF NOT INSCOPE THEN ERROR(103)
            ELSE
                BEGIN ACCESS := DRCT; VLEVEL := PFLEV + 1;
                  DPLMT := LCAFTERMARKSTACK
                END
      END (*CASE*);
```

```
      IF TYPTR <> NIL THEN
        IF (TYPTR^.FORM <= POWER) AND
           (TYPTR^.SIZE > PTRSIZE) THEN
          BEGIN LOADADDRESS; ACCESS := MULTI END
    END (*WITH*);
  IF NOT (SY IN SELECTSYS + FSYS) THEN
    BEGIN ERROR(59); SKIP(SELECTSYS + FSYS) END;
  WHILE SY IN SELECTSYS DO
    BEGIN
(*[*) IF SY = LBRACK THEN
        BEGIN
          REPEAT LATTR := GATTR;
            WITH LATTR DO
              IF TYPTR <> NIL THEN
                IF TYPTR^.FORM <> ARRAYS THEN
                  BEGIN ERROR(138); TYPTR := NIL END;
            LOADADDRESS;
            INSYMBOL; EXPRESSION(FSYS + [COMMA,RBRACK]);
            LOAD;
            IF GATTR.TYPTR <> NIL THEN
              IF GATTR.TYPTR^.FORM <> SCALAR THEN ERROR(113);
            IF LATTR.TYPTR <> NIL THEN
              WITH LATTR.TYPTR^ DO
                BEGIN
                  IF COMPTYPES(INXTYPE,GATTR.TYPTR) THEN
                    BEGIN
                      IF (INXTYPE <> NIL) AND
                          NOT STRGTYPE(LATTR.TYPTR) THEN
                        BEGIN GETBOUNDS(INXTYPE,LMIN,LMAX);
                          IF RANGECHECK THEN
                            BEGIN GENLDC(LMIN); GENLDC(LMAX);
                              GEN0(8(*CHK*))
                            END;
                          IF LMIN <> 0 THEN
                            BEGIN GENLDC(ABS(LMIN));
                              IF LMIN > 0 THEN GEN0(21(*SBI*))
                              ELSE GEN0(2(*ADI*))
                            END
                        END
                    END
                  ELSE ERROR(139);
                  WITH GATTR DO
                    BEGIN TYPTR := AELTYPE; KIND := VARBL;
                      ACCESS := INDRCT; IDPLMT := 0;
                      IF TYPTR <> NIL THEN
                        IF AISPACKD THEN
                          IF ELWIDTH = 8 THEN
                            BEGIN ACCESS := BYTE;
                              IF STRGTYPE(LATTR.TYPTR) AND RANGECHECK THEN
                                GEN0(27(*IXS*))
                              ELSE GEN0(2(*ADI*))
                            END
                          ELSE
                            BEGIN ACCESS := PACKD;
                              GEN2(64(*IXP*),ELSPERWD,ELWIDTH)
                            END
                        ELSE
                          BEGIN GEN1(36(*IXA*),TYPTR^.SIZE);
                            IF (TYPTR^.FORM <= POWER) AND
                               (TYPTR^.SIZE > PTRSIZE) THEN
```

```
                              ACCESS := MULTI
                            END
                        END
                    END
            UNTIL SY <> COMMA;
            IF SY = RBRACK THEN INSYMBOL ELSE ERROR(12)
          END (*IF SY = LBRACK*)
        ELSE
(*.*)     IF SY = PERIOD THEN
            BEGIN
              WITH GATTR DO
                BEGIN
                  IF TYPTR <> NIL THEN
                    IF TYPTR^.FORM <> RECORDS THEN
                      BEGIN ERROR(140); TYPTR := NIL END;
                  INSYMBOL;
                  IF SY = IDENT THEN
                    BEGIN
                      IF TYPTR <> NIL THEN
                        BEGIN SEARCHSECTION(TYPTR^.FSTFLD,LCP);
                          IF LCP = NIL THEN
                            BEGIN ERROR(152); TYPTR := NIL END
                          ELSE
                            WITH LCP^ DO
                              BEGIN TYPTR := IDTYPE;
                                CASE ACCESS OF
                                  DRCT:   DPLMT := DPLMT + FLDADDR;
                                  INDRCT: IDPLMT := IDPLMT + FLDADDR;
                                  MULTI,BYTE,
                                  PACKD:  ERROR(400)
                                END (*CASE ACCESS*);
                                IF FISPACKD THEN
                                  BEGIN LOADADDRESS;
                                    IF ((FLDRBIT = 0) OR (FLDRBIT = 8))
                                        AND (FLDWIDTH = 8) THEN
                                      BEGIN ACCESS := BYTE;
                                        IF FLDRBIT = 8 THEN GEN1(34(*INC*),1)
                                      END
                                    ELSE
                                      BEGIN ACCESS := PACKD;
                                        GENLDC(FLDWIDTH); GENLDC(FLDRBIT)
                                      END
                                  END;
                                IF TYPTR <> NIL THEN
                                  IF (TYPTR^.FORM <= POWER) AND
                                     (TYPTR^.SIZE > PTRSIZE) THEN
                                    BEGIN LOADADDRESS; ACCESS := MULTI END
                              END
                        END;
                      INSYMBOL
                    END (*SY = IDENT*)
                  ELSE ERROR(2)
                END (*WITH GATTR*)
            END (*IF SY = PERIOD*)
          ELSE
(*^*)       BEGIN
              IF GATTR.TYPTR <> NIL THEN
                WITH GATTR,TYPTR^ DO
                  IF (FORM = POINTER) OR (FORM = FILES) THEN
                    BEGIN LOAD; KIND := VARBL;
```

```
                    ACCESS := INDRCT; IDPLMT := 0;
                    IF FORM = POINTER THEN TYPTR := ELTYPE
                    ELSE
                      BEGIN TYPTR := FILTYPE;
                        IF TYPTR = NIL THEN ERROR(399)
                      END;
                    IF TYPTR <> NIL THEN
                      IF (TYPTR^.FORM <= POWER) AND
                         (TYPTR^.SIZE > PTRSIZE) THEN
                              ACCESS := MULTI
                  END
                ELSE ERROR(141);
              INSYMBOL
            END;
        IF NOT (SY IN FSYS + SELECTSYS) THEN
          BEGIN ERROR(6); SKIP(FSYS + SELECTSYS) END
      END (*WHILE*)

  END (*SELECTOR*) ;

(* $I BODYPART.B.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

  PROCEDURE CALL(FSYS: SETOFSYS; FCP: CTP);
    VAR LKEY: 1..43; WASLPARENT: BOOLEAN;

    PROCEDURE VARIABLE(FSYS: SETOFSYS);
      VAR LCP: CTP;
    BEGIN
      IF SY = IDENT THEN
        BEGIN SEARCHID(VARS+[FIELD],LCP); INSYMBOL END
      ELSE BEGIN ERROR(2); LCP := UVARPTR END;
      SELECTOR(FSYS,LCP)
    END (*VARIABLE*) ;

    PROCEDURE STRGVAR(FSYS: SETOFSYS; MUSTBEVAR: BOOLEAN);
    BEGIN EXPRESSION(FSYS);
      WITH GATTR DO
        IF ((KIND = CST) AND (TYPTR = CHARPTR))
            OR STRGTYPE(TYPTR) THEN
          IF KIND = VARBL THEN LOADADDRESS
          ELSE
            BEGIN
              IF MUSTBEVAR THEN ERROR(154);
              IF KIND = CST THEN
                BEGIN
                  IF TYPTR = CHARPTR THEN
                    BEGIN
                      WITH SCONST^ DO
                        BEGIN CCLASS := STRG; SLGTH := 1;
                          SVAL[1] := CHR(CVAL.IVAL)
                        END;
                      CVAL.VALP := SCONST;
                      NEW(TYPTR,ARRAYS,TRUE,TRUE);
                      TYPTR^ := STRGPTR^;
                      TYPTR^.MAXLENG := 1
                    END;
                  LOADADDRESS
```

```
            END
        END
    ELSE
      BEGIN
        IF GATTR.TYPTR <> NIL THEN ERROR(125);
        GATTR.TYPTR := STRGPTR
      END
END (*STRGVAR*) ;

PROCEDURE ROUTINE(LKEY: INTEGER);

  PROCEDURE NEWSTMT;
    LABEL 1;
    VAR LSP,LSP1: STP; VARTS,LMIN,LMAX: INTEGER;
        LSIZE,LSZ: ADDRRANGE; LVAL: VALU;

  BEGIN VARIABLE(FSYS + [COMMA,RPARENT]); LOADADDRESS;
    LSP := NIL; VARTS := 0; LSIZE := 0;
    IF GATTR.TYPTR <> NIL THEN
      WITH GATTR.TYPTR^ DO
        IF FORM = POINTER THEN
          BEGIN
            IF ELTYPE <> NIL THEN
              WITH ELTYPE^ DO
                BEGIN LSIZE := SIZE;
                  IF FORM = RECORDS THEN LSP := RECVAR
                END
          END
        ELSE ERROR(116);
    WHILE SY = COMMA DO
      BEGIN INSYMBOL;
        CONSTANT(FSYS + [COMMA,RPARENT],LSP1,LVAL);
        VARTS := VARTS + 1;
        IF LSP = NIL THEN ERROR(158)
        ELSE
          IF LSP^.FORM <> TAGFLD THEN ERROR(162)
          ELSE
            IF LSP^.TAGFIELDP <> NIL THEN
              IF STRGTYPE(LSP1) OR (LSP1 = REALPTR) THEN ERROR(159)
              ELSE
                IF COMPTYPES(LSP^.TAGFIELDP^.IDTYPE,LSP1) THEN
                  BEGIN
                    LSP1 := LSP^.FSTVAR;
                    WHILE LSP1 <> NIL DO
                      WITH LSP1^ DO
                        IF VARVAL.IVAL = LVAL.IVAL THEN
                          BEGIN LSIZE := SIZE; LSP := SUBVAR;
                            GOTO 1
                          END
                        ELSE LSP1 := NXTVAR;
                    LSIZE := LSP^.SIZE; LSP := NIL;
                  END
                ELSE ERROR(116);
  1:  END (*WHILE*) ;
    GENLDC(LSIZE);
    GEN1(30(*CSP*),1(*NEW*))
  END (*NEWSTMT*) ;

  PROCEDURE MOVE;
  BEGIN VARIABLE(FSYS + [COMMA]); LOADADDRESS;
```

```
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  IF LKEY = 27 THEN
    BEGIN EXPRESSION(FSYS + [COMMA]); LOAD END
  ELSE
    BEGIN VARIABLE(FSYS + [COMMA]); LOADADDRESS END;
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  EXPRESSION(FSYS + [RPARENT]); LOAD;
  IF LKEY = 27 THEN GEN1(30(*CSP*),10(*FLC*))
  ELSE
    IF LKEY = 21 THEN GEN1(30(*CSP*),2(*MVL*))
    ELSE GEN1(30(*CSP*),3(*MVR*))
END (*MOVE*) ;

PROCEDURE EXIT;
  VAR LCP: CTP;
BEGIN
  IF SY = IDENT THEN
    BEGIN SEARCHID([PROC,FUNC],LCP); INSYMBOL END
  ELSE
    IF (SY = PROGSY) THEN
      BEGIN LCP := OUTERBLOCK; INSYMBOL END
    ELSE LCP := NIL;
  IF LCP <> NIL THEN
    IF LCP^.PFDECKIND = DECLARED THEN
      BEGIN GENLDC(LCP^.PFSEG); GENLDC(LCP^.PFNAME);
        IF INMODULE THEN
          BEGIN LINKERREF(PROC,LCP^.PFSEG,IC-2);
            IF SEPPROC THEN LINKERREF(PROC,-LCP^.PFNAME,IC-1);
          END
      END
    ELSE ERROR(125)
  ELSE ERROR(125);
  GEN1(30(*CSP*),4(*XIT*))
END (*EXIT*) ;

PROCEDURE UNITIO;
BEGIN
  IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  VARIABLE(FSYS + [COMMA]); LOADADDRESS;
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  EXPRESSION(FSYS + [COMMA,RPARENT]); LOAD;
  IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
  IF SY = COMMA THEN
    BEGIN INSYMBOL;
      IF SY = COMMA THEN GENLDC(0)
      ELSE
        BEGIN
          EXPRESSION(FSYS + [COMMA,RPARENT]); LOAD;
          IF GATTR.TYPTR <> INTPTR THEN ERROR(125)
        END
    END
  ELSE GENLDC(0);
  IF SY = COMMA THEN
    BEGIN INSYMBOL;
      EXPRESSION(FSYS + [RPARENT]); LOAD;
      IF GATTR.TYPTR <> INTPTR THEN ERROR(125)
    END
  ELSE GENLDC(0);
  IF LKEY = 13 THEN GEN1(30(*CSP*),5(*URD*))
```

```
  ELSE GEN1(30(*CSP*),6(*UWT*))
END (*UNITIO*);

PROCEDURE CONCAT;
  VAR LLC: ADDRRANGE; TEMPLGTH: INTEGER;
BEGIN TEMPLGTH := 0;
  LLC := LC; LC := LC + (STRGLGTH DIV CHRSPERWD) + 1;
  GENLDC(0); GEN2(56(*STR*),0,LLC);
  GEN2(50(*LDA*),0,LLC);
  REPEAT
    STRGVAR(FSYS + [COMMA,RPARENT],FALSE);
    TEMPLGTH := TEMPLGTH + GATTR.TYPTR^.MAXLENG;
    IF TEMPLGTH < STRGLGTH THEN GENLDC(TEMPLGTH)
    ELSE GENLDC(STRGLGTH);
    GEN2(77(*CXP*),0(*SYS*),23(*SCONCAT*));
    GEN2(50(*LDA*),0,LLC);
    TEST := SY <> COMMA;
    IF NOT TEST THEN INSYMBOL
  UNTIL TEST;
  IF TEMPLGTH < STRGLGTH THEN
    LC := LLC + (TEMPLGTH DIV CHRSPERWD) + 1
  ELSE TEMPLGTH := STRGLGTH;
  IF LC > LCMAX THEN LCMAX := LC;
  LC := LLC;
  WITH GATTR DO
    BEGIN NEW(TYPTR,ARRAYS,TRUE,TRUE);
      TYPTR^ := STRGPTR^;
      TYPTR^.MAXLENG := TEMPLGTH
    END
END (*CONCAT*) ;

PROCEDURE COPYDELETE;
  VAR LLC: ADDRRANGE; LSP: STP;
BEGIN
  IF LKEY = 19 THEN
    BEGIN LLC := LC;
      LC := LC + (STRGLGTH DIV CHRSPERWD) + 1;
    END;
  IF LKEY <> 43 THEN
    BEGIN
      STRGVAR(FSYS + [COMMA], LKEY = 18);
      IF LKEY = 19 THEN
        BEGIN LSP := GATTR.TYPTR;
          GEN2(50(*LDA*),0,LLC)
        END;
      IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
    END;
  EXPRESSION(FSYS + [COMMA]); LOAD;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  EXPRESSION(FSYS + [RPARENT]); LOAD;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
  IF LKEY = 19 THEN
    BEGIN
      GEN2(77(*CXP*),0(*SYS*),25(*SCOPY*));
      GEN2(50(*LDA*),0,LLC);
      IF LSP^.MAXLENG < STRGLGTH THEN
        LC := LLC + (LSP^.MAXLENG DIV CHRSPERWD) + 1;
```

```
      IF LC > LCMAX THEN LCMAX := LC;
      LC := LLC; GATTR.TYPTR := LSP
    END
  ELSE
    IF LKEY = 43 THEN
      GEN2(77(*CXP*),0(*SYS*),29(*GOTOXY*))
    ELSE GEN2(77(*CXP*),0(*SYS*),26(*SDELETE*))
END (*COPYDELETE*) ;

PROCEDURE STR;
BEGIN
  WITH GATTR DO
    BEGIN
      IF COMPTYPES(LONGINTPTR,TYPTR) THEN
      ELSE IF TYPTR = INTPTR THEN
             BEGIN GENLDC(1); TYPTR := LONGINTPTR END
           ELSE ERROR(125);
      IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
      STRGVAR(FSYS + [RPARENT], TRUE);
      IF STRGTYPE(TYPTR) THEN
        BEGIN GENLDC(TYPTR^.MAXLENG); GENLDC(12(*DSTR*));
          GENNR(DECOPS)
        END
      ELSE ERROR(116);
    END
END (*STR*);

PROCEDURE CLOSE;
BEGIN
  VARIABLE(FSYS + [COMMA,RPARENT]); LOADADDRESS;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR^.FORM <> FILES THEN ERROR(125);
  IF SY = COMMA THEN
    BEGIN INSYMBOL;
      IF SY = IDENT THEN
       BEGIN
        IF ID = 'NORMAL  ' THEN GENLDC(0)
        ELSE
          IF ID = 'LOCK    ' THEN GENLDC(1)
          ELSE
            IF ID = 'PURGE   ' THEN GENLDC(2)
            ELSE
              IF ID = 'CRUNCH  ' THEN GENLDC(3)
              ELSE ERROR(2);
        INSYMBOL
       END
      ELSE ERROR(2)
    END
  ELSE GENLDC(0);
  GEN2(77(*CXP*),0(*SYS*),6(*FCLOSE*));
  IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*))
END (*CLOSE*) ;

PROCEDURE GETPUTETC;
BEGIN
  VARIABLE(FSYS + [COMMA,RPARENT]); LOADADDRESS;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR^.FORM <> FILES THEN ERROR(125)
    ELSE
      IF GATTR.TYPTR^.FILTYPE = NIL THEN ERROR(399);
```

```
  CASE LKEY OF
    32:  BEGIN
           IF SY = COMMA THEN
             BEGIN
               INSYMBOL; EXPRESSION(FSYS + [RPARENT]); LOAD;
               IF GATTR.TYPTR <> INTPTR THEN ERROR(125)
             END
           ELSE ERROR(125);
           GENNR(SEEK)
         END;
    34:  GEN2(77(*CXP*),0(*SYS*),7(*FGET*));
    35:  GEN2(77(*CXP*),0(*SYS*),8(*FPUT*));
    40:  BEGIN
           IF GATTR.TYPTR <> NIL THEN
             IF GATTR.TYPTR^.FILTYPE <> CHARPTR THEN ERROR(399);
           GENLDC(12); GENLDC(0);
           GEN2(77(*CXP*),0(*SYS*),17(*WRC*))
         END
  END (*CASE*) ;
  IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*))
END (*GETPUTETC*) ;

PROCEDURE SCAN;
BEGIN
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  IF SY = RELOP THEN
    BEGIN
      IF OP = EQOP THEN GENLDC(0)
      ELSE
        IF OP = NEOP THEN GENLDC(1)
        ELSE ERROR(125);
      INSYMBOL
    END
  ELSE ERROR(125);
  EXPRESSION(FSYS + [COMMA]); LOAD;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR <> CHARPTR THEN ERROR(125);
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  VARIABLE(FSYS + [COMMA,RPARENT]); LOADADDRESS;
  IF SY = COMMA THEN
    BEGIN INSYMBOL;
      EXPRESSION(FSYS + [RPARENT]); LOAD
    END
  ELSE GENLDC(0);
  GEN1(30(*CSP*),11(*SCN*));
  GATTR.TYPTR := INTPTR
END (*SCAN*) ;

PROCEDURE BLOCKIO;
BEGIN
  VARIABLE(FSYS + [COMMA]); LOADADDRESS;
  IF GATTR.TYPTR <> NIL THEN
    IF GATTR.TYPTR^.FORM <> FILES THEN ERROR(125)
    ELSE
      IF GATTR.TYPTR^.FILTYPE <> NIL THEN ERROR(399);
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
  VARIABLE(FSYS + [COMMA]); LOADADDRESS;
  IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
```

```
        EXPRESSION(FSYS + [COMMA,RPARENT]); LOAD;
        IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
        IF SY = COMMA THEN
          BEGIN INSYMBOL;
            EXPRESSION(FSYS + [RPARENT]); LOAD;
            IF GATTR.TYPTR <> INTPTR THEN ERROR(125)
          END
        ELSE GENLDC(-1);
        IF LKEY = 37 THEN GENLDC(1) ELSE GENLDC(0);
        GENLDC(0); GENLDC(0);
        GEN2(77(*CXP*),0(*SYS*),28(*BLOCKIO*));
        IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*));
        GATTR.TYPTR := INTPTR
      END (*BLOCKIO*) ;

      PROCEDURE SIZEOF;
        VAR LCP: CTP;
      BEGIN
        IF SY = IDENT THEN
          BEGIN SEARCHID(VARS + [TYPES,FIELD],LCP); INSYMBOL;
            IF LCP^.IDTYPE <> NIL THEN
              GENLDC(LCP^.IDTYPE^.SIZE*CHRSPERWD)
          END;
        GATTR.TYPTR := INTPTR
      END (*SIZEOF*) ;

  BEGIN (*ROUTINE*)
    CASE LKEY OF
      12:      NEWSTMT;
      13,14:   UNITIO;
      15:      CONCAT;
      18,19,43:COPYDELETE;
      21,22,27:MOVE;
      23:      EXIT;
      31:      CLOSE;
      32,34,
      35,40:   GETPUTETC;
      36:      SCAN;
      37,38:   BLOCKIO;
      41:      SIZEOF;
      42:      STR
    END (*CASES*)
  END (*ROUTINE*) ;

(* $I BODYPART.C.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

    PROCEDURE LOADIDADDR(FCP: CTP);
    BEGIN
        WITH FCP^ DO
          IF KLASS = ACTUALVARS THEN
            IF VLEV = 1 THEN GEN1(37(*LAO*),VADDR)
            ELSE GEN2(50(*LDA*),LEVEL-VLEV,VADDR)
          ELSE  (*FORMALVARS*)
            IF VLEV = 1 THEN GEN1(39(*LDO*),VADDR)
            ELSE GEN2(54(*LOD*),LEVEL-VLEV,VADDR)
    END (*LOADIDADDR*) ;
```

```
PROCEDURE READ;
  VAR FILEPTR,LCP: CTP;
BEGIN FILEPTR := INPUTPTR;
  IF (SY = IDENT) AND WASLPARENT THEN
    BEGIN SEARCHID(VARS+[FIELD],LCP);
      IF LCP^.IDTYPE <> NIL THEN
        IF LCP^.IDTYPE^.FORM = FILES THEN
          IF LCP^.IDTYPE^.FILTYPE = CHARPTR THEN
            BEGIN INSYMBOL; FILEPTR := LCP;
              IF NOT (SY IN [COMMA,RPARENT]) THEN ERROR(20);
              IF SY = COMMA THEN INSYMBOL
            END
    END
  ELSE
    IF WASLPARENT THEN ERROR(2);
  IF WASLPARENT AND (SY <> RPARENT) THEN
    BEGIN
      REPEAT LOADIDADDR(FILEPTR);
        VARIABLE(FSYS + [COMMA,RPARENT]);
        IF GATTR.ACCESS = BYTE THEN ERROR(103);
        LOADADDRESS;
        IF GATTR.TYPTR <> NIL THEN
          IF COMPTYPES(INTPTR,GATTR.TYPTR) THEN
            GEN2(77(*CXP*),0(*SYS*),12(*FRDI*))
          ELSE
            IF COMPTYPES(REALPTR,GATTR.TYPTR) THEN
              GENNR(FREADREAL)
            ELSE
              IF COMPTYPES(LONGINTPTR,GATTR.TYPTR) THEN
                BEGIN GENLDC(GATTR.TYPTR^.SIZE);
                  GENNR(FREADDEC)
                END
              ELSE
                IF COMPTYPES(CHARPTR,GATTR.TYPTR) THEN
                  GEN2(77(*CXP*),0(*SYS*),16(*FRDC*))
                ELSE
                  IF STRGTYPE(GATTR.TYPTR) THEN
                    BEGIN GENLDC(GATTR.TYPTR^.MAXLENG);
                      GEN2(77(*CXP*),0(*SYS*),18(*FRDS*))
                    END
                  ELSE ERROR(125);
        IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*));
        TEST := SY <> COMMA;
        IF NOT TEST THEN INSYMBOL
      UNTIL TEST
    END;
  IF LKEY = 2 THEN
    BEGIN LOADIDADDR(FILEPTR);
      GEN2(77(*CXP*),0(*SYS*),21(*FRLN*));
      IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*))
    END
END (*READ*) ;

PROCEDURE WRITE;
  VAR LSP: STP; DEFAULT: BOOLEAN;
      FILEPTR,LCP: CTP; LEN,LMIN,LMAX: INTEGER;
BEGIN FILEPTR := OUTPUTPTR;
  IF (SY = IDENT) AND WASLPARENT THEN
    BEGIN SEARCHID(VARS + [FIELD,KONST,FUNC],LCP);
      IF LCP^.IDTYPE <> NIL THEN
```

```
        IF LCP^.IDTYPE^.FORM = FILES THEN
          IF LCP^.IDTYPE^.FILTYPE = CHARPTR THEN
            BEGIN INSYMBOL; FILEPTR := LCP;
              IF NOT (SY IN [COMMA,RPARENT]) THEN ERROR(20);
              IF SY = COMMA THEN INSYMBOL
            END
    END;
  IF WASLPARENT AND (SY <> RPARENT) THEN
    BEGIN
      REPEAT LOADIDADDR(FILEPTR);
        EXPRESSION(FSYS + [COMMA,COLON,RPARENT]);
        LSP := GATTR.TYPTR;
        IF LSP <> NIL THEN
          WITH LSP^ DO
            BEGIN
              IF FORM > LONGINT THEN LOADADDRESS
              ELSE
                BEGIN LOAD;
                  IF FORM = LONGINT THEN
                    BEGIN GENLDC(DECSIZE(MAXDEC)); GENLDC(0(*DAJ*));
                      GENNR(DECOPS)
                    END
                END
            END;
        IF SY = COLON THEN
          BEGIN INSYMBOL;
            EXPRESSION(FSYS + [COMMA,COLON,RPARENT]);
            IF GATTR.TYPTR <> NIL THEN
              IF GATTR.TYPTR <> INTPTR THEN ERROR(20);
            LOAD; DEFAULT := FALSE
          END
        ELSE DEFAULT := TRUE;
        IF LSP = INTPTR THEN
          BEGIN IF DEFAULT THEN GENLDC(0);
            GEN2(77(*CXP*),0(*SYS*),13(*FWRI*))
          END
        ELSE
          IF LSP = REALPTR THEN
            BEGIN IF DEFAULT THEN GENLDC(0);
              IF SY = COLON THEN
                BEGIN INSYMBOL;
                  EXPRESSION(FSYS + [COMMA,RPARENT]); LOAD;
                  IF GATTR.TYPTR <> NIL THEN
                    IF GATTR.TYPTR <> INTPTR THEN ERROR(125)
                END
              ELSE GENLDC(0);
              GENNR(FWRITEREAL)
            END
          ELSE
            IF COMPTYPES(LSP,LONGINTPTR) THEN
              BEGIN IF DEFAULT THEN GENLDC(0); GENNR(FWRITEDEC) END
            ELSE
              IF LSP = CHARPTR THEN
                BEGIN IF DEFAULT THEN GENLDC(0);
                  GEN2(77(*CXP*),0(*SYS*),17(*FWRC*))
                END
              ELSE
                IF STRGTYPE(LSP) THEN
                  BEGIN IF DEFAULT THEN GENLDC(0);
                    GEN2(77(*CXP*),0(*SYS*),19(*FWRS*))
```

```
                  END
                ELSE
                  IF PAOFCHAR(LSP) THEN
                    BEGIN LMAX := 0;
                      IF LSP^.INXTYPE <> NIL THEN
                        BEGIN GETBOUNDS(LSP^.INXTYPE,LMIN,LMAX);
                           LMAX := LMAX - LMIN + 1
                        END;
                      IF DEFAULT THEN GENLDC(LMAX);
                      GENLDC(LMAX);
                      GEN2(77(*CXP*),0(*SYS*),20(*FWRB*))
                    END
                  ELSE ERROR(125);
          IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*));
          TEST := SY <> COMMA;
          IF NOT TEST THEN INSYMBOL
        UNTIL TEST;
      END;
    IF LKEY = 4 THEN (*WRITELN*)
      BEGIN LOADIDADDR(FILEPTR);
        GEN2(77(*CXP*),0(*SYS*),22(*FWLN*));
        IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*))
      END
  END (*WRITE*) ;

  PROCEDURE CALLNONSPECIAL;
    LABEL 1;
    VAR NXT,LCP: CTP; LSP: STP; LB: BOOLEAN;
        LMIN,LMAX: INTEGER;
  BEGIN
    WITH FCP^ DO
      BEGIN NXT := NEXT;
        IF PFDECKIND = DECLARED THEN
          IF PFKIND <> ACTUAL THEN ERROR(400)
      END;
    IF SY = LPARENT THEN
      BEGIN
        REPEAT
          IF NXT = NIL THEN ERROR(126);
          INSYMBOL;
          EXPRESSION(FSYS + [COMMA,RPARENT]);
          IF (GATTR.TYPTR <> NIL) AND (NXT <> NIL) THEN
            BEGIN LSP := NXT^.IDTYPE;
              IF (NXT^.KLASS = FORMALVARS) OR (LSP <> NIL) THEN
                BEGIN
                  IF NXT^.KLASS = ACTUALVARS THEN
                    IF GATTR.TYPTR^.FORM <= POWER THEN
                      BEGIN LB := (GATTR.TYPTR = CHARPTR)
                                  AND (GATTR.KIND = CST);
                        LOAD;
                        IF LSP^.FORM = POWER THEN
                          GEN1(32(*ADJ*),LSP^.SIZE)
                        ELSE
                        IF LSP^.FORM = LONGINT THEN
                          BEGIN
                            IF GATTR.TYPTR = INTPTR THEN
                              BEGIN GENLDC(INTSIZE);
                                GATTR.TYPTR := LONGINTPTR
                              END;
                            GENLDC(LSP^.SIZE);
```

```
                        GENLDC(0(*DAJ*));
                        GENNR(DECOPS)
                      END
                    ELSE
                    IF (LSP^.FORM = SUBRANGE)
                          AND RANGECHECK THEN
                      BEGIN GENLDC(LSP^.MIN.IVAL);
                        GENLDC(LSP^.MAX.IVAL);
                        GEN0(8(*CHK*))
                      END
                    ELSE
                    IF (GATTR.TYPTR = INTPTR) AND
                          COMPTYPES(LSP,REALPTR) THEN
                      BEGIN GEN0(10(*FLT*));
                        GATTR.TYPTR := REALPTR
                      END
                    ELSE
                    IF LB AND STRGTYPE(LSP) THEN
                      GATTR.TYPTR := STRGPTR
                  END
                ELSE (*FORM > POWER*)
                  BEGIN LB := STRGTYPE(GATTR.TYPTR)
                              AND (GATTR.KIND = CST);
                    LOADADDRESS;
                    IF LB AND PAOFCHAR(LSP) THEN
                      IF NOT LSP^.AISSTRNG THEN
                        BEGIN GEN0(80(*S1P*));
                          IF LSP^.INXTYPE <> NIL THEN
                            BEGIN
                              GETBOUNDS(LSP^.INXTYPE,LMIN,LMAX);
                              IF LMAX-LMIN+1 <>
                                  GATTR.TYPTR^.MAXLENG THEN ERROR(142);
                            END;
                          GATTR.TYPTR := LSP
                        END
                  END
              ELSE (*KLASS = FORMALVARS*)
                IF GATTR.KIND = VARBL THEN
                  BEGIN
                    IF GATTR.ACCESS = BYTE THEN ERROR(103);
                    LOADADDRESS;
                    IF LSP <> NIL THEN
                      IF LSP^.FORM IN [POWER,LONGINT] THEN
                        IF GATTR.TYPTR^.SIZE <>
                            LSP^.SIZE THEN ERROR(142)
                  END
                ELSE ERROR(154);
              IF NOT COMPTYPES(LSP,GATTR.TYPTR) THEN ERROR(142)
            END
        END;
      IF NXT <> NIL THEN NXT := NXT^.NEXT
    UNTIL SY <> COMMA;
    IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
  END (*LPARENT*) ;
IF NXT <> NIL THEN ERROR(126);
WITH FCP^ DO
  IF PFDECKIND = DECLARED THEN
    BEGIN
      IF KLASS = FUNC THEN
        BEGIN GENLDC(0); GENLDC(0) END;
```

```
          IF INMODULE THEN
            IF SEPPROC THEN
              IF (PFSEG = SEG) AND (PFLEV = 1) THEN
                BEGIN GEN1(79(*CGP*),0); LINKERREF(PROC,-PFNAME,IC-1) END
              ELSE
                IF PFLEV = 0 THEN GEN2(77(*CXP*),PFSEG,PFNAME)
                ELSE ERROR(405) (*CALL NOT ALLOWED IN SEP PROC*)
            ELSE
              IF IMPORTED THEN
                BEGIN GEN2(77(*CXP*),0,PFNAME); LINKERREF(PROC,PFSEG,IC-2) END
              ELSE GOTO 1
          ELSE
     1:     IF PFSEG <> SEG THEN
              GEN2(77(*CXP*),PFSEG,PFNAME)
            ELSE
              IF PFLEV = 0 THEN GEN1(66(*CBP*),PFNAME)
              ELSE
                IF PFLEV = LEVEL THEN GEN1(78(*CLP*),PFNAME)
                ELSE
                  IF PFLEV = 1 THEN GEN1(79(*CGP*),PFNAME)
                  ELSE GEN1(46(*CIP*),PFNAME)
        END
       ELSE
        IF CSPNUM = 23 THEN GEN1(30,40)  (* TEMP I.5 TRANSLATION --
                                            MEM WILL BE CSP 23 IN II.0  *)
        ELSE
          IF (CSPNUM <> 21) AND (CSPNUM <> 22) THEN
            GEN1(30(*CSP*),CSPNUM);
    GATTR.TYPTR := FCP^.IDTYPE
  END (*CALLNONSPECIAL*) ;

BEGIN (*CALL*)
  IF FCP^.PFDECKIND = SPECIAL THEN
    BEGIN WASLPARENT := TRUE; LKEY := FCP^.KEY;
      IF SY = LPARENT THEN INSYMBOL
      ELSE
        IF LKEY IN [2,4,5,6] THEN WASLPARENT := FALSE
        ELSE ERROR(9);
      IF LKEY IN [7,8,9,10,11,13,14,25,36,39,42] THEN
        BEGIN EXPRESSION(FSYS + [COMMA,RPARENT]); LOAD END;
      IF LKEY IN [12,13,14,15,18,19,21,22,23,27,31,32,34,35,36,37,38,
                  40,41,42,43] THEN ROUTINE(LKEY)
      ELSE
        CASE LKEY OF
           1,2: READ;
           3,4: WRITE;
           5,6: BEGIN (*EOF & EOLN*)
                  IF WASLPARENT THEN
                    BEGIN VARIABLE(FSYS + [RPARENT]); LOADADDRESS;
                      IF GATTR.TYPTR <> NIL THEN
                        IF GATTR.TYPTR^.FORM <> FILES THEN ERROR(125)
                        ELSE
                          IF (GATTR.TYPTR^.FILTYPE <> CHARPTR) AND
                              (LKEY = 6) THEN ERROR(399)
                    END
                  ELSE
                    LOADIDADDR(INPUTPTR);
                  GENLDC(0); GENLDC(0);
                  IF LKEY = 5 THEN GEN2(77(*CXP*),0(*SYS*),10(*FEOF*))
                  ELSE GEN2(77(*CXP*),0(*SYS*),11(*FEOLN*));
```

```
        GATTR.TYPTR := BOOLPTR
      END (*EOF*) ;
    7,8: BEGIN GENLDC(1); (*PREDSUCC*)
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR^.FORM = SCALAR THEN
            IF LKEY = 8 THEN GEN0(2(*ADI*))
            ELSE GEN0(21(*SBI*))
          ELSE ERROR(115)
      END (*PREDSUCC*) ;
     9: BEGIN (*ORD*)
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR^.FORM >= POWER THEN ERROR(125);
        GATTR.TYPTR := INTPTR
      END (*ORD*) ;
    10: BEGIN (*SQR*)
        IF GATTR.TYPTR <> NIL THEN
        IF GATTR.TYPTR = INTPTR THEN GEN0(24(*SQI*))
        ELSE
          IF GATTR.TYPTR = REALPTR THEN GEN0(25(*SQR*))
          ELSE BEGIN ERROR(125); GATTR.TYPTR := INTPTR END
      END (*SQR*) ;
    11: BEGIN (*ABS*)
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR = INTPTR THEN GEN0(0(*ABI*))
          ELSE
            IF GATTR.TYPTR = REALPTR THEN GEN0(1(*ABR*))
            ELSE BEGIN ERROR(125); GATTR.TYPTR := INTPTR END
      END (*ABS*) ;
    16: BEGIN (*LENGTH*)
        STRGVAR(FSYS + [RPARENT],FALSE);
        GEN0(62(*LDB*)); GATTR.TYPTR := INTPTR
      END (*LENGTH*) ;
    17: BEGIN (*INSERT*)
        STRGVAR(FSYS + [COMMA],FALSE);
        IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
        STRGVAR(FSYS + [COMMA],TRUE);
        GENLDC(GATTR.TYPTR^.MAXLENG);
        IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
        EXPRESSION(FSYS + [RPARENT]); LOAD;
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
        GEN2(77(*CXP*),0(*SYS*),24(*SINSERT*))
      END (*INSERT*) ;
    20: BEGIN (*POS*)
        STRGVAR(FSYS + [COMMA],FALSE);
        IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
        STRGVAR(FSYS + [RPARENT],FALSE);
        GENLDC(0); GENLDC(0);
        GEN2(77(*CXP*),0(*SYS*),27(*SPOS*));
        GATTR.TYPTR := INTPTR
      END (*POS*) ;
    24: BEGIN (*IDSEARCH*)
        VARIABLE(FSYS + [COMMA]); LOADADDRESS;
        IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
        VARIABLE(FSYS + [RPARENT]); LOADADDRESS;
        GEN1(30(*CSP*),7(*IDS*))
      END (*IDSEARCH*) ;
    25: BEGIN (*TREESEARCH*)
        IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
        VARIABLE(FSYS + [COMMA]); LOADADDRESS;
```

```
                    IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
                    VARIABLE(FSYS + [RPARENT]); LOADADDRESS;
                    GATTR.TYPTR := INTPTR;
                    GEN1(30(*CSP*),8(*TRS*))
                  END (*TREESEARCH*) ;
              26: BEGIN (*TIME*)
                    VARIABLE(FSYS + [COMMA]); LOADADDRESS;
                    IF GATTR.TYPTR <> NIL THEN
                      IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
                    IF SY = COMMA THEN INSYMBOL ELSE ERROR(20);
                    VARIABLE(FSYS + [RPARENT]); LOADADDRESS;
                    IF GATTR.TYPTR <> NIL THEN
                      IF GATTR.TYPTR <> INTPTR THEN ERROR(125);
                    GEN1(30(*CSP*),9(*TIM*))
                  END (*TIME*) ;
    33,28,29,30: BEGIN (*OPEN,RESET,REWRITE*)
                    VARIABLE(FSYS + [COMMA,RPARENT]); LOADADDRESS;
                    IF GATTR.TYPTR <> NIL THEN
                      IF GATTR.TYPTR^.FORM <> FILES THEN ERROR(125);
                    IF SY <> COMMA THEN
                      IF LKEY = 33 THEN
                        GEN2(77(*CXP*),0(*SYS*),4(*FRESET*))
                      ELSE ERROR(20)
                    ELSE
                      BEGIN INSYMBOL;
                        STRGVAR(FSYS + [RPARENT],FALSE);
                        IF (LKEY = 28) OR (LKEY = 30) THEN
                          GENLDC(0)
                        ELSE GENLDC(1);
                        GENLDC(0); GEN2(77(*CXP*),0(*SYS*),5(*FOPEN*))
                      END;
                    IF IOCHECK THEN GEN1(30(*CSP*),0(*IOC*))
                  END (*OPEN*) ;
              39: BEGIN (*TRUNC*)
                    IF GATTR.TYPTR = INTPTR THEN
                      BEGIN GEN0(10(*FLT*));
                        GATTR.TYPTR := REALPTR
                      END;
                    IF GATTR.TYPTR <> NIL THEN
                      IF GATTR.TYPTR = REALPTR THEN
                        GEN1(30(*CSP*),23(*TRUNC*)) (*** TEMPORARY --
                                          TRUNC WILL BE CSP 14 IN II.0 ***)
                      ELSE
                        IF GATTR.TYPTR^.FORM = LONGINT THEN
                          BEGIN
                            GENLDC(INTSIZE); GENLDC(0 (*DAJ*));
                            GENNR(DECOPS)
                          END
                        ELSE ERROR(125);
                    GATTR.TYPTR := INTPTR
                  END
          END (*SPECIAL CASES*) ;
        IF WASLPARENT THEN
          IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
      END (*SPECIAL PROCEDURES AND FUNCTIONS*)
    ELSE CALLNONSPECIAL
  END (*CALL*) ;

(* $I BODYPART.D.TEXT*)
```

```
(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

  PROCEDURE EXPRESSION(*FSYS: SETOFSYS*);
    LABEL 1;    (* STRING COMPARE KLUDGE *)
    VAR LATTR: ATTR; LOP: OPERATOR; TYPIND: INTEGER;
        LSIZE: ADDRRANGE; LSTRING,GSTRING: BOOLEAN;
        LMIN,LMAX: INTEGER;

    PROCEDURE FLOATIT(VAR FSP: STP; FORCEFLOAT: BOOLEAN);
    BEGIN
      IF (GATTR.TYPTR = REALPTR) OR (FSP = REALPTR) OR FORCEFLOAT THEN
        BEGIN
          IF GATTR.TYPTR = INTPTR THEN
            BEGIN GEN0(10(*FLT*)); GATTR.TYPTR := REALPTR END;
          IF FSP = INTPTR THEN
            BEGIN GEN0(9(*FLO*)); FSP := REALPTR END
        END
    END (*FLOATIT*) ;

    PROCEDURE STRETCHIT(VAR FSP: STP);

    BEGIN
      IF (FSP^.FORM = LONGINT) OR (GATTR.TYPTR^.FORM = LONGINT) THEN
        IF GATTR.TYPTR = INTPTR THEN
          BEGIN GENLDC(INTSIZE); GATTR.TYPTR := LONGINTPTR END
        ELSE
          IF FSP = INTPTR THEN
            BEGIN GENLDC(14(*DCV*)); GENNR(DECOPS); FSP := LONGINTPTR END
    END (*STRETCHIT*) ;

    PROCEDURE SIMPLEEXPRESSION(FSYS: SETOFSYS);
      VAR LATTR: ATTR; LOP: OPERATOR; SIGNED: BOOLEAN;

      PROCEDURE TERM(FSYS: SETOFSYS);
        VAR LATTR: ATTR; LSP: STP; LOP: OPERATOR;

        PROCEDURE FACTOR(FSYS: SETOFSYS);
          VAR LCP: CTP; LVP: CSP; VARPART,ALLCONST: BOOLEAN;
              LSP: STP; HIGHVAL,LOWVAL,LIC,LOP: INTEGER;
              CSTPART: SET OF 0..127;
        BEGIN
          IF NOT (SY IN FACBEGSYS) THEN
            BEGIN ERROR(58); SKIP(FSYS + FACBEGSYS);
              GATTR.TYPTR := NIL
            END;
          WHILE SY IN FACBEGSYS DO
            BEGIN
              CASE SY OF
        (*ID*)  IDENT:
                  BEGIN SEARCHID([KONST,FORMALVARS,ACTUALVARS,FIELD,FUNC],LCP);
                    INSYMBOL;
                    IF LCP^.KLASS = FUNC THEN
                      BEGIN CALL(FSYS,LCP); GATTR.KIND := EXPR END
                    ELSE
                      IF LCP^.KLASS = KONST THEN
                        WITH GATTR, LCP^ DO
                          BEGIN TYPTR := IDTYPE; KIND := CST;
                            CVAL := VALUES
                          END
```

```
              ELSE SELECTOR(FSYS,LCP);
            IF GATTR.TYPTR <> NIL THEN
              WITH GATTR,TYPTR^ DO
                IF FORM = SUBRANGE THEN TYPTR := RANGETYPE
          END;
(*CST*)   INTCONST:
          BEGIN
            WITH GATTR DO
              BEGIN TYPTR := INTPTR; KIND := CST;
                CVAL := VAL
              END;
            INSYMBOL
          END;
        REALCONST:
          BEGIN
            WITH GATTR DO
              BEGIN TYPTR := REALPTR; KIND := CST;
                CVAL := VAL
              END;
            INSYMBOL
          END;
        STRINGCONST:
          BEGIN
            WITH GATTR DO
              BEGIN
                IF LGTH = 1 THEN TYPTR := CHARPTR
                ELSE
                  BEGIN NEW(LSP,ARRAYS,TRUE,TRUE);
                    LSP^ := STRGPTR^;
                    LSP^.MAXLENG := LGTH;
                    TYPTR := LSP
                  END;
                KIND := CST; CVAL := VAL
              END;
            INSYMBOL
          END;
        LONGCONST:
          BEGIN
            WITH GATTR DO
              BEGIN NEW(LSP,LONGINT);
                LSP^ := LONGINTPTR^;
                LSP^.SIZE := DECSIZE(LGTH);
                TYPTR := LSP; KIND := CST; CVAL := VAL
              END;
            INSYMBOL
          END;
(*(*)     LPARENT:
          BEGIN INSYMBOL; EXPRESSION(FSYS + [RPARENT]);
            IF SY = RPARENT THEN INSYMBOL ELSE ERROR(4)
          END;
(*NOT*)   NOTSY:
            WITH GATTR DO
              BEGIN INSYMBOL; FACTOR(FSYS);
                IF (KIND = CST) AND (TYPTR = BOOLPTR) THEN
                  CVAL.IVAL := ORD(NOT ODD(CVAL.IVAL))
                ELSE
                BEGIN LOAD; GEN0(19(*NOT*));
                  IF TYPTR <> NIL THEN
                    IF TYPTR <> BOOLPTR THEN
                      BEGIN ERROR(135); TYPTR := NIL END
```

```
              END
            END;
(*[*)   LBRACK:
          BEGIN INSYMBOL; CSTPART := [ ]; VARPART := FALSE;
            NEW(LSP,POWER);
            WITH LSP^ DO
              BEGIN ELSET := NIL; SIZE := 0; FORM := POWER END;
            IF SY = RBRACK THEN
              BEGIN
                WITH GATTR DO
                  BEGIN TYPTR := LSP; KIND := CST END;
                INSYMBOL
              END
            ELSE
              BEGIN
                REPEAT EXPRESSION(FSYS + [COMMA,RBRACK,COLON]);
                  IF GATTR.TYPTR <> NIL THEN
                    IF GATTR.TYPTR^.FORM <> SCALAR THEN
                      BEGIN ERROR(136); GATTR.TYPTR := NIL END
                    ELSE
                      IF COMPTYPES(LSP^.ELSET,GATTR.TYPTR) THEN
                        BEGIN ALLCONST := FALSE; LOP := 23(*SGS*);
                          IF (GATTR.KIND = CST) AND
                             (GATTR.CVAL.IVAL <= 127) THEN
                            BEGIN ALLCONST := TRUE;
                              LOWVAL := GATTR.CVAL.IVAL;
                              HIGHVAL := LOWVAL
                            END;
                          LIC := IC; LOAD;
                          IF SY = COLON THEN
                            BEGIN INSYMBOL; LOP := 20(*SRS*);
                              EXPRESSION(FSYS + [COMMA,RBRACK]);
                              IF COMPTYPES(LSP^.ELSET,GATTR.TYPTR) THEN
                              ELSE
                                BEGIN ERROR(137); GATTR.TYPTR:=NIL END;
                              IF ALLCONST THEN
                                IF (GATTR.KIND = CST) AND
                                   (GATTR.CVAL.IVAL <= 127) THEN
                                    HIGHVAL := GATTR.CVAL.IVAL
                                ELSE
                                  BEGIN LOAD; ALLCONST := FALSE END
                              ELSE LOAD
                            END;
                          IF ALLCONST THEN
                            BEGIN IC := LIC; (*FORGET FIRST CONST*)
                              CSTPART := CSTPART + [LOWVAL..HIGHVAL]
                            END
                          ELSE
                            BEGIN GEN0(LOP);
                              IF VARPART THEN GEN0(28(*UNI*))
                              ELSE VARPART := TRUE
                            END;
                          LSP^.ELSET := GATTR.TYPTR;
                          GATTR.TYPTR := LSP
                        END
                      ELSE ERROR(137);
                  TEST := SY <> COMMA;
                  IF NOT TEST THEN INSYMBOL
                UNTIL TEST;
                IF SY = RBRACK THEN INSYMBOL ELSE ERROR(12)
```

```
                END;
              IF VARPART THEN
                BEGIN
                  IF CSTPART <> [ ] THEN
                    BEGIN
                      SCONST^.PVAL := CSTPART;
                      SCONST^.CCLASS := PSET;
                      GATTR.CVAL.VALP := SCONST;
                      GATTR.KIND := CST;
                      LOAD; GEN0(28(*UNI*))
                    END;
                  GATTR.KIND := EXPR
                END
              ELSE
                BEGIN
                  SCONST^.PVAL := CSTPART;
                  SCONST^.CCLASS := PSET;
                  GATTR.CVAL.VALP := SCONST;
                  GATTR.KIND := CST
                END
            END
        END (*CASE*) ;
        IF NOT (SY IN FSYS) THEN
          BEGIN ERROR(6); SKIP(FSYS + FACBEGSYS) END
      END (*WHILE*)
  END (*FACTOR*) ;

BEGIN (*TERM*)
  FACTOR(FSYS + [MULOP]);
  WHILE SY = MULOP DO
    BEGIN LOAD; LATTR := GATTR; LOP := OP;
      INSYMBOL; FACTOR(FSYS + [MULOP]); LOAD;
      IF (LATTR.TYPTR <> NIL) AND (GATTR.TYPTR <> NIL) THEN
        CASE LOP OF
(***)     MUL:  BEGIN FLOATIT(LATTR.TYPTR,FALSE); STRETCHIT(LATTR.TYPTR);
                IF (LATTR.TYPTR = INTPTR) AND (GATTR.TYPTR = INTPTR)
                  THEN GEN0(15(*MPI*))
                ELSE
                  IF (LATTR.TYPTR = REALPTR) AND
                     (GATTR.TYPTR = REALPTR) THEN GEN0(16(*MPR*))
                  ELSE
                    IF (GATTR.TYPTR^.FORM = LONGINT) AND
                       (LATTR.TYPTR^.FORM = LONGINT) THEN
                      BEGIN GENLDC(8(*DMP*)); GENNR(DECOPS) END
                    ELSE
                      IF (LATTR.TYPTR^.FORM = POWER)
                          AND COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
                        GEN0(12(*INT*))
                      ELSE BEGIN ERROR(134); GATTR.TYPTR:=NIL END
                END;
(*/*)     RDIV: BEGIN FLOATIT(LATTR.TYPTR,TRUE);
                IF (LATTR.TYPTR = REALPTR) AND
                   (GATTR.TYPTR = REALPTR) THEN GEN0(7(*DVR*))
                ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
                END;
(*DIV*)   IDIV: BEGIN STRETCHIT(LATTR.TYPTR);
                IF (LATTR.TYPTR = INTPTR) AND
                   (GATTR.TYPTR = INTPTR) THEN GEN0(6(*DVI*))
                ELSE
                  IF (LATTR.TYPTR^.FORM = LONGINT) AND
```

```
                        (GATTR.TYPTR^.FORM = LONGINT) THEN
                       BEGIN GENLDC(10(*DDV*)); GENNR(DECOPS) END
                     ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
                END;
  (*MOD*)   IMOD: IF (LATTR.TYPTR = INTPTR) AND
                     (GATTR.TYPTR = INTPTR) THEN GEN0(14(*MOD*))
                ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END;
  (*AND*)   ANDOP:IF (LATTR.TYPTR = BOOLPTR) AND
                     (GATTR.TYPTR = BOOLPTR) THEN GEN0(4(*AND*))
                ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
          END (*CASE*)
        ELSE GATTR.TYPTR := NIL
      END (*WHILE*)
  END (*TERM*) ;

BEGIN (*SIMPLEEXPRESSION*)
  SIGNED := FALSE;
  IF (SY = ADDOP) AND (OP IN [PLUS,MINUS]) THEN
    BEGIN SIGNED := OP = MINUS; INSYMBOL END;
  TERM(FSYS + [ADDOP]);
  IF SIGNED THEN
    BEGIN LOAD;
      IF GATTR.TYPTR = INTPTR THEN GEN0(17(*NGI*))
      ELSE
        IF GATTR.TYPTR = REALPTR THEN GEN0(18(*NGR*))
        ELSE
          IF GATTR.TYPTR^.FORM = LONGINT THEN
            BEGIN GENLDC(6(*DNG*)); GENNR(DECOPS) END
          ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
    END;
  WHILE SY = ADDOP DO
    BEGIN LOAD; LATTR := GATTR; LOP := OP;
      INSYMBOL; TERM(FSYS + [ADDOP]); LOAD;
      IF (LATTR.TYPTR <> NIL) AND (GATTR.TYPTR <> NIL) THEN
        CASE LOP OF
(*+*)     PLUS:
            BEGIN FLOATIT(LATTR.TYPTR,FALSE); STRETCHIT(LATTR.TYPTR);
              IF (LATTR.TYPTR = INTPTR)AND(GATTR.TYPTR = INTPTR) THEN
                GEN0(2(*ADI*))
              ELSE
                IF (LATTR.TYPTR = REALPTR)AND(GATTR.TYPTR = REALPTR) THEN
                  GEN0(3(*ADR*))
                ELSE
                  IF (GATTR.TYPTR^.FORM = LONGINT) AND
                     (LATTR.TYPTR^.FORM = LONGINT) THEN
                    BEGIN GENLDC(2(*DAD*)); GENNR(DECOPS) END
                  ELSE
                    IF (LATTR.TYPTR^.FORM = POWER)
                       AND COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
                      GEN0(28(*UNI*))
                    ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
            END;
(*-*)     MINUS:
            BEGIN FLOATIT(LATTR.TYPTR,FALSE); STRETCHIT(LATTR.TYPTR);
              IF (LATTR.TYPTR = INTPTR) AND (GATTR.TYPTR = INTPTR) THEN
                GEN0(21(*SBI*))
              ELSE
                IF (LATTR.TYPTR = REALPTR) AND (GATTR.TYPTR = REALPTR)
                  THEN GEN0(22(*SBR*))
                ELSE
```

```
                    IF (GATTR.TYPTR^.FORM = LONGINT) AND
                       (LATTR.TYPTR^.FORM = LONGINT) THEN
                      BEGIN GENLDC(4(*DSB*)); GENNR(DECOPS) END
                    ELSE
                      IF (LATTR.TYPTR^.FORM = POWER)
                          AND COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
                        GEN0(5(*DIF*))
                      ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
              END;
  (*OR*)    OROP:
              IF (LATTR.TYPTR = BOOLPTR) AND (GATTR.TYPTR = BOOLPTR) THEN
                GEN0(13(*IOR*))
              ELSE BEGIN ERROR(134); GATTR.TYPTR := NIL END
          END (*CASE*)
        ELSE GATTR.TYPTR := NIL
      END (*WHILE*)
  END (*SIMPLEEXPRESSION*) ;

  PROCEDURE MAKEPA(VAR STRGFSP: STP; PAFSP: STP);
    VAR LMIN,LMAX: INTEGER;
  BEGIN
    IF PAFSP^.INXTYPE <> NIL THEN
      BEGIN GETBOUNDS(PAFSP^.INXTYPE,LMIN,LMAX);
        IF LMAX-LMIN+1 <> STRGFSP^.MAXLENG THEN ERROR(129)
      END;
    STRGFSP := PAFSP
  END (*MAKEPA*) ;

BEGIN (*EXPRESSION*)
  SIMPLEEXPRESSION(FSYS + [RELOP]);
  IF SY = RELOP THEN
    BEGIN
      LSTRING := (GATTR.KIND = CST) AND
              (STRGTYPE(GATTR.TYPTR) OR (GATTR.TYPTR = CHARPTR));
      IF GATTR.TYPTR <> NIL THEN
        IF GATTR.TYPTR^.FORM <= POWER THEN LOAD
        ELSE LOADADDRESS;
      LATTR := GATTR; LOP := OP;
      INSYMBOL; SIMPLEEXPRESSION(FSYS);
      GSTRING := (GATTR.KIND = CST) AND
              (STRGTYPE(GATTR.TYPTR) OR (GATTR.TYPTR = CHARPTR));
      IF GATTR.TYPTR <> NIL THEN
        IF GATTR.TYPTR^.FORM <= POWER THEN LOAD
        ELSE LOADADDRESS;
      IF (LATTR.TYPTR <> NIL) AND (GATTR.TYPTR <> NIL) THEN
        IF LOP = INOP THEN
          IF GATTR.TYPTR^.FORM = POWER THEN
            IF COMPTYPES(LATTR.TYPTR,GATTR.TYPTR^.ELSET) THEN
              GEN0(11(*INN*))
            ELSE BEGIN ERROR(129); GATTR.TYPTR := NIL END
          ELSE BEGIN ERROR(130); GATTR.TYPTR := NIL END
        ELSE
          BEGIN
            IF LATTR.TYPTR <> GATTR.TYPTR THEN
              BEGIN FLOATIT(LATTR.TYPTR,FALSE); STRETCHIT(LATTR.TYPTR) END;
            IF LSTRING THEN
              BEGIN
                IF PAOFCHAR(GATTR.TYPTR) THEN
                  IF NOT GATTR.TYPTR^.AISSTRNG THEN
                    BEGIN GEN0(29(*S2P*));
```

```
          MAKEPA(LATTR.TYPTR,GATTR.TYPTR)
        END
  END
ELSE
  IF GSTRING THEN
    BEGIN
      IF PAOFCHAR(LATTR.TYPTR) THEN
        IF NOT LATTR.TYPTR^.AISSTRNG THEN
          BEGIN GEN0(80(*S1P*));
            MAKEPA(GATTR.TYPTR,LATTR.TYPTR)
          END;
    END;
IF (LSTRING AND STRGTYPE(GATTR.TYPTR)) OR
   (GSTRING AND STRGTYPE(LATTR.TYPTR)) THEN GOTO 1;
IF COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
  BEGIN LSIZE := LATTR.TYPTR^.SIZE; (*INVALID FOR LONG INTEGERS*)
    CASE LATTR.TYPTR^.FORM OF
      SCALAR:
        IF LATTR.TYPTR = REALPTR THEN TYPIND := 1
        ELSE
          IF LATTR.TYPTR = BOOLPTR THEN TYPIND := 3
          ELSE TYPIND := 0;
      POINTER:
        BEGIN
          IF LOP IN [LTOP,LEOP,GTOP,GEOP] THEN ERROR(131);
          TYPIND := 0
        END;
      LONGINT: TYPIND := 7;
      POWER:
        BEGIN
          IF LOP IN [LTOP,GTOP] THEN ERROR(132);
          TYPIND := 4
        END;
      ARRAYS:
        BEGIN
          TYPIND := 6;
          IF PAOFCHAR(LATTR.TYPTR) THEN
            IF LATTR.TYPTR^.AISSTRNG THEN
1:            TYPIND := 2
            ELSE
              BEGIN TYPIND := 5;
                IF LATTR.TYPTR^.INXTYPE <> NIL THEN
                  BEGIN
                    GETBOUNDS(LATTR.TYPTR^.INXTYPE,LMIN,LMAX);
                    LSIZE := LMAX - LMIN + 1
                  END
              END
          ELSE
            IF LOP IN [LTOP,LEOP,GTOP,GEOP] THEN ERROR(131)
        END;
      RECORDS:
        BEGIN
          IF LOP IN [LTOP,LEOP,GTOP,GEOP] THEN ERROR(131);
          TYPIND := 6
        END;
      FILES:
        BEGIN ERROR(133); TYPIND := 0 END
    END;
    IF TYPIND = 7 THEN
      BEGIN GENLDC(ORD(LOP)); GENLDC(16(*DCMP*));
```

```
                    GENNR(DECOPS)
                  END
                ELSE
                  CASE LOP OF
                    LTOP: GEN2(53(*LES*),TYPIND,LSIZE);
                    LEOP: GEN2(52(*LEQ*),TYPIND,LSIZE);
                    GTOP: GEN2(49(*GRT*),TYPIND,LSIZE);
                    GEOP: GEN2(48(*GEQ*),TYPIND,LSIZE);
                    NEOP: GEN2(55(*NEQ*),TYPIND,LSIZE);
                    EQOP: GEN2(47(*EQU*),TYPIND,LSIZE)
                  END
              END
            ELSE ERROR(129)
          END;
        GATTR.TYPTR := BOOLPTR; GATTR.KIND := EXPR
      END (*SY = RELOP*)
  END (*EXPRESSION*) ;

(* $I BODYPART.E.TEXT*)

(*    COPYRIGHT (C) 1978, REGENTS OF THE        *)
(*    UNIVERSITY OF CALIFORNIA, SAN DIEGO       *)

  PROCEDURE STATEMENT(FSYS: SETOFSYS);
    LABEL 1;
    VAR LCP: CTP; TTOP: DISPRANGE; LLP: LABELP; HEAP: ^INTEGER;

    PROCEDURE ASSIGNMENT(FCP: CTP);
      VAR LATTR: ATTR; CSTRING,PAONLEFT: BOOLEAN; LMIN,LMAX: INTEGER;
    BEGIN SELECTOR(FSYS + [BECOMES],FCP);
      IF SY = BECOMES THEN
        BEGIN LMAX := 0; CSTRING := FALSE;
          IF GATTR.TYPTR <> NIL THEN
            IF (GATTR.ACCESS = INDRCT) OR (GATTR.TYPTR^.FORM > POWER) THEN
              LOADADDRESS;
          PAONLEFT := PAOFCHAR(GATTR.TYPTR);
          LATTR := GATTR;
          INSYMBOL; EXPRESSION(FSYS);
          IF GATTR.KIND = CST THEN
            CSTRING := (GATTR.TYPTR = CHARPTR) OR STRGTYPE(GATTR.TYPTR);
          IF GATTR.TYPTR <> NIL THEN
            IF GATTR.TYPTR^.FORM <= POWER THEN LOAD
            ELSE LOADADDRESS;
          IF (LATTR.TYPTR <> NIL) AND (GATTR.TYPTR <> NIL) THEN
            BEGIN
              IF GATTR.TYPTR = INTPTR THEN
                IF COMPTYPES(REALPTR,LATTR.TYPTR) THEN
                  BEGIN GEN0(10(*FLT*)); GATTR.TYPTR := REALPTR END;
              IF COMPTYPES(LONGINTPTR,LATTR.TYPTR) THEN
                BEGIN
                  IF GATTR.TYPTR = INTPTR THEN
                    BEGIN GENLDC(INTSIZE);
                      GATTR.TYPTR := LONGINTPTR
                    END;
                  IF GATTR.TYPTR^.FORM <> LONGINT THEN
                    BEGIN ERROR(129); GATTR.TYPTR := LONGINTPTR END
                END;
              IF PAONLEFT THEN
                IF LATTR.TYPTR^.AISSTRNG THEN
                  IF CSTRING AND (GATTR.TYPTR = CHARPTR) THEN
```

```
              GATTR.TYPTR := STRGPTR
            ELSE
          ELSE
            IF LATTR.TYPTR^.INXTYPE <> NIL THEN
              BEGIN GETBOUNDS(LATTR.TYPTR^.INXTYPE,LMIN,LMAX);
                LMAX := LMAX - LMIN + 1;
                IF CSTRING AND (GATTR.TYPTR <> CHARPTR) THEN
                  BEGIN GEN0(80(*S1P*));
                    IF LMAX <> GATTR.TYPTR^.MAXLENG THEN ERROR(129);
                    GATTR.TYPTR := LATTR.TYPTR
                  END
              END
            ELSE GATTR.TYPTR := LATTR.TYPTR;
          IF COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
            CASE LATTR.TYPTR^.FORM OF
              SUBRANGE: BEGIN
                          IF RANGECHECK THEN
                            BEGIN
                              GENLDC(LATTR.TYPTR^.MIN.IVAL);
                              GENLDC(LATTR.TYPTR^.MAX.IVAL);
                              GEN0(8(*CHK*))
                            END;
                          STORE(LATTR)
                        END;
              POWER:    BEGIN
                          GEN1(32(*ADJ*),LATTR.TYPTR^.SIZE);
                          STORE(LATTR)
                        END;
              SCALAR,
              POINTER:  STORE(LATTR);
              LONGINT: BEGIN
                          GENLDC(LATTR.TYPTR^.SIZE);
                          GENLDC(0(*DAJ*));
                          GENNR(DECOPS);
                          STORE(LATTR)
                        END;
              ARRAYS:  IF PAONLEFT THEN
                         IF LATTR.TYPTR^.AISSTRNG THEN
                           GEN1(42(*SAS*),LATTR.TYPTR^.MAXLENG)
                         ELSE GEN1(41(*MVB*),LMAX)
                       ELSE GEN1(40(*MOV*),LATTR.TYPTR^.SIZE);
              RECORDS: GEN1(40(*MOV*),LATTR.TYPTR^.SIZE);
              FILES:   ERROR(146)
            END
          ELSE ERROR(129)
        END
    END (*SY = BECOMES*)
  ELSE ERROR(51)
END (*ASSIGNMENT*) ;

PROCEDURE GOTOSTATEMENT;
  VAR LLP: LABELP; FOUND: BOOLEAN; TTOP: DISPRANGE;
BEGIN
  IF NOT GOTOOK THEN ERROR(6);
  IF SY = INTCONST THEN
    BEGIN
      FOUND := FALSE; TTOP := TOP;
      WHILE DISPLAY[TTOP].OCCUR <> BLCK DO TTOP := TTOP - 1;
      LLP := DISPLAY[TTOP].FLABEL;
      WHILE (LLP <> NIL) AND NOT FOUND DO
```

```
        WITH LLP^ DO
          IF LABVAL = VAL.IVAL THEN
            BEGIN FOUND := TRUE;
              GENJMP(57(*UJP*),CODELBP)
            END
          ELSE LLP := NEXTLAB;
      IF NOT FOUND THEN ERROR(167);
      INSYMBOL
    END
  ELSE ERROR(15)
END (*GOTOSTATEMENT*) ;

PROCEDURE COMPOUNDSTATEMENT;
BEGIN
  REPEAT
    REPEAT STATEMENT(FSYS + [SEMICOLON,ENDSY])
    UNTIL NOT (SY IN STATBEGSYS);
    TEST := SY <> SEMICOLON;
    IF NOT TEST THEN INSYMBOL
  UNTIL TEST;
  IF SY = ENDSY THEN INSYMBOL ELSE ERROR(13)
END (*COMPOUNDSTATEMENET*) ;

PROCEDURE IFSTATEMENT;
  VAR LCIX1,LCIX2: LBP; LIC: INTEGER; CONDCOMPILE,NOTHENCLAUSE: BOOLEAN;
BEGIN
  CONDCOMPILE := FALSE;
  EXPRESSION(FSYS + [THENSY]);
  IF (GATTR.KIND = CST) THEN
    IF (GATTR.TYPTR = BOOLPTR) THEN
      BEGIN CONDCOMPILE := TRUE;
        NOTHENCLAUSE := NOT ODD(GATTR.CVAL.IVAL);
        LIC := IC
      END;
  IF NOT CONDCOMPILE THEN
    BEGIN GENLABEL(LCIX1); GENFJP(LCIX1) END;
  IF SY = THENSY THEN INSYMBOL ELSE ERROR(52);
  STATEMENT(FSYS + [ELSESY]);
  IF CONDCOMPILE THEN
    IF NOTHENCLAUSE THEN IC := LIC
    ELSE LIC := IC;
  IF SY = ELSESY THEN
    BEGIN
      IF NOT CONDCOMPILE THEN
        BEGIN GENLABEL(LCIX2); GENJMP(57(*UJP*),LCIX2); PUTLABEL(LCIX1) END;
      INSYMBOL; STATEMENT(FSYS);
      IF CONDCOMPILE THEN
        BEGIN
          IF NOT NOTHENCLAUSE THEN IC := LIC
        END
      ELSE PUTLABEL(LCIX2)
    END
  ELSE
    IF NOT CONDCOMPILE THEN PUTLABEL(LCIX1)
END (*IFSTATEMENT*) ;

PROCEDURE CASESTATEMENT;
  LABEL 1;
  TYPE CIP = ^CASEINFO;
       CASEINFO = RECORD
```

```
                    NEXT: CIP;
                    CSSTART: INTEGER;
                    CSLAB: INTEGER
                  END;
  VAR LSP,LSP1: STP; FSTPTR,LPT1,LPT2,LPT3: CIP; LVAL: VALU;
      LADDR, LCIX: LBP; NULSTMT, LMIN, LMAX: INTEGER;
BEGIN EXPRESSION(FSYS + [OFSY,COMMA,COLON]);
  LOAD; GENLABEL(LCIX); GENJMP(57(*UJP*),LCIX);
  LSP := GATTR.TYPTR;
  IF LSP <> NIL THEN
    IF (LSP^.FORM <> SCALAR) OR (LSP = REALPTR) THEN
      BEGIN ERROR(144); LSP := NIL END;
  IF SY = OFSY THEN INSYMBOL ELSE ERROR(8);
  FSTPTR := NIL; GENLABEL(LADDR);
  REPEAT
    LPT3 := NIL;
    REPEAT CONSTANT(FSYS + [COMMA,COLON],LSP1,LVAL);
      IF LSP <> NIL THEN
        IF COMPTYPES(LSP,LSP1) THEN
          BEGIN LPT1 := FSTPTR; LPT2 := NIL;
            WHILE LPT1 <> NIL DO
              WITH LPT1^ DO
                BEGIN
                  IF CSLAB <= LVAL.IVAL THEN
                    BEGIN IF CSLAB = LVAL.IVAL THEN ERROR(156);
                      GOTO 1
                    END;
                  LPT2 := LPT1; LPT1 := NEXT
                END;
1:          NEW(LPT3);
            WITH LPT3^ DO
              BEGIN NEXT := LPT1; CSLAB := LVAL.IVAL;
                CSSTART := IC
              END;
            IF LPT2 = NIL THEN FSTPTR := LPT3
            ELSE LPT2^.NEXT := LPT3
          END
        ELSE ERROR(147);
      TEST := SY <> COMMA;
      IF NOT TEST THEN INSYMBOL
    UNTIL TEST;
    IF SY = COLON THEN INSYMBOL ELSE ERROR(5);
    REPEAT STATEMENT(FSYS + [SEMICOLON])
    UNTIL NOT (SY IN STATBEGSYS);
    IF LPT3 <> NIL THEN
      GENJMP(57(*UJP*),LADDR);
    TEST := SY <> SEMICOLON;
    IF NOT TEST THEN INSYMBOL
  UNTIL TEST OR (SY = ENDSY);
  PUTLABEL(LCIX);
  IF FSTPTR <> NIL THEN
    BEGIN LMAX := FSTPTR^.CSLAB;
      LPT1 := FSTPTR; FSTPTR := NIL;
      REPEAT LPT2 := LPT1^.NEXT; LPT1^.NEXT := FSTPTR;
        FSTPTR := LPT1; LPT1 := LPT2
      UNTIL LPT1 = NIL;
      LMIN := FSTPTR^.CSLAB;
          GEN0(44(*XJP*));
          GENWORD(LMIN); GENWORD(LMAX);
          NULSTMT := IC;
```

```
          GENJMP(57(*UJP*),LADDR);
          REPEAT
            WITH FSTPTR^ DO
              BEGIN
                WHILE CSLAB > LMIN DO
                  BEGIN GENWORD(IC-NULSTMT); LMIN := LMIN + 1 END;
                GENWORD(IC-CSSTART);
                FSTPTR := NEXT; LMIN := LMIN + 1
              END
          UNTIL FSTPTR = NIL;
          PUTLABEL(LADDR)
    END;
    IF SY = ENDSY THEN INSYMBOL ELSE ERROR(13)
END (*CASESTATEMENT*) ;

PROCEDURE REPEATSTATEMENT;
  VAR LADDR: LBP;
BEGIN GENLABEL(LADDR); PUTLABEL(LADDR);
  REPEAT
    REPEAT STATEMENT(FSYS + [SEMICOLON,UNTILSY])
    UNTIL NOT (SY IN STATBEGSYS);
    TEST := SY <> SEMICOLON;
    IF NOT TEST THEN INSYMBOL
  UNTIL TEST;
  IF SY = UNTILSY THEN
    BEGIN INSYMBOL; EXPRESSION(FSYS); GENFJP(LADDR)
    END
  ELSE ERROR(53)
END (*REPEATSTATEMENT*) ;

PROCEDURE WHILESTATEMENT;
  VAR LADDR, LCIX: LBP;
BEGIN GENLABEL(LADDR); PUTLABEL(LADDR);
  EXPRESSION(FSYS + [DOSY]); GENLABEL(LCIX); GENFJP(LCIX);
  IF SY = DOSY THEN INSYMBOL ELSE ERROR(54);
  STATEMENT(FSYS); GENJMP(57(*UJP*),LADDR); PUTLABEL(LCIX)
END (*WHILESTATEMENT*) ;

PROCEDURE FORSTATEMENT;
  VAR LATTR: ATTR; LSP: STP;  LSY: SYMBOL;
      LCIX, LADDR: LBP;
BEGIN
  IF SY = IDENT THEN
    BEGIN SEARCHID(VARS,LCP);
      WITH LCP^, LATTR DO
        BEGIN TYPTR := IDTYPE; KIND := VARBL;
          IF KLASS = ACTUALVARS THEN
            BEGIN ACCESS := DRCT; VLEVEL := VLEV;
              DPLMT := VADDR
            END
          ELSE BEGIN ERROR(155); TYPTR := NIL END
        END;
      IF LATTR.TYPTR <> NIL THEN
        IF (LATTR.TYPTR^.FORM > SUBRANGE)
           OR COMPTYPES(REALPTR,LATTR.TYPTR) THEN
          BEGIN ERROR(143); LATTR.TYPTR := NIL END;
      INSYMBOL
    END
  ELSE
    BEGIN ERROR(2); SKIP(FSYS + [BECOMES,TOSY,DOWNTOSY,DOSY])
```

```
      END;
    IF SY = BECOMES THEN
      BEGIN INSYMBOL; EXPRESSION(FSYS + [TOSY,DOWNTOSY,DOSY]);
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR^.FORM <> SCALAR THEN ERROR(144)
            ELSE
              IF COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
                BEGIN LOAD;
                  IF LATTR.TYPTR <> NIL THEN
                    IF (LATTR.TYPTR^.FORM = SUBRANGE) AND RANGECHECK THEN
                      BEGIN
                        GENLDC(LATTR.TYPTR^.MIN.IVAL);
                        GENLDC(LATTR.TYPTR^.MAX.IVAL);
                        GEN0(8(*CHK*))
                      END;
                  STORE(LATTR)
                END
              ELSE ERROR(145)
      END
    ELSE
      BEGIN ERROR(51); SKIP(FSYS + [TOSY,DOWNTOSY,DOSY]) END;
    GENLABEL(LADDR);
    IF SY IN [TOSY,DOWNTOSY] THEN
      BEGIN LSY := SY; INSYMBOL; EXPRESSION(FSYS + [DOSY]);
        IF GATTR.TYPTR <> NIL THEN
          IF GATTR.TYPTR^.FORM <> SCALAR THEN ERROR(144)
          ELSE
            IF COMPTYPES(LATTR.TYPTR,GATTR.TYPTR) THEN
              BEGIN LOAD;
                IF LATTR.TYPTR <> NIL THEN
                  IF (LATTR.TYPTR^.FORM = SUBRANGE) AND RANGECHECK THEN
                    BEGIN
                      GENLDC(LATTR.TYPTR^.MIN.IVAL);
                      GENLDC(LATTR.TYPTR^.MAX.IVAL);
                      GEN0(8(*CHK*))
                    END;
                GEN2(56(*STR*),0,LC); PUTLABEL(LADDR);
                GATTR := LATTR; LOAD; GEN2(54(*LOD*),0,LC);
                LC := LC + INTSIZE;
                IF LC > LCMAX THEN LCMAX := LC;
                IF LSY = TOSY THEN GEN2(52(*LEQ*),0,INTSIZE)
                ELSE GEN2(48(*GEQ*),0,INTSIZE);
              END
            ELSE ERROR(145)
      END
    ELSE BEGIN ERROR(55); SKIP(FSYS + [DOSY]) END;
    GENLABEL(LCIX); GENJMP(33(*FJP*),LCIX);
    IF SY = DOSY THEN INSYMBOL ELSE ERROR(54);
    STATEMENT(FSYS);
    GATTR := LATTR; LOAD; GENLDC(1);
    IF LSY = TOSY THEN GEN0(2(*ADI*)) ELSE GEN0(21(*SBI*));
    STORE(LATTR); GENJMP(57(*UJP*),LADDR); PUTLABEL(LCIX);
    LC := LC - INTSIZE
  END (*FORSTATEMENT*) ;


  PROCEDURE WITHSTATEMENT;
    VAR LCP: CTP; LCNT1,LCNT2: DISPRANGE;
  BEGIN LCNT1 := 0; LCNT2 := 0;
    REPEAT
```

```
      IF SY = IDENT THEN
        BEGIN SEARCHID(VARS + [FIELD],LCP); INSYMBOL END
      ELSE BEGIN ERROR(2); LCP := UVARPTR END;
      SELECTOR(FSYS + [COMMA,DOSY],LCP);
      IF GATTR.TYPTR <> NIL THEN
        IF GATTR.TYPTR^.FORM = RECORDS THEN
          IF TOP < DISPLIMIT THEN
            BEGIN TOP := TOP + 1; LCNT1 := LCNT1 + 1;
              WITH DISPLAY[TOP] DO
                BEGIN FNAME := GATTR.TYPTR^.FSTFLD END;
              IF GATTR.ACCESS = DRCT THEN
                WITH DISPLAY[TOP] DO
                  BEGIN OCCUR := CREC; CLEV := GATTR.VLEVEL;
                    CDSPL := GATTR.DPLMT
                  END
              ELSE
                BEGIN LOADADDRESS; GEN2(56(*STR*),0,LC);
                  WITH DISPLAY[TOP] DO
                    BEGIN OCCUR := VREC; VDSPL := LC END;
                  LC := LC + PTRSIZE; LCNT2 := LCNT2 + PTRSIZE;
                  IF LC > LCMAX THEN LCMAX := LC
                END
            END
          ELSE ERROR(250)
        ELSE ERROR(140);
      TEST := SY <> COMMA;
      IF NOT TEST THEN INSYMBOL
    UNTIL TEST;
    IF SY = DOSY THEN INSYMBOL ELSE ERROR(54);
    STATEMENT(FSYS);
    TOP := TOP - LCNT1; LC := LC - LCNT2;
  END (*WITHSTATEMENT*) ;

BEGIN (*STATEMENT*)
  STMTLEV := STMTLEV + 1;
  IF SY = INTCONST THEN (*LABEL*)
    BEGIN TTOP := TOP;
      WHILE DISPLAY[TTOP].OCCUR <> BLCK DO TTOP := TTOP-1;
      LLP := DISPLAY[TTOP].FLABEL;
      WHILE LLP <> NIL DO
        WITH LLP^ DO
          IF LABVAL = VAL.IVAL THEN
            BEGIN
              IF CODELBP^.DEFINED THEN ERROR(165);
              PUTLABEL(CODELBP); GOTO 1
            END
          ELSE LLP := NEXTLAB;
      ERROR(167);
1:    INSYMBOL;
      IF SY = COLON THEN INSYMBOL ELSE ERROR(5)
    END;
  IF DEBUGGING THEN
    BEGIN GEN1(85(*BPT*),SCREENDOTS+1); BPTONLINE := TRUE END;
  IF NOT (SY IN FSYS + [IDENT]) THEN
    BEGIN ERROR(6); SKIP(FSYS) END;
  IF SY IN STATBEGSYS + [IDENT] THEN
    BEGIN MARK(HEAP); (*FOR LABEL CLEANUP*)
      CASE SY OF
        IDENT:    BEGIN SEARCHID(VARS + [FIELD,FUNC,PROC],LCP);
                    INSYMBOL;
```

```
                    IF LCP^.KLASS = PROC THEN CALL(FSYS,LCP)
                    ELSE ASSIGNMENT(LCP)
                  END;
          BEGINSY:  BEGIN INSYMBOL; COMPOUNDSTATEMENT END;
          GOTOSY:   BEGIN INSYMBOL; GOTOSTATEMENT END;
          IFSY:     BEGIN INSYMBOL; IFSTATEMENT END;
          CASESY:   BEGIN INSYMBOL; CASESTATEMENT END;
          WHILESY:  BEGIN INSYMBOL; WHILESTATEMENT END;
          REPEATSY: BEGIN INSYMBOL; REPEATSTATEMENT END;
          FORSY:    BEGIN INSYMBOL; FORSTATEMENT END;
          WITHSY:   BEGIN INSYMBOL; WITHSTATEMENT END
        END;
        RELEASE(HEAP);
        IF IC + 100 > MAXCODE THEN
          BEGIN ERROR(253); IC := 0 END;
        IF NOT (SY IN [SEMICOLON,ENDSY,ELSESY,UNTILSY]) THEN
          BEGIN ERROR(6); SKIP(FSYS) END
      END;
    STMTLEV := STMTLEV - 1
  END (*STATEMENT*) ;

PROCEDURE BODY;

VAR LLC1,EXITIC: ADDRRANGE;  LCP: CTP;  LOP: OPRANGE;
     LLP: LABELP;  LMIN,LMAX: INTEGER;  JTINX: JTABRANGE;
     DUMMYVAR: ARRAY[0..0] OF INTEGER; (*FOR PRETTY DISPLAY OF STACK AND HEAP*)

BEGIN
  IF (NOSWAP) AND (STARTINGUP) THEN
    BEGIN
      DECLARATIONPART(FSYS); (* BRING IN DECLARATIONPART *)
      EXIT(BODYPART);
    END;
  NEXTJTAB := 1;
  IF NOISY THEN
    BEGIN WRITELN(OUTPUT);
      IF NOT NOSWAP THEN (*MUST ADJUST DISPLAY OF STACK AND HEAP*)
        UNITWRITE(3,DUMMYVAR[-1600],35);
      DUMMYVAR[0]:=MEMAVAIL;
      IF DUMMYVAR[0] < SMALLESTSPACE THEN SMALLESTSPACE:=DUMMYVAR[0];
      IF FPROCP <> NIL THEN
          WRITELN(OUTPUT,FPROCP^.NAME,' [',DUMMYVAR[0]:5,' words]');
      WRITE(OUTPUT,'<',SCREENDOTS:4,'>')
    END;
  IF FPROCP <> NIL THEN
    BEGIN
      LLC1 := FPROCP^.LOCALLC; LCP := FPROCP^.NEXT;
      WHILE LCP <> NIL DO
        WITH LCP^ DO
          BEGIN
              IF IDTYPE <> NIL THEN
                IF (KLASS = ACTUALVARS) THEN
                  IF (IDTYPE^.FORM > POWER) THEN
                    BEGIN LLC1 := LLC1 - PTRSIZE;
                      GEN2(50(*LDA*),0,VADDR);
                      GEN2(54(*LOD*),0,LLC1);
                      IF PAOFCHAR(IDTYPE) THEN
                        WITH IDTYPE^ DO
                          IF AISSTRNG THEN GEN1(42(*SAS*),MAXLENG)
                          ELSE
```

```
                          IF INXTYPE <> NIL THEN
                            BEGIN GETBOUNDS(INXTYPE,LMIN,LMAX);
                              GEN1(41(*MVB*),LMAX - LMIN + 1)
                            END
                          ELSE
                      ELSE GEN1(40(*MOV*),IDTYPE^.SIZE)
                    END
                  ELSE LLC1 := LLC1 - IDTYPE^.SIZE
                ELSE
                  IF KLASS = FORMALVARS THEN LLC1 := LLC1 - PTRSIZE;
            LCP := NEXT
          END;
  END;
STARTDOTS := SCREENDOTS;
LCMAX := LC;
LLP := DISPLAY[TOP].FLABEL;
WHILE LLP <> NIL DO
  BEGIN GENLABEL(LLP^.CODELBP);
    LLP := LLP^.NEXTLAB
  END;
IF NOT INMODULE THEN
  IF LEVEL = 1 THEN
    BEGIN LCP := USINGLIST;
      WHILE LCP <> NIL DO
        BEGIN
          IF LCP^.SEGID >= 0 THEN
            BEGIN GENLDC(LCP^.SEGID); GEN1(30(*CSP*),21(*GETSEG*)) END;
          LCP := LCP^.NEXT
        END;
      IF USERINFO.STUPID THEN
          GEN2(77(*CXP*),6(*TURTLE*),1(*INIT*))
    END;
LCP := DISPLAY[TOP].FFILE;
WHILE LCP <> NIL DO
  WITH LCP^,IDTYPE^ DO
    BEGIN
      GEN2(50(*LDA*),0,VADDR);
      GEN2(50(*LDA*),0,VADDR+FILESIZE);
      IF FILTYPE = NIL THEN GENLDC(-1)
      ELSE
        IF IDTYPE = INTRACTVPTR THEN GENLDC(0)
        ELSE
          IF FILTYPE = CHARPTR THEN GENLDC(-2)
          ELSE GENLDC(FILTYPE^.SIZE);
      GEN2(77(*CXP*),0(*SYS*),3(*FINIT*));
      LCP := NEXT
    END;
IF (LEVEL = 1) AND NOT SYSCOMP THEN
  GEN1(85(*BPT*),SCREENDOTS+1);
REPEAT
  REPEAT STATEMENT(FSYS + [SEMICOLON,ENDSY])
  UNTIL NOT (SY IN STATBEGSYS);
  TEST := SY <> SEMICOLON;
  IF NOT TEST THEN INSYMBOL
UNTIL TEST;
IF SY = ENDSY THEN INSYMBOL ELSE ERROR(13);
EXITIC := IC;
LCP := DISPLAY[TOP].FFILE;
WHILE LCP <> NIL DO
  WITH LCP^ DO
```

```
      BEGIN
        GEN2(50(*LDA*),0,VADDR);
        GENLDC(0); GEN2(77(*CXP*),0(*SYS*),6(*FCLOSE*));
        LCP := NEXT
      END;
  IF NOT INMODULE THEN
    IF LEVEL = 1 THEN
      BEGIN
        LCP := USINGLIST;
        WHILE LCP <> NIL DO
          BEGIN
            IF LCP^.SEGID >= 0 THEN
              BEGIN GENLDC(LCP^.SEGID); GEN1(30(*CSP*),22(*RELSEG*)) END;
            LCP := LCP^.NEXT
          END
      END;
  IF FPROCP = NIL THEN GEN0(86(*XIT*))
  ELSE
    BEGIN
      IF FPROCP^.PFLEV = 0 THEN LOP := 65(*RBP*)
      ELSE LOP := 45(*RNP*);
      IF FPROCP^.IDTYPE = NIL THEN GEN1(LOP,0)
      ELSE GEN1(LOP,FPROCP^.IDTYPE^.SIZE)
    END;
  LLP := DISPLAY[TOP].FLABEL;  (* CHECK UNDEFINED LABELS *)
  WHILE LLP <> NIL DO
    WITH LLP^,CODELBP^ DO
      BEGIN
        IF NOT DEFINED THEN
          IF REFLIST <> MAXADDR THEN ERROR(168);
        LLP := NEXTLAB
      END;
  JTINX := NEXTJTAB - 1;
  IF ODD(IC) THEN IC := IC + 1;
  WHILE JTINX > 0 DO
    BEGIN GENWORD(IC-JTAB[JTINX]); JTINX := JTINX-1 END;
  IF FPROCP = NIL THEN
    BEGIN GENWORD((LCMAX-LCAFTERMARKSTACK)*2); GENWORD(0) END
  ELSE
    WITH FPROCP^ DO
      BEGIN GENWORD((LCMAX-LOCALLC)*2);
        GENWORD((LOCALLC-LCAFTERMARKSTACK)*2)
      END;
  GENWORD(IC-EXITIC); GENWORD(IC);
  GENBYTE(CURPROC); GENBYTE(LEVEL-1);
  IF NOT CODEINSEG THEN
    BEGIN CODEINSEG := TRUE;
      SEGTABLE[SEG].DISKADDR := CURBLK
    END;
  WRITECODE(FALSE);
  SEGINX := SEGINX + IC;
  PROCTABLE[CURPROC] := SEGINX - 2
END (*BODY*) ;

BEGIN (*BODYPART*)
  BODY
END ;

(* $I UNITPART.TEXT*)
```

```
    (*************************************************************)
    (*                                                           *)
    (*  Copyright (c) l978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                           *)
    (*************************************************************)

SEGMENT PROCEDURE WRITELINKERINFO(DECSTUFF:BOOLEAN);

  TYPE
    LITYPES = (EOFMARK,MODDULE,GLOBREF,PUBBLIC,PRIVVATE,CONNSTANT,GLOBDEF,
               PUBLICDEF,CONSTDEF,EXTPROC,EXTFUNC,SSEPPROC,SSEPFUNC,
               SEPPREF,SEPFREF);
    OPFORMAT = (WORD, BYTE, BIG);
    LIENTRY = RECORD
                LINAME: ALPHA;
                CASE LITYPE: LITYPES OF
                  MODDULE,
                  PUBBLIC,
                  PRIVVATE,
                  SEPPREF,
                  SEPFREF:          (FORMAT: OPFORMAT;
                                     NREFS: INTEGER;
                                     NWORDS: INTEGER);
                  CONSTDEF:         (CONSTANT: INTEGER);
                  PUBLICDEF:        (BASEOFFSET: INTEGER);
                  EXTPROC,EXTFUNC,
                  SSEPPROC,SSEPFUNC:(PROCNUM: INTEGER;
                                     NPARAMS: INTEGER;
                                     RANGE: ^INTEGER)
              END;

  VAR FCP,LCP: CTP; CURRENTBLOCK: INTEGER; I: NONRESIDENT;
      EXTNAME: ALPHA; FIC: ADDRRANGE;
      LIREC: LIENTRY;

  PROCEDURE GETREFS(ID,LENGTH: INTEGER);
    VAR LIC: ADDRRANGE; J,MAX,BLOCKCOUNT,COUNT: INTEGER;

    PROCEDURE GETNEXTBLOCK;
    BEGIN
      CURRENTBLOCK := CURRENTBLOCK + 1;
      IF CURRENTBLOCK > REFBLK THEN CURRENTBLOCK := 0;
      IF BLOCKREAD(REFFILE,REFLIST^,1,CURRENTBLOCK) <> 1 THEN;
    END (*GETNEXTBLOCK*) ;

  BEGIN (*GETREFS*)
    IF (NREFS = 1) AND (REFBLK = 0) THEN EXIT(GETREFS);
    COUNT := 0;
    FOR BLOCKCOUNT := 0 TO REFBLK DO
      BEGIN
        IF CURRENTBLOCK < REFBLK THEN MAX := REFSPERBLK ELSE MAX := NREFS-1;
        FOR J := 1 TO MAX DO
          IF ID = REFLIST^[J].KEY THEN
            BEGIN GENWORD(REFLIST^[J].OFFSET); COUNT := COUNT + 1 END;
        IF BLOCKCOUNT < REFBLK THEN GETNEXTBLOCK;
      END;
    LIC := IC; IC := FIC; GENWORD(COUNT); IC := LIC;
```

```
  (*NOW FILL REST OF 8-WORD RECORD*)
  FOR J := 1 TO ((8 - (COUNT MOD 8)) MOD 8) DO GENWORD(0)
END (* GETREFS *) ;

PROCEDURE GLOBALSEARCH(FCP: CTP);
  VAR NEEDEDBYLINKER: BOOLEAN;

BEGIN
  NEEDEDBYLINKER := TRUE;
  WITH LIREC,FCP^ DO
    CASE KLASS OF
      TYPES: NEEDEDBYLINKER := FALSE;
      KONST: IF (IDTYPE^.SIZE = 1) AND NOT INMODULE THEN
               BEGIN LITYPE := CONSTDEF;
                 CONSTANT := VALUES.IVAL
               END
             ELSE NEEDEDBYLINKER := FALSE;
      FORMALVARS,
      ACTUALVARS:
             BEGIN
               IF INMODULE THEN
                 BEGIN
                   IF PUBLIC THEN
                     BEGIN LITYPE := PUBBLIC;
                       NWORDS := 0
                     END
                   ELSE
                     BEGIN LITYPE := PRIVVATE;
                       IF KLASS = FORMALVARS THEN
                         NWORDS := PTRSIZE
                       ELSE
                         NWORDS := IDTYPE^.SIZE
                     END;
                   FORMAT := BIG
                 END
               ELSE
                 BEGIN LITYPE := PUBLICDEF;
                   BASEOFFSET := VADDR
                 END
             END;
      FIELD: NEEDEDBYLINKER := FALSE;
      PROC,
      FUNC:  BEGIN
               IF PFDECKIND = DECLARED THEN
                 IF PFKIND = ACTUAL THEN
                   IF KLASS = PROC THEN
                     IF EXTURNAL THEN
                       IF SEPPROC THEN LITYPE := SEPPREF
                       ELSE LITYPE := EXTPROC
                     ELSE
                       IF SEPPROC THEN
                         LITYPE := SSEPPROC
                       ELSE NEEDEDBYLINKER := FALSE
                   ELSE (*KLASS = FUNC*)
                     IF EXTURNAL THEN
                       IF SEPPROC THEN LITYPE := SEPFREF
                       ELSE LITYPE := EXTFUNC
                     ELSE
                       IF SEPPROC THEN
                         LITYPE := SSEPFUNC
```

```
                    ELSE NEEDEDBYLINKER := FALSE
                ELSE NEEDEDBYLINKER := FALSE
              ELSE NEEDEDBYLINKER := FALSE;
              IF NEEDEDBYLINKER THEN
                BEGIN
                  LCP := NEXT; NPARAMS := 0;
                  WHILE LCP <> NIL DO
                    BEGIN
                      WITH LCP^ DO
                        IF KLASS = FORMALVARS THEN
                          NPARAMS := NPARAMS + PTRSIZE
                        ELSE
                          IF KLASS = ACTUALVARS THEN
                            IF IDTYPE^.FORM <= POWER THEN
                              NPARAMS := NPARAMS + IDTYPE^.SIZE
                            ELSE NPARAMS := NPARAMS + PTRSIZE;
                      LCP := LCP^.NEXT
                    END;
                  IF LITYPE IN [SEPPREF,SEPFREF] THEN
                    BEGIN FORMAT := BYTE; NWORDS := NPARAMS END
                  ELSE
                    BEGIN PROCNUM := PFNAME; RANGE := NIL END
                END
              END (*PROC,FUNC*);
    MODULE:  BEGIN
                IF NOT INMODULE THEN NEEDEDBYLINKER := FALSE
                ELSE
                  BEGIN LITYPE := MODDULE; NWORDS := 0; FORMAT := BYTE END
              END
  END (*CASE,WITH*);
IF NEEDEDBYLINKER THEN
  IF SEGTABLE[SEG].SEGKIND = 2 (*SEGPROC*) THEN
    WITH LIREC DO
      IF (LITYPE = CONSTDEF) OR (LITYPE = PUBLICDEF) THEN
        NEEDEDBYLINKER := FALSE;
IF NEEDEDBYLINKER THEN
  WITH LIREC DO
    BEGIN LINAME := FCP^.NAME;
      FOR LGTH := 1 TO 8 DO GENBYTE(ORD(LINAME[LGTH]));
      GENWORD(ORD(LITYPE));
      CASE LITYPE OF
        MODDULE,
        PUBBLIC,
        PRIVVATE,
        SEPPREF,SEPFREF: BEGIN
                           GENWORD(ORD(FORMAT));
                           FIC := IC; GENWORD(0);
                           GENWORD(NWORDS);
                           IF LITYPE = MODDULE THEN GETREFS(FCP^.SEGID,1)
                           ELSE
                            IF LITYPE IN [SEPPREF,SEPFREF] THEN
                              GETREFS(-FCP^.PFNAME,1)
                            ELSE GETREFS(FCP^.VADDR + 32,FCP^.IDTYPE^.SIZE);
                         END;
        CONSTDEF: BEGIN  GENWORD(CONSTANT); GENWORD(0); GENWORD(0) END;
        PUBLICDEF: BEGIN GENWORD(BASEOFFSET); GENWORD(0); GENWORD(0) END;
        EXTPROC,EXTFUNC:     BEGIN
                               GENWORD(PROCNUM);
                               GENWORD(NPARAMS);
                               GENWORD(ORD(RANGE))
```

```
                                END;
            SSEPPROC,SSEPFUNC:  BEGIN
                                  GENWORD(PROCNUM);
                                  GENWORD(NPARAMS);
                                  GENWORD(ORD(RANGE));
                                  FOR LGTH := 1 TO 8 DO
                                    GENBYTE(ORD(LINAME[LGTH]));
                                  IF LITYPE = SSEPPROC THEN
                                    GENWORD(ORD(SEPPREF))
                                  ELSE GENWORD(ORD(SEPFREF));
                                  GENWORD(ORD(BYTE));
                                  FIC := IC; GENWORD(0); GENWORD(NPARAMS);
                                  GETREFS(-PROCNUM,1)
                                END
          END(*CASE*)
        END(*WITH*);
    IF IC >= 1024 THEN BEGIN WRITECODE(FALSE); IC := 0 END;

    IF FCP^.LLINK <> NIL THEN GLOBALSEARCH(FCP^.LLINK);
    IF FCP^.RLINK <> NIL THEN GLOBALSEARCH(FCP^.RLINK)

  END (*GLOBALSEARCH*);

BEGIN (*WRITELINKERINFO*)
  IC := 0;
  IF CODEINSEG THEN ERROR(399);
  IF INMODULE THEN
      CURRENTBLOCK := REFBLK;
  IF DECSTUFF THEN (*SKIP IF NO DECLARATIONPART LINKER INFO*)
    BEGIN FCP := DISPLAY[GLEV].FNAME;
      IF FCP <> NIL THEN GLOBALSEARCH(FCP)
    END;
  (*NOW DO NONRESIDENT PROCS*)
  WITH LIREC DO
    FOR I := SEEK TO DECOPS DO
      IF PFNUMOF[I] <> 0 THEN
        BEGIN
          CASE I OF
            SEEK:       BEGIN LINAME := 'FSEEK   '; NPARAMS := 2 END;
            FREADREAL:  BEGIN LINAME := 'FREADREA'; NPARAMS := 2 END;
            FWRITEREAL: BEGIN LINAME := 'FWRITERE'; NPARAMS := 5 END;
            FREADDEC:   BEGIN LINAME := 'FREADDEC'; NPARAMS := 3 END;
            FWRITEDEC:  BEGIN LINAME := 'FWRITEDE'; NPARAMS := 10 END;
            DECOPS:     BEGIN LINAME := 'DECOPS  '; NPARAMS := 0 END;
          END;
          FOR LGTH := 1 TO 8 DO GENBYTE(ORD(LINAME[LGTH]));
          IF SEPPROC THEN
            BEGIN GENWORD(ORD(SEPPREF));
              GENWORD(ORD(BYTE)); FIC := IC; GENWORD(0); GENWORD(NPARAMS);
              GETREFS(-PFNUMOF[I],1)
            END
          ELSE
            BEGIN GENWORD(ORD(EXTPROC));
              GENWORD(PFNUMOF[I]); GENWORD(NPARAMS); GENWORD(0)
            END;
          PFNUMOF[I] := 0;
        END;
  (* NOW DO EOFMARK END-RECORD*)
  FOR LGTH := 1 TO 8 DO GENBYTE(ORD(' '));
  GENWORD(ORD(EOFMARK)); GENWORD(LCMAX);
```

```
  GENWORD(0);GENWORD(0);
  WRITECODE(TRUE);
  CLINKERINFO := FALSE;
  IF DECSTUFF THEN DLINKERINFO := FALSE
END (*WRITELINKERINFO*);

SEGMENT PROCEDURE UNITPART(FSYS: SETOFSYS);
  VAR UMARKP: TESTP;

  PROCEDURE OPENREFFILE;
  BEGIN
    REWRITE(REFFILE,'*SYSTEM.INFO[*]');
    IF IORESULT <> 0 THEN ERROR(402)
  END (* OPENREFFILE *) ;

  PROCEDURE UNITDECLARATION(FSYS: SETOFSYS; VAR UMARKP:TESTP);
    VAR LCP: CTP; FOUND: BOOLEAN; LLEXSTK: LEXSTKREC;
  BEGIN
    IF INMODULE THEN ERROR(182 (* NESTED MODULES NOT ALLOWED *));
    IF CODEINSEG THEN
      BEGIN ERROR(399); SEGINX := 0; CURBYTE := 0 END;
    WITH LLEXSTK DO
      BEGIN
        DOLDTOP := TOP;
        DOLDLEV := LEVEL;
        POLDPROC := CURPROC;
        SOLDPROC := NEXTPROC;
        DOLDSEG := SEG;
        DLLC := LC;
        PREVLEXSTACKP := TOS
      END;
    SEG := NEXTSEG;
    NEXTSEG := NEXTSEG + 1;
    IF NEXTSEG > MAXSEG THEN ERROR(250);
    NEXTPROC := 1;
    PUBLICPROCS := FALSE;
    INMODULE := TRUE;
    INSYMBOL;
    IF SY <> IDENT THEN ERROR(2)
    ELSE
      BEGIN FOUND := FALSE;
        LCP := MODPTR;
        WHILE (LCP <> NIL) AND NOT FOUND DO
          IF LCP^.NAME <> ID THEN LCP := LCP^.NEXT
          ELSE BEGIN FOUND := TRUE; ERROR(101) END;
        IF NOT FOUND THEN
          BEGIN NEW(LCP,MODULE);
            WITH LCP^ DO
              BEGIN NAME := ID; IDTYPE := NIL; NEXT := MODPTR;
                KLASS := MODULE; SEGID := SEG
              END;
            MODPTR := LCP
          END;
      END;
    SEGTABLE[SEG].SEGNAME := ID;
    MARK(UMARKP);
    NEW(REFLIST);
    NEW(TOS);
    TOS^ := LLEXSTK;
    LEVEL := 1;
```

```
    IF TOP < DISPLIMIT THEN
      BEGIN TOP := TOP +1;
        WITH DISPLAY[TOP] DO
          BEGIN FNAME := NIL; FFILE := NIL; FLABEL := NIL; OCCUR := BLCK END;
        IF LCP <> NIL THEN ENTERID(LCP)
      END
    ELSE ERROR(250);
    INSYMBOL;
    IF SY = SEMICOLON THEN INSYMBOL ELSE ERROR(14)
  END (*UNITDECLARATION*) ;

BEGIN (*UNITPART*)
  OPENREFFILE;
  REPEAT
    RESET(REFFILE); NREFS := 1; REFBLK := 0;
    IF (SY = SEPARATSY) THEN
      BEGIN SEPPROC := TRUE;
        INSYMBOL; IF SY <> UNITSY THEN ERROR(24)
      END
    ELSE
      SEPPROC := FALSE;
    UNITDECLARATION(FSYS,UMARKP);
    IF SEPPROC THEN SEGTABLE[SEG].SEGKIND := 4 ELSE SEGTABLE[SEG].SEGKIND := 3;
    SEGTABLE[SEG].TEXTADDR := CURBLK;
    WRITETEXT;
    IF SY = INTERSY THEN INSYMBOL
    ELSE ERROR(22);
    ININTERFACE := TRUE;
    DECLARATIONPART(FSYS);
    IF PUBLICPROCS THEN
      BEGIN
        ININTERFACE := FALSE;
        IF SY <> IMPLESY THEN BEGIN ERROR(23); SKIP(FSYS - STATBEGSYS) END
        ELSE INSYMBOL;
        BLOCK(FSYS - [SEPARATSY,UNITSY,INTERSY,IMPLESY]);
        IF REFBLK > 0 THEN
          IF BLOCKWRITE(REFFILE,REFLIST^,1,REFBLK) <> 1 THEN ERROR(402);
        WRITELINKERINFO(TRUE);
      END
    ELSE
      BEGIN DLINKERINFO := FALSE;
        WITH SEGTABLE[SEG] DO
          BEGIN CODELENG := 0; DISKADDR :=CURBLK; SEGKIND := 0 END;
      END;
    SEPPROC := FALSE; (*FALSE WHENEVER NOT INMODULE*)
    INMODULE := FALSE;
    IF SY = ENDSY THEN INSYMBOL
    ELSE BEGIN ERROR(13); SKIP(FSYS) END;
    IF SY <> PERIOD THEN
      IF SY = SEMICOLON THEN INSYMBOL
      ELSE ERROR(14);
    WITH TOS^ DO
        BEGIN
          TOP := DOLDTOP;
          LEVEL := DOLDLEV;
          CURPROC := POLDPROC;
          NEXTPROC := SOLDPROC;
          SEG := DOLDSEG;
          LC := DLLC;
        END;
```

```
    TOS := TOS^.PREVLEXSTACKP;
    RELEASE(UMARKP)
  UNTIL NOT (SY IN [UNITSY,SEPARATSY]);
  CLOSE(REFFILE)
END (*UNITPART*);

(* $I PROCS.A.TEXT*)

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) l978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

PROCEDURE ERROR(*ERRORNUM: INTEGER*);
  VAR CH: CHAR; ERRSTART: INTEGER;
      A: PACKED ARRAY [0..179] OF CHAR;
BEGIN
  WITH USERINFO DO
    IF (ERRSYM <> SYMCURSOR) OR (ERRBLK <> SYMBLK) THEN
      BEGIN ERRBLK := SYMBLK;
        ERRSYM := SYMCURSOR; ERRNUM := ERRORNUM;
        IF STUPID THEN CH := 'E'
        ELSE
          BEGIN
            IF NOISY THEN WRITELN(OUTPUT)
            ELSE
              IF LIST AND (ERRORNUM <= 400) THEN
                EXIT(ERROR);
            IF LINESTART = 0 THEN
              WRITE(OUTPUT,SYMBUFP^:SYMCURSOR)
            ELSE
              BEGIN
                ERRSTART := SCAN(-(LINESTART-1),=CHR(EOL),
                                 SYMBUFP^[LINESTART-2])+LINESTART-1;
                MOVELEFT(SYMBUFP^[ERRSTART],A[0],SYMCURSOR-ERRSTART);
                WRITE(OUTPUT,A:SYMCURSOR-ERRSTART)
              END;
            WRITELN(OUTPUT,' <<<<');
            WRITE(OUTPUT,'Line ',SCREENDOTS,', error ',ERRORNUM:0,':');
            IF NOISY THEN
              WRITE(OUTPUT,' <sp>(continue), <esc>(terminate), E(dit');
            WRITE(OUTPUT,CHR(7));
            REPEAT READ(KEYBOARD,CH)
            UNTIL (CH = ' ') OR (CH = 'E') OR (CH = 'e') OR (CH = ALTMODE)
          END;
        IF (CH = 'E') OR (CH = 'e') THEN
          BEGIN ERRBLK := SYMBLK-2; EXIT(PASCALCOMPILER) END;
        IF (ERRORNUM > 400) OR (CH = CHR(27)) THEN
          BEGIN ERRBLK := 0; EXIT(PASCALCOMPILER) END;
        WRITELN(OUTPUT);
        IF NOISY THEN
          WRITE(OUTPUT,'<',SCREENDOTS:4,'>')
      END
END (*ERROR*) ;

PROCEDURE GETNEXTPAGE;
```

```
BEGIN SYMCURSOR := 0; LINESTART := 0;
  IF USING THEN
    BEGIN
      IF USEFILE = WORKCODE THEN
        BEGIN
          IF BLOCKREAD(USERINFO.WORKCODE^,SYMBUFP^,2,SYMBLK) <> 2 THEN
            USING := FALSE
        END
      ELSE
        IF USEFILE = SYSLIBRARY THEN
          IF BLOCKREAD(LIBRARY,SYMBUFP^,2,SYMBLK) <> 2 THEN
            USING := FALSE;
      IF NOT USING THEN
        BEGIN
          SYMBLK := PREVSYMBLK; SYMCURSOR := PREVSYMCURSOR;
          LINESTART := PREVLINESTART
        END
    END;
  IF NOT USING THEN
    BEGIN
      IF INCLUDING THEN
        IF BLOCKREAD(INCLFILE,SYMBUFP^,2,SYMBLK) <> 2 THEN
          BEGIN CLOSE(INCLFILE); INCLUDING := FALSE;
            SYMBLK := OLDSYMBLK; SYMCURSOR := OLDSYMCURSOR;
            LINESTART := OLDLINESTART
          END
    END;
  IF NOT (INCLUDING OR USING) THEN
    IF BLOCKREAD(USERINFO.WORKSYM^,SYMBUFP^,2,SYMBLK) <> 2 THEN
      ERROR(401);
  IF SYMCURSOR = 0 THEN
    BEGIN
      IF INMODULE THEN
        IF ININTERFACE AND NOT USING THEN WRITETEXT;
      IF SYMBUFP^[0] = CHR(16(*DLE*)) THEN
       SYMCURSOR := 2
    END;
  SYMBLK := SYMBLK+2
END (*GETNEXTPAGE*) ;

(*$I+*)
PROCEDURE PRINTLINE;
  VAR DORLEV,STARORC: CHAR; LENG: INTEGER;
      A: PACKED ARRAY [0..99] OF CHAR;
BEGIN STARORC := ':';
  IF DP THEN DORLEV := 'D'
  ELSE DORLEV := CHR((BEGSTMTLEV MOD 10) + ORD('0'));
  IF BPTONLINE THEN STARORC := '*';
  WRITE(LP,SCREENDOTS:6,SEG:4,CURPROC:5,
            STARORC,DORLEV,LINEINFO:6,' ');
  LENG := SYMCURSOR-LINESTART;
  IF LENG > 100 THEN LENG := 100;
  MOVELEFT(SYMBUFP^[LINESTART],A,LENG);
  IF A[0] = CHR(16(*DLE*)) THEN
    BEGIN
      IF A[1] > ' ' THEN
        WRITE(LP,' ':ORD(A[1])-ORD(' '));
      LENG := LENG-2;
      MOVELEFT(A[2],A,LENG)
    END;
```

```
  A[LENG-1] := CHR(EOL); (*JUST TO MAKE SURE*)
  WRITE(LP,A:LENG);
  WITH USERINFO DO
    IF (ERRBLK = SYMBLK) AND (ERRSYM > LINESTART) THEN
      WRITELN(LP,'>>>>>> Error # ',ERRNUM)
END (*PRINTLINE*) ;
(*$I-*)

PROCEDURE ENTERID(*FCP: CTP*);
  VAR LCP,LCP1: CTP; I: INTEGER;
BEGIN LCP := DISPLAY[TOP].FNAME;
  IF LCP = NIL THEN DISPLAY[TOP].FNAME := FCP
  ELSE
    BEGIN I := TREESEARCH(LCP,LCP1,FCP^.NAME);
      WHILE I = 0 DO
        BEGIN ERROR(101);
          IF LCP1^.RLINK = NIL THEN I := 1
          ELSE I := TREESEARCH(LCP1^.RLINK,LCP1,FCP^.NAME)
        END;
      IF I = 1 THEN LCP1^.RLINK := FCP ELSE LCP1^.LLINK := FCP
    END;
  FCP^.LLINK := NIL; FCP^.RLINK := NIL
END (*ENTERID*) ;

PROCEDURE INSYMBOL; (* COMPILER VERSION 3.4 06-NOV-76 *)
  LABEL 1;
  VAR LVP: CSP; X: INTEGER;

PROCEDURE CHECKEND;
BEGIN (* CHECKS FOR THE END OF THE PAGE *)
  SCREENDOTS := SCREENDOTS+1;
  SYMCURSOR := SYMCURSOR + 1;
  IF NOISY THEN
    BEGIN WRITE(OUTPUT,'.');
      IF (SCREENDOTS-STARTDOTS) MOD 50 = 0 THEN
        BEGIN WRITELN(OUTPUT);
          WRITE(OUTPUT,'<',SCREENDOTS:4,'>')
        END
    END;
  IF LIST THEN PRINTLINE;
  BPTONLINE := FALSE;
  IF SYMBUFP^[SYMCURSOR]=CHR(0) THEN GETNEXTPAGE
  ELSE LINESTART := SYMCURSOR;
  IF SYMBUFP^[SYMCURSOR] = CHR(12(*FF*)) THEN SYMCURSOR:=SYMCURSOR+1;
  IF SYMBUFP^[SYMCURSOR] = CHR(16(*DLE*)) THEN
    SYMCURSOR := SYMCURSOR+2
  ELSE
    BEGIN
      SYMCURSOR := SYMCURSOR+SCAN(80,<>CHR(9),SYMBUFP^[SYMCURSOR]);
      SYMCURSOR := SYMCURSOR+SCAN(80,<>' ',SYMBUFP^[SYMCURSOR])
    END;
  IF DP THEN LINEINFO := LC ELSE LINEINFO := IC
END;

PROCEDURE COMMENTER(STOPPER: CHAR);
  VAR CH,SW,DEL: CHAR; LTITLE: STRING[40];

  PROCEDURE SCANSTRING(VAR STRG: STRING; MAXLENG: INTEGER);
    VAR LENG: INTEGER;
  BEGIN SYMCURSOR := SYMCURSOR+2;
```

```
    LENG := SCAN(MAXLENG,=STOPPER,SYMBUFP^[SYMCURSOR]);
    STRG[0] := CHR(LENG);
    MOVELEFT(SYMBUFP^[SYMCURSOR],STRG[1],LENG);
    SYMCURSOR := SYMCURSOR+LENG+1
  END (*SCANSTRING*) ;

BEGIN
  SYMCURSOR := SYMCURSOR+1; (* POINT TO THE FIRST CH PAST "(*" *)
  IF SYMBUFP^[SYMCURSOR]='$' THEN
     IF SYMBUFP^[SYMCURSOR+1] <> STOPPER THEN
        REPEAT
          CH := SYMBUFP^[SYMCURSOR+1];
          SW := SYMBUFP^[SYMCURSOR+2];
          DEL := SYMBUFP^[SYMCURSOR+3];
          IF (SW = ',') OR (SW = STOPPER) THEN
            BEGIN DEL := SW; SW := '+';
              SYMCURSOR := SYMCURSOR-1
            END;
          CASE CH OF
          'C': BEGIN
                 IF LEVEL > 1 THEN ERROR(194);
                 NEW(COMMENT); SCANSTRING(COMMENT^,80); EXIT(COMMENTER)
               END;
          'D': DEBUGGING := (SW='+');
          'G': GOTOOK := (SW='+');
          'I': IF (SW='+') OR (SW='-') THEN IOCHECK := (SW='+')
               ELSE
                 BEGIN SCANSTRING(LTITLE,40);
                   IF STOPPER = '*' THEN
                     SYMCURSOR := SYMCURSOR+1;
                   IF LIST THEN
                     BEGIN
                       SYMCURSOR := SYMCURSOR + 1;
                       PRINTLINE;
                       SYMCURSOR := SYMCURSOR - 1;
                     END;
                   IF INCLUDING OR INMODULE AND ININTERFACE THEN
                     BEGIN ERROR(406); EXIT(COMMENTER) END;
                   OPENOLD(INCLFILE,LTITLE);
                   IF IORESULT <> 0 THEN
                     BEGIN OPENOLD(INCLFILE,CONCAT(LTITLE,'.TEXT'));
                        IF IORESULT <> 0 THEN ERROR(403)
                     END;
                   INCLUDING := TRUE;
                   OLDSYMCURSOR := SYMCURSOR;
                   OLDLINESTART := LINESTART;
                   OLDSYMBLK := SYMBLK-2;
                   SYMBLK := 2; GETNEXTPAGE;
                   INSYMBOL; EXIT(INSYMBOL)
                 END;
          'L': IF (SW='+') OR (SW='-') THEN
                 BEGIN LIST := (SW='+');
                   IF LIST THEN OPENNEW(LP,'*SYSTEM.LST.TEXT')
                 END
               ELSE
                 BEGIN SCANSTRING(LTITLE,40);
                   OPENNEW(LP,LTITLE);
                   LIST := IORESULT = 0;
                   EXIT(COMMENTER)
                 END;
```

```
          'Q': NOISY := (SW='-');
          'P': WRITE(LP,CHR(12(*FF*)));
          'R': RANGECHECK := (SW='+');
          'S': NOSWAP:=(SW='-');
          'T': TINY := (SW='+');
          'U': IF (SW='+') OR (SW='-') THEN
                 BEGIN SYSCOMP := (SW = '-');
                   RANGECHECK := NOT SYSCOMP;
                   IOCHECK := RANGECHECK;
                   GOTOOK := SYSCOMP
                 END
               ELSE
                 IF NOT USING THEN
                   BEGIN SCANSTRING(SYSTEMLIB,40);
                     CLOSE(LIBRARY); LIBNOTOPEN := TRUE;
                     EXIT(COMMENTER)
                   END
          END (*CASES*);
          SYMCURSOR := SYMCURSOR+3;
        UNTIL DEL <> ',';
  SYMCURSOR := SYMCURSOR-1; (* ADJUST *)
  REPEAT
    REPEAT
      SYMCURSOR := SYMCURSOR+1;
      WHILE SYMBUFP^[SYMCURSOR] = CHR(EOL) DO CHECKEND
    UNTIL SYMBUFP^[SYMCURSOR]=STOPPER;
  UNTIL (SYMBUFP^[SYMCURSOR+1]=')') OR (STOPPER='}');
  SYMCURSOR := SYMCURSOR+1;
END (*COMMENTER*);

PROCEDURE STRING;
LABEL 1;
VAR
  T: PACKED ARRAY [1..80] OF CHAR;
  TP,NBLANKS,L: INTEGER;
  DUPLE: BOOLEAN;

BEGIN
  DUPLE := FALSE; (* INDICATES WHEN '' IS PRESENT *)
  TP := 0; (* INDEX INTO TEMPORARY STRING *)
  REPEAT
    IF DUPLE THEN SYMCURSOR := SYMCURSOR+1;
    REPEAT
      SYMCURSOR := SYMCURSOR+1;
      TP := TP+1;
      IF SYMBUFP^[SYMCURSOR] = CHR(EOL) THEN
        BEGIN ERROR(202); CHECKEND; GOTO 1 END;
      T[TP] := SYMBUFP^[SYMCURSOR];
    UNTIL SYMBUFP^[SYMCURSOR]='''';
    DUPLE := TRUE;
  UNTIL SYMBUFP^[SYMCURSOR+1]<>'''';
1:  TP := TP-1; (* ADJUST *)
  SY := STRINGCONST; OP := NOOP;
  LGTH := TP; (* GROSS *)
  IF TP=1 (* SINGLE CHARACTER CONSTANT *)
    THEN
      VAL.IVAL := ORD(T[1])
    ELSE
      WITH SCONST^ DO
        BEGIN
```

```
          CCLASS := STRG;
          SLGTH := TP;
          MOVELEFT(T[1],SVAL[1],TP);
          VAL.VALP := SCONST
        END
END(*STRING*);

PROCEDURE NUMBER;
VAR
  EXPONENT,ENDI,ENDF,ENDE,SIGN,IPART,FPART,EPART,
  ISUM:  INTEGER;
  TIPE: (REALTIPE,INTEGERTIPE);
  RSUM: REAL;
  NOTLONG: BOOLEAN;
  K,J: INTEGER;
BEGIN
  (* TAKES A NUMBER AND DECIDES WHETHER IT'S REAL
     OR INTEGER AND CONVERTS IT TO THE INTERNAL
     FORM. *)
  TIPE := INTEGERTIPE;
  ENDI := 0;
  ENDF := 0;
  ENDE := 0;
  SIGN := 1;
  NOTLONG := TRUE;
  EPART := 9999; (* OUT OF REACH *)
  IPART := SYMCURSOR; (* INTEGER PART STARTS HERE *)
  REPEAT
    SYMCURSOR := SYMCURSOR+1
  UNTIL (SYMBUFP^[SYMCURSOR]<'0') OR (SYMBUFP^[SYMCURSOR]>'9');
  (* SYMCURSOR NOW POINTS AT FIRST CHARACTER PAST INTEGER PART *)
  ENDI := SYMCURSOR-1; (* MARK THE END OF IPART *)
  IF SYMBUFP^[SYMCURSOR]='.'
    THEN
      IF SYMBUFP^[SYMCURSOR+1]<>'.'  (* WATCH OUT FOR '..' *)
        THEN
          BEGIN
            TIPE := REALTIPE;
            SYMCURSOR := SYMCURSOR+1;
            FPART := SYMCURSOR; (* BEGINNING OF FPART *)
            WHILE (SYMBUFP^[SYMCURSOR] >= '0') AND
                  (SYMBUFP^[SYMCURSOR] <= '9') DO
              SYMCURSOR := SYMCURSOR+1;
            IF SYMCURSOR = FPART THEN ERROR(201);
            ENDF := SYMCURSOR-1;
          END;
  IF SYMBUFP^[SYMCURSOR]='E'
    THEN
      BEGIN
        TIPE := REALTIPE;
        SYMCURSOR := SYMCURSOR+1;
        IF SYMBUFP^[SYMCURSOR]='-'
          THEN
            BEGIN
              SYMCURSOR := SYMCURSOR+1;
              SIGN := -1;
            END
          ELSE
            IF SYMBUFP^[SYMCURSOR]='+'
              THEN
```

```
            SYMCURSOR := SYMCURSOR+1;
      EPART := SYMCURSOR; (* BEGINNING OF EXPONENT *)
      WHILE (SYMBUFP^[SYMCURSOR]>='0') AND (SYMBUFP^[SYMCURSOR]<='9') DO
        SYMCURSOR := SYMCURSOR+1;
      ENDE := SYMCURSOR-1;
      IF ENDE<EPART THEN ERROR(201); (* ERROR IN REAL CONSTANT *)
    END;
(* NOW CONVERT TO INTERNAL FORM *)
IF TIPE=INTEGERTIPE THEN
  BEGIN
    ISUM := 0;
    FOR J := IPART TO ENDI DO
      BEGIN
        IF (ISUM>MAXINT DIV 10) OR ((ISUM=MAXINT DIV 10) AND
                  (ORD(SYMBUFP^[J]) - ORD('0') > MAXINT MOD 10)) THEN
            BEGIN NOTLONG := FALSE; K := J; J := ENDI END
        ELSE ISUM := ISUM*10+(ORD(SYMBUFP^[J])-ORD('0'));
      END;
      IF NOTLONG THEN
        BEGIN
          SY := INTCONST;  OP := NOOP;
          VAL.IVAL := ISUM;
        END
      ELSE
        BEGIN
          IF ENDI - IPART >= MAXDEC THEN
            BEGIN ERROR(203); IPART := ENDI; K := ENDI END;
          NEW(LVP,LONG);
          WITH LVP^ DO
            BEGIN CCLASS := LONG; J := 4; LLENG := 0;
              WHILE K <= ENDI DO
                BEGIN
                  IF J = 4 THEN
                    BEGIN LLENG := LLENG + 1;
                      LONGVAL[LLENG] := ISUM;
                      ISUM := 0;
                      J := 0
                    END;
                  ISUM := ISUM * 10 + ORD(SYMBUFP^[K])-ORD('0');
                  K := K + 1; J := J + 1
                END;
              LLAST := J;
              IF J > 0 THEN
                BEGIN LLENG := LLENG + 1;
                  LONGVAL[LLENG] := ISUM
                END;
            END;
          SY := LONGCONST; OP := NOOP;
          LGTH := ENDI - IPART + 1;
          VAL.VALP := LVP
        END;
  END (*TIPE = INTEGERTIPE*)
ELSE
  BEGIN (* REAL NUMBER HERE *)
    RSUM := 0;
    FOR J := IPART TO ENDI DO
      BEGIN
        RSUM := RSUM*10+(ORD(SYMBUFP^[J])-ORD('0'));
      END;
    FOR J := ENDF DOWNTO FPART DO
```

```
        RSUM := RSUM+(ORD(SYMBUFP^[J])-ORD('0'))/PWROFTEN(J-FPART+1);
      EXPONENT := 0;
      FOR J := EPART TO ENDE DO
        EXPONENT := EXPONENT*10+ORD(SYMBUFP^[J])-ORD('0');
      IF SIGN=-1 THEN
        RSUM := RSUM/PWROFTEN(EXPONENT)
      ELSE
        RSUM := RSUM*PWROFTEN(EXPONENT);
      SY := REALCONST;  OP := NOOP;
      NEW(LVP,REEL);
      LVP^.CCLASS := REEL;
      LVP^.RVAL := RSUM;
      VAL.VALP := LVP;
    END;
  SYMCURSOR := SYMCURSOR-1; (* ADJUST FOR POSTERITY *)
END (*NUMBER*) ;

BEGIN (* INSYMBOL *)
  IF GETSTMTLEV THEN BEGIN BEGSTMTLEV := STMTLEV; GETSTMTLEV := FALSE END;
  OP := NOOP;
1:  SY := OTHERSY; (* IF NO CASES EXERCISED BLOW UP *)
  CASE SYMBUFP^[SYMCURSOR] OF
  '''':STRING;
  '0','1','2','3','4','5','6','7','8','9':
       NUMBER;
  'A','B','C','D','E','F','G','H','I','J','K','L','M',
  'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
  'a','b','c','d','e','f','g','h','i','j','k','l','m',
  'n','o','p','q','r','s','t','u','v','w','x','y','z':
       IDSEARCH(SYMCURSOR,SYMBUFP^); (* MAGIC PROC *)
  '{': BEGIN COMMENTER('}'); GOTO 1 END;
  '(': BEGIN
         IF SYMBUFP^[SYMCURSOR+1]='*' THEN
             BEGIN
                SYMCURSOR := SYMCURSOR+1;
                COMMENTER('*');
                SYMCURSOR := SYMCURSOR+1;
                GOTO 1; (* GET ANOTHER TOKEN *)
             END
           ELSE
             SY := LPARENT;
       END;
  ')': SY := RPARENT;
  ',': SY := COMMA;
  ' ',' ': BEGIN SYMCURSOR := SYMCURSOR+1; GOTO 1; END;
  '.': BEGIN
         IF SYMBUFP^[SYMCURSOR+1]='.'
           THEN
             BEGIN
               SYMCURSOR := SYMCURSOR+1;
               SY := COLON
             END
           ELSE
             SY := PERIOD;
       END;
  ':': IF SYMBUFP^[SYMCURSOR+1]='='
         THEN
           BEGIN
             SYMCURSOR := SYMCURSOR+1;
             SY := BECOMES;
```

```
          END
        ELSE
           SY := COLON;
  ';': SY := SEMICOLON;
  '^': SY := ARROW;
  '[': SY := LBRACK;
  ']': SY := RBRACK;
  '*': BEGIN SY := MULOP; OP := MUL END;
  '+': BEGIN SY := ADDOP; OP := PLUS END;
  '-': BEGIN SY := ADDOP; OP := MINUS END;
  '/': BEGIN SY := MULOP; OP := RDIV END;
  '<': BEGIN
         SY := RELOP;
         OP := LTOP;
         CASE SYMBUFP^[SYMCURSOR+1] OF
           '>': BEGIN
                  OP := NEOP;
                  SYMCURSOR := SYMCURSOR+1
                END;
           '=': BEGIN
                  OP := LEOP;
                  SYMCURSOR := SYMCURSOR+1
                END
         END;
       END;
  '=': BEGIN SY := RELOP; OP := EQOP END;
  '>': BEGIN
         SY := RELOP;
         IF SYMBUFP^[SYMCURSOR+1]='='
           THEN
             BEGIN
               OP := GEOP;
               SYMCURSOR := SYMCURSOR+1;
             END
           ELSE
             OP := GTOP;
       END
END (* CASE SYMBUFP^[SYMCURSOR] OF *);
  IF SY=OTHERSY THEN
    IF SYMBUFP^[SYMCURSOR] = CHR(EOL) THEN
      BEGIN CHECKEND; GETSTMTLEV := TRUE; GOTO 1 END
    ELSE ERROR(400);
  SYMCURSOR := SYMCURSOR+1; (* NEXT CALL TALKS ABOUT NEXT TOKEN *)
END (*INSYMBOL*) ;

(* $I PROCS.B.TEXT*)

  (*      COPYRIGHT  (C) 1978, REGENTS OF THE      *)
  (*      UNIVERSITY OF CALIFORNIA, SAN DIEGO      *)

  PROCEDURE SEARCHSECTION(*FCP: CTP; VAR FCP1: CTP*);
  BEGIN
    IF FCP <> NIL THEN
      IF TREESEARCH(FCP,FCP1,ID) = 0 THEN (*NADA*)
      ELSE FCP1 := NIL
    ELSE FCP1 := NIL
  END (*SEARCHSECTION*) ;

  PROCEDURE SEARCHID(*FIDCLS: SETOFIDS; VAR FCP: CTP*);
    LABEL 1; VAR LCP: CTP;
```

```
  BEGIN
    FOR DISX := TOP DOWNTO 0 DO
      BEGIN LCP := DISPLAY[DISX].FNAME;
        IF LCP <> NIL THEN
          IF TREESEARCH(LCP,LCP,ID) = 0 THEN
            IF LCP^.KLASS IN FIDCLS THEN GOTO 1
            ELSE
              IF PRTERR THEN ERROR(103)
              ELSE LCP := NIL
          ELSE LCP := NIL
      END;
    IF PRTERR THEN
      BEGIN ERROR(104);
        IF TYPES IN FIDCLS THEN LCP := UTYPPTR
        ELSE
          IF ACTUALVARS IN FIDCLS THEN LCP := UVARPTR
          ELSE
            IF FIELD IN FIDCLS THEN LCP := UFLDPTR
            ELSE
              IF KONST IN FIDCLS THEN LCP := UCSTPTR
              ELSE
                IF PROC IN FIDCLS THEN LCP := UPRCPTR
                ELSE LCP := UFCTPTR
      END;
1:  FCP := LCP
  END (*SEARCHID*) ;

  PROCEDURE GETBOUNDS(*FSP: STP; VAR FMIN,FMAX: INTEGER*);
  BEGIN
    WITH FSP^ DO
      IF FORM = SUBRANGE THEN
        BEGIN FMIN := MIN.IVAL; FMAX := MAX.IVAL END
      ELSE
        BEGIN FMIN := 0;
          IF FSP = CHARPTR THEN FMAX := 255
          ELSE
            IF FSP^.FCONST <> NIL THEN
              FMAX := FSP^.FCONST^.VALUES.IVAL
            ELSE FMAX := 0
        END
  END (*GETBOUNDS*) ;

  PROCEDURE SKIP(*FSYS: SETOFSYS*);
  BEGIN WHILE NOT(SY IN FSYS) DO INSYMBOL
  END (*SKIP*) ;

  FUNCTION PAOFCHAR(*FSP: STP): BOOLEAN*);
  BEGIN PAOFCHAR := FALSE;
    IF FSP <> NIL THEN
      IF FSP^.FORM = ARRAYS THEN
        PAOFCHAR := FSP^.AISPACKD AND (FSP^.AELTYPE = CHARPTR)
  END (*PAOFCHAR*) ;

  FUNCTION STRGTYPE(*FSP: STP) : BOOLEAN*);
  BEGIN STRGTYPE := FALSE;
    IF PAOFCHAR(FSP) THEN STRGTYPE := FSP^.AISSTRNG
  END (*STRGTYPE*) ;

  FUNCTION DECSIZE(*I: INTEGER): INTEGER*);
  BEGIN DECSIZE := (TRUNC(I*3.321) + 1 + BITSPERWD) DIV BITSPERWD
```

```
END (*DECSIZE*) ;
PROCEDURE CONSTANT(*FSYS: SETOFSYS; VAR FSP: STP; VAR FVALU: VALU*);
  VAR LSP: STP; LCP: CTP; SIGN: (NONE,POS,NEG);
      LVP: CSP;
BEGIN LSP := NIL; FVALU.IVAL := 0;
  IF NOT(SY IN CONSTBEGSYS) THEN
    BEGIN ERROR(50); SKIP(FSYS+CONSTBEGSYS) END;
  IF SY IN CONSTBEGSYS THEN
    BEGIN
      IF SY = STRINGCONSTSY THEN
        BEGIN
          IF LGTH = 1 THEN LSP := CHARPTR
          ELSE
            BEGIN
              NEW(LSP,ARRAYS,TRUE,TRUE);
              LSP^ := STRGPTR^;
              LSP^.MAXLENG := LGTH;
              LSP^.INXTYPE := NIL;
              NEW(LVP);
              LVP^ := VAL.VALP^;
              VAL.VALP := LVP
            END;
          FVALU := VAL; INSYMBOL
        END
      ELSE
        BEGIN
          SIGN := NONE;
          IF (SY = ADDOP) AND (OP IN [PLUS,MINUS]) THEN
            BEGIN IF OP = PLUS THEN SIGN := POS ELSE SIGN := NEG;
              INSYMBOL
            END;
          IF SY = IDENT THEN
            BEGIN SEARCHID([KONST],LCP);
              WITH LCP^ DO
                BEGIN LSP := IDTYPE; FVALU := VALUES END;
              IF SIGN <> NONE THEN
                IF LSP = INTPTR THEN
                  BEGIN IF SIGN = NEG THEN
                    FVALU.IVAL := -FVALU.IVAL END
                ELSE
                  IF LSP = REALPTR THEN
                    BEGIN
                      IF SIGN = NEG THEN
                        BEGIN NEW(LVP,REEL);
                          LVP^.CCLASS := REEL;
                          LVP^.RVAL := -FVALU.VALP^.RVAL;
                          FVALU.VALP := LVP;
                        END
                    END
                  ELSE
                    IF COMPTYPES(LSP,LONGINTPTR) THEN
                      BEGIN
                        IF SIGN = NEG THEN
                          BEGIN NEW(LVP,LONG);
                            LVP^.CCLASS := LONG;
                            LVP^.LONGVAL[1] := - FVALU.VALP^.LONGVAL[1];
                            FVALU.VALP := LVP
                          END
                      END
                    ELSE ERROR(105);
```

```
            INSYMBOL;
          END
        ELSE
          IF SY = INTCONST THEN
            BEGIN IF SIGN = NEG THEN VAL.IVAL := -VAL.IVAL;
              LSP := INTPTR; FVALU := VAL; INSYMBOL
            END
          ELSE
            IF SY = REALCONST THEN
              BEGIN IF SIGN = NEG THEN
                      VAL.VALP^.RVAL := -VAL.VALP^.RVAL;
                LSP := REALPTR; FVALU := VAL; INSYMBOL
              END
            ELSE
              IF SY = LONGCONST THEN
                BEGIN
                  IF SIGN = NEG THEN
                    BEGIN VAL.VALP^.LONGVAL[1] := - VAL.VALP^.LONGVAL[1];
                      NEW(LSP,LONGINT);
                      LSP^.SIZE := DECSIZE(LGTH);
                      LSP^.FORM := LONGINT;
                      FVALU := VAL;
                      INSYMBOL
                    END
                END
              ELSE
                BEGIN ERROR(106); SKIP(FSYS) END
      END;
    IF NOT (SY IN FSYS) THEN
      BEGIN ERROR(6); SKIP(FSYS) END
    END;
  FSP := LSP
END (*CONSTANT*) ;

FUNCTION COMPTYPES(*FSP1,FSP2: STP) : BOOLEAN*);
  VAR NXT1,NXT2: CTP; COMP: BOOLEAN;
    LTESTP1,LTESTP2 : TESTP;
BEGIN
  IF FSP1 = FSP2 THEN COMPTYPES := TRUE
  ELSE
    IF (FSP1 = NIL) OR (FSP2 = NIL) THEN COMPTYPES := TRUE
    ELSE
      IF FSP1^.FORM = FSP2^.FORM THEN
        CASE FSP1^.FORM OF
          SCALAR:
            COMPTYPES := FALSE;
          SUBRANGE:
            COMPTYPES := COMPTYPES(FSP1^.RANGETYPE,
                                   FSP2^.RANGETYPE);
          POINTER:
              BEGIN
                COMP := FALSE; LTESTP1 := GLOBTESTP;
                LTESTP2 := GLOBTESTP;
                WHILE LTESTP1 <> NIL DO
                  WITH LTESTP1^ DO
                    BEGIN
                      IF (ELT1 = FSP1^.ELTYPE) AND
                        (ELT2 = FSP2^.ELTYPE) THEN COMP := TRUE;
                      LTESTP1 := LASTTESTP
                    END;
```

```
                IF NOT COMP THEN
                  BEGIN NEW(LTESTP1);
                    WITH LTESTP1^ DO
                      BEGIN ELT1 := FSP1^.ELTYPE;
                        ELT2 := FSP2^.ELTYPE;
                        LASTTESTP := GLOBTESTP
                      END;
                    GLOBTESTP := LTESTP1;
                    COMP := COMPTYPES(FSP1^.ELTYPE,FSP2^.ELTYPE)
                  END;
                COMPTYPES := COMP; GLOBTESTP := LTESTP2
              END;
            LONGINT: COMPTYPES := TRUE;
            POWER:
              COMPTYPES := COMPTYPES(FSP1^.ELSET,FSP2^.ELSET);
            ARRAYS:
              BEGIN
                COMP := COMPTYPES(FSP1^.AELTYPE,FSP2^.AELTYPE)
                        AND (FSP1^.AISPACKD = FSP2^.AISPACKD);
                IF COMP AND FSP1^.AISPACKD THEN
                    COMP := (FSP1^.ELSPERWD = FSP2^.ELSPERWD)
                            AND (FSP1^.ELWIDTH = FSP2^.ELWIDTH)
                            AND (FSP1^.AISSTRNG = FSP2^.AISSTRNG);
                IF COMP AND NOT STRGTYPE(FSP1) THEN
                  COMP := (FSP1^.SIZE = FSP2^.SIZE);
                COMPTYPES := COMP;
              END;
            RECORDS:
              BEGIN NXT1 := FSP1^.FSTFLD; NXT2 := FSP2^.FSTFLD;
                COMP := TRUE;
                WHILE (NXT1 <> NIL) AND (NXT2 <> NIL) AND COMP DO
                  BEGIN COMP:=COMPTYPES(NXT1^.IDTYPE,NXT2^.IDTYPE);
                    NXT1 := NXT1^.NEXT; NXT2 := NXT2^.NEXT
                  END;
                COMPTYPES := COMP AND (NXT1 = NIL) AND (NXT2 = NIL)
                            AND (FSP1^.RECVAR = NIL)
                            AND (FSP2^.RECVAR = NIL)
              END;
            FILES:
              COMPTYPES := COMPTYPES(FSP1^.FILTYPE,FSP2^.FILTYPE)
          END (*CASE*)
        ELSE (*FSP1^.FORM <> FSP2^.FORM*)
          IF FSP1^.FORM = SUBRANGE THEN
            COMPTYPES := COMPTYPES(FSP1^.RANGETYPE,FSP2)
          ELSE
            IF FSP2^.FORM = SUBRANGE THEN
              COMPTYPES := COMPTYPES(FSP1,FSP2^.RANGETYPE)
            ELSE COMPTYPES := FALSE
END (*COMPTYPES*) ;


PROCEDURE GENBYTE(*FBYTE: INTEGER*);
BEGIN
  CODEP^[IC] := CHR(FBYTE); IC := IC+1
END (*GENBYTE*) ;

PROCEDURE GENWORD(*FWORD: INTEGER*);
BEGIN
  IF ODD(IC) THEN IC := IC + 1;
  MOVELEFT(FWORD,CODEP^[IC],2);
```

```
    IC := IC + 2
  END (*GENWORD*) ;

PROCEDURE WRITETEXT;
  BEGIN
    MOVELEFT(SYMBUFP^[SYMCURSOR],CODEP^[0],1024);
    IF USERINFO.ERRNUM = 0 THEN
      IF BLOCKWRITE(USERINFO.WORKCODE^,CODEP^[0],2,CURBLK) <> 2 THEN
        ERROR(402);
    CURBLK := CURBLK + 2
  END (*WRITETEXT*) ;

  PROCEDURE WRITECODE(*FORCEBUF: BOOLEAN*);
    VAR CODEINX,LIC,I: INTEGER;
  BEGIN CODEINX := 0; LIC := IC;
    REPEAT
      I := 512-CURBYTE;
      IF I > LIC THEN I := LIC;
      MOVELEFT(CODEP^[CODEINX],DISKBUF[CURBYTE],I);
      CODEINX := CODEINX+I;
      CURBYTE := CURBYTE+I;
      IF (CURBYTE = 512) OR FORCEBUF THEN
        BEGIN
          IF USERINFO.ERRNUM = 0 THEN
            IF BLOCKWRITE(USERINFO.WORKCODE^,DISKBUF,1,CURBLK) <> 1 THEN
              ERROR(402);
          CURBLK := CURBLK+1; CURBYTE := 0
        END;
      LIC := LIC-I
    UNTIL LIC = 0;
  END (*WRITECODE*) ;

  PROCEDURE FINISHSEG;
    VAR I: INTEGER;
  BEGIN IC := 0;
    FOR I := NEXTPROC-1 DOWNTO 1 DO
      IF PROCTABLE[I] = 0 THEN
        GENWORD(0)
      ELSE
        GENWORD(SEGINX+IC-PROCTABLE[I]);
    GENBYTE(SEG); GENBYTE(NEXTPROC-1);
    SEGTABLE[SEG].CODELENG := SEGINX+IC;
    WRITECODE(TRUE); SEGINX := 0; CODEINSEG := FALSE
  END (*FINISHSEG*) ;

(* $I BLOCK.TEXT*)

PROCEDURE BLOCK(*FSYS: SETOFSYS*);
LABEL 1;
VAR BFSYFOUND: BOOLEAN;

  PROCEDURE FINDFORW(FCP: CTP);
    BEGIN
      IF FCP <> NIL THEN
        WITH FCP^ DO
          BEGIN
            IF KLASS IN [PROC,FUNC] THEN
              IF PFDECKIND = DECLARED THEN
                IF PFKIND = ACTUAL THEN
                  IF FORWDECL THEN
```

```
                BEGIN
                  USERINFO.ERRNUM := 117; WRITELN(OUTPUT);
                  WRITE(OUTPUT,NAME,' undefined')
                END;
          FINDFORW(RLINK); FINDFORW(LLINK)
        END
  END (*FINDFORW*) ;

BEGIN (*BLOCK*)
   IF (NOSWAP) AND (STARTINGUP) THEN
     BEGIN
       BODYPART(FSYS,NIL);
       EXIT(BLOCK);
     END;
   IF (SY IN [UNITSY,SEPARATSY]) AND (NOT INMODULE) THEN
     BEGIN
       UNITPART(FSYS + [UNITSY,INTERSY,IMPLESY,ENDSY]);
       IF SY = PERIOD THEN EXIT(BLOCK)
     END;
   NEWBLOCK:=TRUE;
   REPEAT
     IF NOT NEWBLOCK THEN
       BEGIN
         DP := FALSE; STMTLEV := 0; IC := 0; LINEINFO := 0;
         IF (NOT SYSCOMP) OR (LEVEL>1) THEN FINDFORW(DISPLAY[TOP].FNAME);
         IF INMODULE THEN
           IF TOS^.PREVLEXSTACKP^.DFPROCP = OUTERBLOCK THEN
             IF (SY = ENDSY) THEN
               BEGIN FINISHSEG; EXIT(BLOCK) END
             ELSE IF (SY = BEGINSY) THEN
               BEGIN ERROR(13); FINISHSEG; EXIT(BLOCK) END;
         IF SY = BEGINSY THEN INSYMBOL ELSE ERROR(17);
         REPEAT
           BODYPART(FSYS + [CASESY] - [ENDSY], TOS^.DFPROCP);
           BFSYFOUND := (SY = TOS^.BFSY) OR (INMODULE AND (SY = ENDSY));
           IF NOT BFSYFOUND THEN
             BEGIN
               IF TOS^.BFSY = SEMICOLON THEN
                 ERROR(14)  (*SEMICOLON EXPECTED*)
               ELSE  ERROR(6);  (* PERIOD EXPECTED *)
               SKIP(FSYS + [TOS^.BFSY]);
               BFSYFOUND := (SY = TOS^.BFSY) OR (INMODULE AND (SY = ENDSY))
             END
         UNTIL (BFSYFOUND) OR (SY IN BLOCKBEGSYS);
         IF NOT BFSYFOUND THEN
           BEGIN
             IF TOS^.BFSY = SEMICOLON THEN ERROR(14)
             ELSE ERROR(6); (*PERIOD EXPECTED*)
             DECLARATIONPART(FSYS);
           END
         ELSE
           BEGIN
             IF SY = SEMICOLON THEN INSYMBOL;
             IF (NOT(SY IN [BEGINSY,PROCSY,FUNCSY,PROGSY])) AND
                (TOS^.BFSY = SEMICOLON) THEN
               IF NOT (INMODULE AND (SY = ENDSY)) THEN
                 BEGIN
                   ERROR(6); SKIP(FSYS);
                   DECLARATIONPART(FSYS);
                 END
```

```
                ELSE GOTO 1
              ELSE
        1:      BEGIN
                  WITH TOS^ DO
                    BEGIN
                      IF DFPROCP <> NIL THEN
                        DFPROCP^.INSCOPE:=FALSE;
                      IF ISSEGMENT THEN
                        BEGIN
                          IF CODEINSEG THEN FINISHSEG;
                          IF DLINKERINFO AND (LEVEL = 1) THEN
                            BEGIN SEGTABLE[SEG].SEGKIND := 2;
                              WRITELINKERINFO(TRUE)
                            END
                          ELSE
                            IF CLINKERINFO THEN
                              BEGIN SEGTABLE[SEG].SEGKIND := 2;
                                WRITELINKERINFO(FALSE)
                              END;
                          NEXTPROC:=SOLDPROC;
                          SEG:=DOLDSEG;
                        END;
                      LEVEL:=DOLDLEV;
                      TOP:=DOLDTOP;
                      LC:=DLLC;
                      CURPROC:=POLDPROC;
                    END;
                  RELEASE(TOS^.DMARKP);
                  TOS:=TOS^.PREVLEXSTACKP;
                  NEWBLOCK:=(SY IN [PROCSY,FUNCSY,PROGSY]);
                END
            END
        END
      ELSE
        BEGIN DECLARATIONPART(FSYS);
          IF LEVEL = 0 THEN
            IF SY IN [UNITSY,SEPARATSY] THEN
              BEGIN
                UNITPART(FSYS + [UNITSY,INTERSY,IMPLESY,ENDSY]);
                IF SY IN [PROCSY,FUNCSY,PROGSY] THEN DECLARATIONPART(FSYS)
              END
        END;
    UNTIL TOS = NIL;
    FINISHSEG;
 END (*BLOCK*) ;

BEGIN (* PASCALCOMPILER *)
  COMPINIT;
  TIME(LGTH,LOWTIME);
  BLOCK(BLOCKBEGSYS+STATBEGSYS-[CASESY]);
  IF SY <> PERIOD THEN ERROR(21);
  IF LIST THEN
    BEGIN SCREENDOTS := SCREENDOTS+1;
      SYMBUFP^[SYMCURSOR] := CHR(EOL);
      SYMCURSOR := SYMCURSOR+1;
      PRINTLINE
    END;
  USERINFO.ERRBLK := 0;
  TIME(LGTH,STARTDOTS); LOWTIME := STARTDOTS-LOWTIME;
  UNITWRITE(3,IC,7);
```

```
    IF DLINKERINFO OR CLINKERINFO THEN
      BEGIN SEGTABLE[SEG].SEGKIND := 1;
        WRITELINKERINFO(TRUE)
      END;
    CLOSE(LP,LOCK);
    IF NOISY THEN WRITELN(OUTPUT);
    WRITE(OUTPUT,SCREENDOTS,' lines');
    IF LOWTIME > 0 THEN
      WRITE(OUTPUT,', ',(LOWTIME+30) DIV 60,' secs, ',
          ROUND((3600/LOWTIME)*SCREENDOTS),' lines/min');
    IF NOISY THEN
      BEGIN
        WRITELN(OUTPUT);
        WRITE(OUTPUT,'Smallest available space = ',SMALLESTSPACE,' words');
      END;
    IC := 0;
    FOR SEG := 0 TO MAXSEG DO
      WITH SEGTABLE[SEG] DO
        BEGIN GENWORD(DISKADDR); GENWORD(CODELENG) END;
    FOR SEG := 0 TO MAXSEG DO
      WITH SEGTABLE[SEG] DO
        FOR LGTH := 1 TO 8 DO
          GENBYTE(ORD(SEGNAME[LGTH]));
    FOR SEG := 0 TO MAXSEG DO GENWORD(SEGTABLE[SEG].SEGKIND);
    FOR SEG := 0 TO MAXSEG DO GENWORD(SEGTABLE[SEG].TEXTADDR);
    FOR LGTH := 1 TO 80 DO
      IF COMMENT <> NIL THEN GENBYTE(ORD(COMMENT^[LGTH])) ELSE GENBYTE(0);
    FOR LGTH := 1 TO 256 - 8*(MAXSEG + 1) - 40 DO GENWORD(0);
    CURBLK := 0; CURBYTE := 0; WRITECODE(TRUE)
  END (* PASCALCOMPILER *) ;

  BEGIN (* SYSTEM *)
  END.

{ +-----------------------------------------------------------------+
  |                                                                 |
  |                     F     I     N     I     S                   |
  |                                                                 |
  +-----------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 Compiler
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 Disassembler
##########################################################################

(*$S+*)
PROGRAM CODESTAT;

{=========================================================}
{                                                         }
{        UCSD     P-CODE     DISASSEMBLER                 }
{                                                         }
{        Release level:      I.5    Sept, 1978           }
{                                                         }
{        Written by     William  P.  Franks               }
{                                                         }
{        Institute for Information Systems                }
{        UC  San Diego,  La Jolla,  Ca                    }
{                                                         }
{        Kenneth  L.  Bowles,  Director                   }
{                                                         }
{        COPYRIGHT  (C)  1978,  Regents of the            }
{        University of California,  San Diego             }
{                                                         }
{=========================================================}

CONST   MAXPROCNUM=150;

TYPE    NMENONIC=PACKED ARRAY[0..7] OF CHAR;
        BYTETYPE=ARRAY[0..7] OF INTEGER;
        WORDTYPE=ARRAY[0..15] OF INTEGER;
        BYTE=0..255;
        OPTYPE=(SHORT,ONE,OPT,TWO,LOPT,WORDS,CHRS,BLK,CMPRSS,CMPRSS2,WORD);
        OPREC=RECORD CASE OPTYPE OF
                 SHORT:(TOTAL0:INTEGER);
          ONE,CHRS,BLK:(TOTAL1:INTEGER;
                        BYTEONE1:BYTETYPE);
                   TWO:(TOTAL2:INTEGER;
                        BYTEONE2:BYTETYPE;
                        BYTETWO2:BYTETYPE;
                        FLAVOR2:ARRAY[2..29] OF INTEGER);
              WORD,OPT:(TOTAL3:INTEGER;
                        PARMONE3:WORDTYPE);
                  LOPT:(TOTAL4:INTEGER;
                        BYTEONE4:BYTETYPE;
                        PARMTWO4:WORDTYPE);
                 WORDS:(TOTAL5:INTEGER;
                        PARMONE5:WORDTYPE;
                        PARMTWO5:WORDTYPE;
                        PARMTHREE5:WORDTYPE);
                CMPRSS:(TOTAL6:INTEGER;
                        FLAVOR6:ARRAY[0..40] OF INTEGER);
               CMPRSS2:(TOTAL7:INTEGER;
                        FLAVOR7:ARRAY[1..6] OF INTEGER)
          END;
        OPPTR=^OPREC;
        OPFACTS=RECORD
          NAMES:ARRAY[52..255] OF NMENONIC;
          RECTYPES:ARRAY[0..255] OF OPTYPE
        END;
```

```
        JUMPREC=RECORD
          POS,NEG:WORDTYPE
        END;
        PRCLARRY=ARRAY[0..MAXPROCNUM] OF INTEGER;
        DSPTR=^DSARRY;
        DSARRY=ARRAY[0..1] OF INTEGER;
        HEXTYPE=PACKED RECORD CASE INTEGER OF
            0:(DUM2,DUM1,HI,LO:0..15);
            1:(HIBYTE,LOWBYTE:0..255);
            2:(WORD:INTEGER)
          END;

VAR     DISPLAY:BOOLEAN;
        CH,CR:CHAR;
        PCTMAX,MAXOP,INUM,BYTESIZE,BYTEPOS,OP,BUFSTART,PROCNUM,SEGNUM:INTEGER;
        BITE:BYTE;
        DSSTART:DSPTR;
        SWAP,CONTROL,CONSOLE,DONEPROC,LEXCHECK,DATAWATCH,
        LEXLOOK   :BOOLEAN;
        HEXCOUNT,MAXPROC,SEGSTBLK,BUFSTBLK,OPTOTAL,
        SEGSIZE,OFFSET,BACKJUMP,SLDC,
        SLDL,SLDO,SIND,PROCSTART,DATASEG,DATAPROC,
        DATASEGSIZE,LEXLEVEL,DATAREF,DTSGSZ,JUMPTOTAL   :INTEGER;
        HEX:HEXTYPE;
        RNUM:REAL;
        OPCODE:ARRAY[0..255] OF OPPTR;
        LISTFILE:INTERACTIVE;
        HEXCHAR,CODE   :PACKED ARRAY[0..15] OF CHAR;
        INPUTFILE:FILE;
        JUMPSTATS:JUMPREC;
        SEGLEX:ARRAY[0..15] OF INTEGER;
        SEGDIREC:PACKED ARRAY[0..511] OF BYTE;
        NAMES:ARRAY[52..255] OF NMENONIC;
        RECTYPES:PACKED ARRAY[0..255] OF OPTYPE;
        PROCS:ARRAY [0..MAXPROCNUM] OF INTEGER;
        PROCCALL:ARRAY[0..15] OF ^PRCLARRY;
        JUMPS,PROCLEX:ARRAY[0..99] OF INTEGER;
        LASTFILENAME:STRING;
        BUFFER:PACKED ARRAY[0..2559] OF BYTE;

SEGMENT PROCEDURE INIT;
VAR   I:INTEGER;
      FILENAME:STRING;
      OPFILE:FILE OF OPFACTS;

PROCEDURE NEWOP(FLAVOR:OPTYPE);
BEGIN
  CASE FLAVOR OF
        SHORT:NEW(OPCODE[I],SHORT);
          ONE:NEW(OPCODE[I],ONE);
          BLK:NEW(OPCODE[I],BLK);
         CHRS:NEW(OPCODE[I],CHRS);
          OPT:NEW(OPCODE[I],OPT);
          TWO:NEW(OPCODE[I],TWO);
         LOPT:NEW(OPCODE[I],LOPT);
        WORDS:NEW(OPCODE[I],WORDS);
       CMPRSS:NEW(OPCODE[I],CMPRSS);
      CMPRSS2:NEW(OPCODE[I],CMPRSS2);
         WORD:NEW(OPCODE[I],WORD)
    END;
```

```
  WITH OPCODE[I]^ DO
    CASE FLAVOR OF
             SHORT:TOTAL0:=0;
      CHRS,BLK,ONE:BEGIN
                     TOTAL1:=0;
                     FILLCHAR(BYTEONE1,16,0);
                   END;
               TWO:BEGIN
                     TOTAL2:=0;
                     FILLCHAR(BYTEONE2,16,0);
                     FILLCHAR(BYTETWO2,16,0);
                     FILLCHAR(FLAVOR2,56,0);
                   END;
          WORD,OPT:BEGIN
                     TOTAL3:=0;
                     FILLCHAR(PARMONE3,32,0);
                   END;
              LOPT:BEGIN
                     TOTAL4:=0;
                     FILLCHAR(BYTEONE4,16,0);
                     FILLCHAR(PARMTWO4,32,0);
                   END;
             WORDS:BEGIN
                     TOTAL5:=0;
                     FILLCHAR(PARMONE5,32,0);
                     FILLCHAR(PARMTWO5,32,0);
                     FILLCHAR(PARMTHREE5,32,0);
                   END;
            CMPRSS:BEGIN
                     TOTAL6:=0;
                     FILLCHAR(FLAVOR6,82,0);
                   END;
           CMPRSS2:BEGIN
                     TOTAL7:=0;
                     FILLCHAR(FLAVOR7,12,0);
                   END
    END;
END;

BEGIN(* INIT *)
  CR:=CHR(13);
  RESET(OPFILE,'*OPCODES.I5');
  NAMES:=OPFILE^.NAMES;
  FOR I:=0 TO 255 DO
    BEGIN
      NEWOP(OPFILE^.RECTYPES[I]);
      RECTYPES[I]:=OPFILE^.RECTYPES[I];
    END;
  CLOSE(OPFILE);
  PAGE(OUTPUT);
  GOTOXY(22,10);
  WRITELN('UCSD   P-CODE   DISASSEMBLER');
  GOTOXY(0,0);
  WRITE('Input code file: ');
  READLN(FILENAME);
  (*$I-*)
  OPENOLD(INPUTFILE,CONCAT(FILENAME,'.CODE'));
  (*$I+*)
  IF IORESULT <> 0 THEN
     OPENOLD(INPUTFILE,FILENAME);
```

```
  IF BLOCKREAD(INPUTFILE,SEGDIREC,1)=1  THEN ;
  FOR SEGNUM:=0 TO 15 DO
    IF SEGDIREC[SEGNUM*4] + SEGDIREC[SEGNUM*4 + 1]<>0 THEN
      BEGIN
        NEW(PROCCALL[SEGNUM]);
        FILLCHAR(PROCCALL[SEGNUM]^,SIZEOF(PRCLARRY),0);
      END
    ELSE PROCCALL[SEGNUM]:=NIL;
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITELN(' ':10,'Is this code file designed for a machine');
  WRITE(' ':7,'where byte zero is the most significant byte <terak no>?');
  READ(KEYBOARD,CH);
  SWAP:=(CH='Y');
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITE('Dis-assembly output file (<CR> for none): ');
  READLN(FILENAME);
  LASTFILENAME:=FILENAME;
  DISPLAY:=(FILENAME<>'');
  CONSOLE:=(FILENAME='CONSOLE:') OR (FILENAME='#1:');
  IF DISPLAY THEN REWRITE(LISTFILE,FILENAME);
  SEGNUM:=0;
  OPTOTAL:=0;
  SLDC:=0;
  SLDL:=0;
  SLDO:=0;
  SIND:=0;
  JUMPTOTAL:=0;
  HEXCOUNT:=0;
  CODE:='                ';
  HEXCHAR:='0123456789ABCDEF';
  FILLCHAR(JUMPSTATS.POS,32,0);
  FILLCHAR(JUMPSTATS.NEG,32,0);
  LEXLOOK:=FALSE;
END;

PROCEDURE PROMPT; FORWARD;

SEGMENT PROCEDURE DISASSEMBLE;

FUNCTION BUFRESET(BYTEPOS,OFFSET,DIRECTION:INTEGER):INTEGER;
VAR   NEWBYTE:INTEGER;
BEGIN
  NEWBYTE:=BYTEPOS + OFFSET;
  REPEAT
    BUFSTBLK:=BUFSTBLK + DIRECTION;
    BUFSTART:=(BUFSTBLK - SEGSTBLK)*512;
  UNTIL (NEWBYTE - BUFSTART>=0) AND (NEWBYTE - BUFSTART<2557);
  IF BLOCKREAD(INPUTFILE,BUFFER,5,BUFSTBLK)=1 THEN;
  BUFRESET:=NEWBYTE - BUFSTART;
END;

FUNCTION LASTBYTE:BYTE;
VAR   CHANGE:INTEGER;
BEGIN
  IF BYTEPOS<1 THEN
    BEGIN
      BYTEPOS:=BUFRESET(BUFSTART + BYTEPOS,-1,-1);
      OFFSET:=OFFSET - 1;
```

```
    END
  ELSE
    BEGIN
      BYTEPOS:=BYTEPOS - 1;
      OFFSET:=OFFSET - 1;
    END;
  LASTBYTE:=BUFFER[BYTEPOS];
END;

FUNCTION GETBYTE:BYTE;
VAR  HEX:HEXTYPE;
BEGIN
  IF BYTEPOS>2559 THEN
    BYTEPOS:=BUFRESET(BUFSTART + BYTEPOS,0,5);
  GETBYTE:=BUFFER[BYTEPOS];
  IF HEXCOUNT<15 THEN
    BEGIN
      HEX.LOWBYTE:=BUFFER[BYTEPOS];
      CODE[HEXCOUNT]:=HEXCHAR[HEX.HI];
      CODE[HEXCOUNT + 1]:=HEXCHAR[HEX.LO];
      HEXCOUNT:=HEXCOUNT + 2;
    END;
  BYTEPOS:=BYTEPOS + 1;
END;

FUNCTION GETBIG:INTEGER;
VAR  BIG:HEXTYPE;
     FIRSTBYTE:BYTE;
BEGIN
  FIRSTBYTE:=GETBYTE;
  IF FIRSTBYTE>127 THEN
    BEGIN
      BIG.LOWBYTE:=GETBYTE;
      BIG.HIBYTE:=FIRSTBYTE - 128;
      GETBIG:=BIG.WORD;
    END
  ELSE GETBIG:=FIRSTBYTE;
END;

FUNCTION GETWORD:INTEGER;
VAR  WERD:HEXTYPE;
BEGIN
  IF SWAP THEN
    BEGIN
      WERD.HIBYTE:=GETBYTE;
      WERD.LOWBYTE:=GETBYTE;
    END
  ELSE
    BEGIN
      WERD.LOWBYTE:=GETBYTE;
      WERD.HIBYTE:=GETBYTE;
    END;
  GETWORD:=WERD.WORD;
END;

FUNCTION MOSTSIGBIT(OPERAND:INTEGER):INTEGER;
VAR  BYTESIZE:INTEGER;
BEGIN
  IF OPERAND<0 THEN
    MOSTSIGBIT:=15
```

```
  ELSE
    BEGIN
      BYTESIZE:=-1;
      REPEAT
        BYTESIZE:=BYTESIZE + 1;
        OPERAND:=OPERAND DIV 2;
      UNTIL OPERAND=0;
      MOSTSIGBIT:=BYTESIZE;
    END;
END;

PROCEDURE ACTACCESS(FINALEX,OFFSET:INTEGER); FORWARD;

PROCEDURE SHORTOP;
{SLDC ABI  ABR  ADI  ADR  LAND DIF  DVI  DVR  CHK  FLO  FLT  INN  INT
 LOR  MODI MPI  MPR  NGI  NGR  LNOT SRS  SBI  SBR  SGS  SQI  SQR  STO
 IXS  UNI  S2P  LDCN LDP  STP  LDB  STB  EQUI GEQI GTRI LEQI LESI NEQI
 S1P  IXB  BYT  XIT  SLDL SLDO SIND}

BEGIN
  OPCODE[BITE]^.TOTAL0:=OPCODE[BITE]^.TOTAL0 + 1;
  IF BITE=214 THEN DONEPROC:=TRUE;
  IF BITE<128 THEN
    BEGIN
      SLDC:=SLDC + 1;
      IF DISPLAY THEN WRITELN(LISTFILE,NAMES[127],BITE:6,' ':18,CODE);
    END
  ELSE
    BEGIN
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      IF BITE>215 THEN
        IF BITE<232 THEN
          BEGIN
            SLDL:=SLDL + 1;
            IF DATAWATCH THEN ACTACCESS(LEXLEVEL,BITE - 215);
            IF DISPLAY THEN WRITELN(LISTFILE,BITE-215:6,' ':18,CODE);
          END
        ELSE IF BITE<248 THEN
          BEGIN
            SLDO:=SLDO + 1;
            IF DATAWATCH THEN ACTACCESS(0,BITE - 231);
            IF DISPLAY THEN WRITELN(LISTFILE,BITE-231:6,' ':18,CODE);
          END
        ELSE
          BEGIN
            SIND:=SIND + 1;
            IF DISPLAY THEN WRITELN(LISTFILE,BITE-248:6,' ':18,CODE);
          END
      ELSE
        IF DISPLAY THEN WRITELN(LISTFILE,' ':24,CODE);
    END;
  IF DONEPROC THEN
    IF DISPLAY THEN WRITELN(LISTFILE);
END;

PROCEDURE ONEOP;
{ADJ  FJP  SAS  RNP  CIP  UJP  LDM  STM  RBP  CBP  CLP  CGP  EFJ  NFJ}

VAR   JUMPSIZE:INTEGER;
      PCALL:BOOLEAN;
```

```
PROCEDURE JUMPOPST;
VAR   NEG:BOOLEAN;
BEGIN
  NEG:=(JUMPSIZE<0);
  IF NEG THEN JUMPSIZE:=-JUMPSIZE;
  BYTESIZE:=-1;
  REPEAT
    BYTESIZE:=BYTESIZE + 1;
    JUMPSIZE:=JUMPSIZE DIV 2;
  UNTIL JUMPSIZE=0;
  IF NEG THEN
    JUMPSTATS.NEG[BYTESIZE]:=JUMPSTATS.NEG[BYTESIZE] + 1
  ELSE
    JUMPSTATS.POS[BYTESIZE]:=JUMPSTATS.POS[BYTESIZE] + 1;
END;

BEGIN(* ONEOP *)
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL1:=TOTAL1 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      IF (BITE=173) OR (BITE=193) THEN DONEPROC:=TRUE;
      IF (BITE IN [161,185,211,212]) THEN
        BEGIN
          BITE:=GETBYTE;
          IF BITE<128 THEN
            BEGIN
              JUMPTOTAL:=JUMPTOTAL + 1;
              JUMPSIZE:=BITE;
              JUMPOPST;
              IF DISPLAY THEN WRITELN(LISTFILE,
                    BUFSTART + BYTEPOS + BITE - PROCSTART:6,' ':18,CODE);
            END
          ELSE
            BEGIN
              JUMPTOTAL:=JUMPTOTAL + 1;
              JUMPSIZE:=JUMPS[(256-BITE-8)DIV 2] - (BUFSTART+BYTEPOS-PROCSTART);
              JUMPOPST;
              IF DISPLAY THEN WRITELN(LISTFILE,
                              JUMPS[(256 - BITE - 8) DIV 2]:6,' ':18,CODE);
            END;
        END
      ELSE
        BEGIN
          PCALL:=(BITE IN [174,206,207]);
          BITE:=GETBYTE;
          IF PCALL THEN
            PROCCALL[SEGNUM]^[BITE]:=PROCCALL[SEGNUM]^[BITE] + 1;
          IF DISPLAY THEN WRITELN(LISTFILE,BITE:6,' ':18,CODE);
          IF DONEPROC THEN
            IF DISPLAY THEN WRITELN(LISTFILE);
        END;
      BYTESIZE:=MOSTSIGBIT(BITE);
      BYTEONE1[BYTESIZE]:=BYTEONE1[BYTESIZE] + 1;
    END;
END;

PROCEDURE OPTOP;
{INC  IND  IXA  LAO  LDO  MOV  MVB  SRO  LLA  LDL  STL  BTP}
```

```
VAR   BIG:INTEGER;
      LOCAL,GLOBAL:BOOLEAN;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL3:=TOTAL3 + 1;
      IF DATAWATCH THEN
        BEGIN
          LOCAL:=(BITE IN [198,202,204]);
          GLOBAL:=(BITE IN [165,167,171]);
        END;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BIG:=GETBIG;
      BYTESIZE:=MOSTSIGBIT(BIG);
      PARMONE3[BYTESIZE]:=PARMONE3[BYTESIZE] + 1;
      IF DATAWATCH THEN
        IF LOCAL THEN ACTACCESS(LEXLEVEL,BIG)
        ELSE IF GLOBAL THEN ACTACCESS(0,BIG);
      IF DISPLAY THEN WRITELN(LISTFILE,BIG:6,' ':18,CODE);
    END;
END;

PROCEDURE LOPTOP;
{LDA  LOD  STR}
VAR   BIG,LINKS:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL4:=TOTAL4 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BITE:=GETBYTE;
      IF DISPLAY THEN WRITE(LISTFILE,BITE:6);
      LINKS:=BITE;
      BYTESIZE:=MOSTSIGBIT(BITE);
      BYTEONE4[BYTESIZE]:=BYTEONE4[BYTESIZE] + 1;
      BIG:=GETBIG;
      BYTESIZE:=MOSTSIGBIT(BIG);
      PARMTWO4[BYTESIZE]:=PARMTWO4[BYTESIZE] + 1;
      IF DATAWATCH THEN ACTACCESS(LEXLEVEL - LINKS,BIG);
      IF DISPLAY THEN WRITELN(LISTFILE,BIG:6,' ':12,CODE);
    END;
END;

PROCEDURE TWOOP;
{IXP  CXP}
VAR   BYTEONE,BYTETWO:BYTE;
      EXTPR:BOOLEAN;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL2:=TOTAL2+ 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      IF BITE=205 THEN EXTPR:=TRUE ELSE EXTPR:=FALSE;
      BYTEONE:=GETBYTE;
      BYTESIZE:=MOSTSIGBIT(BYTEONE);
      BYTEONE2[BYTESIZE]:=BYTEONE2[BYTESIZE] + 1;
      BYTETWO:=GETBYTE;
      DONEPROC:=(EXTPR) AND (BYTEONE=0) AND (BYTETWO=2);
      IF (EXTPR) AND (BYTEONE=0) AND (BYTETWO>1) AND (BYTETWO<30) THEN
        BEGIN
```

```
          FLAVOR2[BYTETWO]:=FLAVOR2[BYTETWO] + 1;
          IF DISPLAY THEN WRITELN(LISTFILE,NAMES[56 + BYTETWO],' ':16,CODE);
        END
      ELSE
        BEGIN
          IF EXTPR THEN
            PROCCALL[BYTEONE]^[BYTETWO]:=PROCCALL[BYTEONE]^[BYTETWO] + 1;
          IF DISPLAY THEN WRITELN(LISTFILE,BYTEONE:6,BYTETWO:6,' ':12,CODE);
        END;
      BYTESIZE:=MOSTSIGBIT(BYTETWO);
      BYTETWO2[BYTESIZE]:=BYTETWO2[BYTESIZE] + 1;
    END;
END;


PROCEDURE WORDOP;
{  LCI  }
VAR   WERD:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL3:=TOTAL3+ 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      WERD:=GETWORD;
      IF DISPLAY THEN WRITELN(LISTFILE,WERD:6,' ':18,CODE);
      BYTESIZE:=MOSTSIGBIT(WERD);
      PARMONE3[BYTESIZE]:=PARMONE3[BYTESIZE] + 1;
    END;
END;

PROCEDURE WORDSOP;
{  XJP  }
VAR   WORD1,WORD2,WORD3:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL5:=TOTAL5 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      IF ODD(BYTEPOS) THEN BITE:=GETBYTE;
      WORD1:=GETWORD;
      BYTESIZE:=MOSTSIGBIT(WORD1);
      PARMONE5[BYTESIZE]:=PARMONE5[BYTESIZE] + 1;
      WORD2:=GETWORD;
      BYTESIZE:=MOSTSIGBIT(WORD2);
      PARMTWO5[BYTESIZE]:=PARMTWO5[BYTESIZE] + 1;
      BYTESIZE:=MOSTSIGBIT(WORD2-WORD1+1);
      PARMTHREE5[BYTESIZE]:=PARMTHREE5[BYTESIZE] + 1;
      BITE:=GETBYTE;   BITE:=GETBYTE;
      IF BITE<128 THEN
        WORD3:=BUFSTART + BYTEPOS + BITE - PROCSTART
      ELSE
        WORD3:=JUMPS[(256 - BITE - 8) DIV 2];
      IF DISPLAY THEN WRITELN(LISTFILE,WORD1:6,WORD2:6,WORD3:6,' ':6,CODE);
      WORD2:=WORD2 - WORD1 + 1;
      FOR WORD1:=1 TO WORD2 DO
        BEGIN
          HEXCOUNT:=0;
          CODE:='                ';
          WORD3:=GETWORD;
          WORD3:=BUFSTART + BYTEPOS - WORD3 - 2 - PROCSTART;
```

```
          IF DISPLAY THEN WRITELN(LISTFILE,WORD3:41,' ':18,CODE);
        END;
    END;
END;

PROCEDURE CMPRSSOP;
{  CSP  }
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL6:=TOTAL6 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BITE:=GETBYTE;
      IF DISPLAY THEN WRITELN(LISTFILE,NAMES[86 + BITE],' ':16,CODE);
      FLAVOR6[BITE]:=FLAVOR6[BITE] + 1;
    END;
END;


PROCEDURE CMPRSS2OP;
{EQU  GEQ  GTR  LEQ  LES  NEQ}
VAR   BIG:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL7:=TOTAL7 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BITE:=GETBYTE;
      FLAVOR7[BITE DIV 2]:=FLAVOR7[BITE DIV 2] +1;
      IF (BITE=10) OR (BITE=12) THEN BIG:=GETBIG;
      IF DISPLAY THEN
        CASE BITE OF
           2:WRITELN(LISTFILE,'REAL',' ':20,CODE);
           4:WRITELN(LISTFILE,'STR ',' ':20,CODE);
           6:WRITELN(LISTFILE,'BOOL',' ':20,CODE);
           8:WRITELN(LISTFILE,'POWR',' ':20,CODE);
          10:WRITELN(LISTFILE,'BYTE',BIG:6,' ':14,CODE);
          12:WRITELN(LISTFILE,'WORD',BIG:6,' ':14,CODE)
        END;
    END;
END;

PROCEDURE CHRSOP;
{  LCA  }
VAR  SKIPOVER,I:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL1:=TOTAL1 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BITE:=GETBYTE;
      IF DISPLAY THEN WRITE(LISTFILE,BITE:6,'   ''');
      BYTESIZE:=MOSTSIGBIT(BITE);
      BYTEONE1[BYTESIZE]:=BYTEONE1[BYTESIZE] + 1;
      IF DISPLAY THEN
        FOR I:=1 TO BITE DO WRITE(LISTFILE,CHR(GETBYTE))
      ELSE
        FOR I:=1 TO BITE DO SKIPOVER:=GETBYTE;
      IF DISPLAY THEN WRITELN(LISTFILE,'''');
    END;
```

```
END;


PROCEDURE BLKOP;
{  LDC  }
VAR  WERD,I,SKIPOVER:INTEGER;
BEGIN
  WITH OPCODE[BITE]^ DO
    BEGIN
      TOTAL1:=TOTAL1 + 1;
      IF DISPLAY THEN WRITE(LISTFILE,NAMES[BITE]);
      BITE:=GETBYTE;
      IF DISPLAY THEN WRITELN(LISTFILE,BITE:6,' ':18,CODE);
      BYTESIZE:=MOSTSIGBIT(BITE);
      BYTEONE1[BYTESIZE]:=BYTEONE1[BYTESIZE] + 1;
      IF ODD(BYTEPOS) THEN SKIPOVER:=GETBYTE;
      FOR I:=1 TO BITE DO
      BEGIN
        HEXCOUNT:=0;
        CODE:='                ';
        WERD:=GETWORD;
        IF DISPLAY THEN WRITELN(LISTFILE,WERD:41,' ':18,CODE);
      END;
    END;
END;

(* $I DISASM1.TEXT *)
                        {start of DISASM1.TEXT}
        {Copyright (c) Regents of University of California at San Diego}

PROCEDURE PROCEJUR;
VAR HEX:HEXTYPE;
    LINENUM,LPROCNUM:INTEGER;

PROCEDURE JUMPINFO;
VAR  OTHERBYTE:INTEGER;
BEGIN
  BACKJUMP:=0; BYTEPOS:=BYTEPOS - 6; OFFSET:=OFFSET - 6;
  REPEAT
    BACKJUMP:=BACKJUMP + 1;
    OTHERBYTE:=LASTBYTE;
    BITE:=LASTBYTE;
    IF (SWAP) AND (BITE<128) THEN {jumps relative to start of segment}
      JUMPS[BACKJUMP]:=BUFSTART + BYTEPOS - BITE*256 - OTHERBYTE
    ELSE IF (NOT SWAP) THEN
      IF OTHERBYTE<128 THEN
        JUMPS[BACKJUMP]:=BUFSTART + BYTEPOS - BITE - OTHERBYTE*256
      ELSE BITE:=OTHERBYTE;
  UNTIL (BITE>127) OR (BACKJUMP=99);
  JUMPS[0]:=BACKJUMP - 1;
  IF BYTEPOS - OFFSET<0 THEN
      BYTEPOS:=BUFRESET(BUFSTART + BYTEPOS,-OFFSET,-1)
    ELSE
      BYTEPOS:=BYTEPOS - OFFSET;
  PROCSTART:=BUFSTART + BYTEPOS; {jumps now relative to start of procedure}
  FOR BACKJUMP:=1 TO JUMPS[0] DO JUMPS[BACKJUMP]:=JUMPS[BACKJUMP] - PROCSTART;
END;

BEGIN (*PROCEJUR*)
  IF PROCS[PROCNUM]=0 THEN
```

```
    WRITELN('Procedure not in file')
  ELSE
    BEGIN
      BYTEPOS:=SEGSIZE - BUFSTART - 2*(PROCNUM + 1) - PROCS[PROCNUM] - 2;
      IF BYTEPOS<0 THEN
        BYTEPOS:=BUFRESET(SEGSIZE - 2*(PROCNUM + 1),-PROCS[PROCNUM] - 2,-1)
      ELSE IF BYTEPOS>2556 THEN
        BYTEPOS:=BUFRESET(BUFSTART + BYTEPOS,0,1);
      OFFSET:=GETWORD;                  { pointer to ENTER IC }
      LPROCNUM:=GETBYTE;
      LEXLEVEL:=GETBYTE;
      BYTEPOS:=BYTEPOS - 4;
      IF LEXLEVEL=255 THEN LEXLEVEL:=-1;
      IF NOT (LEXCHECK OR LEXLOOK) THEN
        IF LPROCNUM=0 THEN
          WRITELN('Procedure ',PROCNUM:3,' is written in Assembly.')
        ELSE
        BEGIN
          JUMPINFO;
          DONEPROC:=FALSE;
          IF DISPLAY THEN WRITELN(LISTFILE,
                ' ':10,'BLOCK #',BYTEPOS DIV 512 + BUFSTBLK:3,
                '     OFFSET IN BLOCK=',BYTEPOS MOD 512:3,CR,
                'SEGMENT PROC     OFFSET#',' ':35,'HEX CODE')
            ELSE IF NOT CONTROL THEN
              BEGIN
                WRITE('.');
                IF PROCNUM=50 THEN WRITE(CR,'    ');
              END
            ELSE WRITE(CR,'[',PROCNUM:2,']');
          LINENUM:=0;
          REPEAT
            HEX.WORD:=BUFSTART + BYTEPOS - PROCSTART;
            IF DISPLAY THEN WRITE(LISTFILE,SEGNUM:7,PROCNUM:5,HEX.WORD:6,'(',
                HEXCHAR[HEX.DUM1],HEXCHAR[HEX.HI],HEXCHAR[HEX.LO],'):   ');
            IF CONTROL AND NOT CONSOLE THEN
              BEGIN
                WRITE('.');
                LINENUM:=LINENUM + 1;
                IF (LINENUM MOD 50=0) THEN WRITE(CR,'    ');
              END;
            HEXCOUNT:=0;
            CODE:='                ';
            BITE:=GETBYTE;
            OPTOTAL:=OPTOTAL + 1;
            CASE RECTYPES[BITE] OF
                    SHORT:SHORTOP;
                   CMPRSS:CMPRSSOP;
                  CMPRSS2:CMPRSS2OP;
                      ONE:ONEOP;
                     CHRS:CHRSOP;
                      BLK:BLKOP;
                      OPT:OPTOP;
                     LOPT:LOPTOP;
                      TWO:TWOOP;
                    WORDS:WORDSOP;
                     WORD:WORDOP
              END;
          UNTIL DONEPROC;
        END;
```

```
    END;
END;

PROCEDURE ALLPROCS;
VAR I,J,MAXDIST,INDEX:INTEGER;
    SORTNUMS:ARRAY[0..MAXPROCNUM] OF INTEGER;
    SORTPROCS:ARRAY[0..MAXPROCNUM] OF BYTE;
BEGIN
  IF DISPLAY THEN
    BEGIN
      SORTNUMS:=PROCS;
      FOR I:=1 TO MAXPROCNUM DO SORTPROCS[I]:=I;
      FOR I:=1 TO PROCS[0] DO
        BEGIN
          MAXDIST:=0;
          INDEX:=0;
          FOR J:=I TO PROCS[0] DO
            IF SORTNUMS[J]>=MAXDIST THEN
              BEGIN
                MAXDIST:=SORTNUMS[J];
                INDEX:=J;
              END;
          SORTNUMS[INDEX]:=SORTNUMS[I];
          SORTNUMS[I]:=SORTPROCS[INDEX];
          SORTPROCS[INDEX]:=SORTPROCS[I];
        END;
      FOR I:=1 TO PROCS[0] DO
        BEGIN
          PROCNUM:=SORTNUMS[I];
          PROCEJUR;
        END;
    END
  ELSE FOR PROCNUM:=1 TO PROCS[0] DO PROCEJUR;
END;

PROCEDURE SEGMINT;
BEGIN
  IF SWAP THEN
    BEGIN
      SEGSTBLK:=SEGDIREC[SEGNUM*4 + 1];
      SEGSIZE:=SEGDIREC[SEGNUM*4 + 3] + SEGDIREC[SEGNUM*4 + 2]*256;
    END
  ELSE
    BEGIN
      SEGSTBLK:=SEGDIREC[SEGNUM*4];
      SEGSIZE:=SEGDIREC[SEGNUM*4 + 3]*256 + SEGDIREC[SEGNUM*4 + 2];
    END;
  BUFSTBLK:=SEGSTBLK;
  IF SEGSIZE>2560 THEN
      BYTEPOS:=BUFRESET(SEGSIZE,-1,1)
    ELSE
      BYTEPOS:=BUFRESET(SEGSIZE,-1,0);
  PROCS[0]:=BUFFER[BYTEPOS];  (* number of procs in segment *)
  BYTEPOS:=BYTEPOS - 2*PROCS[0] - 1;
  FOR PROCNUM:=PROCS[0] DOWNTO 1 DO PROCS[PROCNUM]:=GETWORD;
  IF NOT (CONTROL OR LEXCHECK) THEN ALLPROCS;
END;

PROCEDURE ACTACCESS; {FINALEX,OFFSET:INTEGER;}
VAR  FINALPROC,FINALSEG:INTEGER;
```

```
     INSIDE:BOOLEAN;
BEGIN
  IF (FINALEX=PROCLEX[DATAPROC]) AND (PROCNUM>=DATAPROC) THEN
    IF SEGNUM=DATASEG THEN
      BEGIN
        INSIDE:=(PROCNUM=DATAPROC);
        FINALPROC:=PROCNUM;
        WHILE PROCLEX[FINALPROC]>PROCLEX[DATAPROC] DO FINALPROC:=FINALPROC - 1;
        IF FINALPROC=DATAPROC THEN
          {$R-}
          DSSTART^[OFFSET]:=DSSTART^[OFFSET] + 1;
          {$R+}
      END
    ELSE IF (DATAPROC=1) AND (SEGNUM>DATASEG) THEN
      BEGIN
        FINALSEG:=SEGNUM;
        WHILE SEGLEX[FINALSEG]>SEGLEX[DATASEG] DO FINALSEG:=FINALSEG - 1;
        IF FINALSEG=DATASEG THEN
          {$R-}
          DSSTART^[OFFSET]:=DSSTART^[OFFSET] + 1;
          {$R+}
      END;
END;

PROCEDURE PROCGUIDE;
TYPE  SPACEPTR=^SPACE;
      SPACE=ARRAY[0..19] OF INTEGER;
VAR   I,J:INTEGER;
      DSSPACE:SPACEPTR;

PROCEDURE DATASEGINFO;
VAR  TEMP:INTEGER;
BEGIN
  PROCEJUR;
  BYTEPOS:=BYTEPOS - 2;
  IF SWAP THEN
    BEGIN
      DTSGSZ:=LASTBYTE;
      DTSGSZ:=DTSGSZ + LASTBYTE*256;
      TEMP:=LASTBYTE;
      DTSGSZ:=DTSGSZ + LASTBYTE*256 + TEMP;
    END
  ELSE
    BEGIN
      DTSGSZ:=LASTBYTE*256;
      DTSGSZ:=DTSGSZ + LASTBYTE;
      TEMP:=LASTBYTE*256;
      DTSGSZ:=DTSGSZ + LASTBYTE + TEMP;
    END;
  DTSGSZ:=DTSGSZ DIV 2;
END;

PROCEDURE PROCLOOK;
BEGIN
  GOTOXY(0,3); WRITE(' ':50); GOTOXY(0,3);
  LEXLOOK:=TRUE;
  I:=(PROCS[0] DIV 5) + 1;
  FOR J:=0 TO ((PROCS[0]-1) DIV I) DO WRITE('    #  LL  SIZE');
  WRITELN;
  FOR PROCNUM:=1 TO PROCS[0] DO
```

```
    BEGIN
      DATASEGINFO;
      GOTOXY(15*((PROCNUM-1) DIV I),5+((PROCNUM-1) MOD I));
      WRITE(PROCNUM:5,':',LEXLEVEL:3,DTSGSZ:6);
    END;
  FOR J:=1 TO (5 - (PROCS[0] MOD 5)) DO WRITELN;
  PROMPT;
  LEXLOOK:=FALSE;
END;

BEGIN {PROCGUIDE}
  SEGMINT;
  REPEAT
    PAGE(OUTPUT);
    WRITE('Procedure guide:  #(of procedure),');
    IF LEXCHECK THEN
      WRITELN('L(isting),Q(uit)')
    ELSE
      WRITELN('A(ll),L(isting),Q(uit)');
    WRITE('  to segment: ');
    FOR I:=1 TO 8 DO WRITE(CHR(SEGDIREC[63 + SEGNUM*8 + I]));
    PROCNUM:=0;
    WRITE(CR,CR,'which procedure ');
    IF LEXCHECK THEN
      WRITE('data segment to watch?')
    ELSE
      WRITE('to dis-assemble?');
    READ(CH);
    IF (CH='L') THEN
        PROCLOOK
    ELSE IF (CH='A') AND (NOT LEXCHECK) THEN
      BEGIN
        PAGE(OUTPUT);
        WRITELN('dis-assembling all',PROCS[0]:3,' procedures',CR,CR);
        IF NOT DISPLAY THEN WRITE(CR,CR,'(',SEGNUM:2,')');
        ALLPROCS;
        PROMPT;
        CH:='Q';
      END
    ELSE IF (CH>='0') AND (CH<='9') THEN
      BEGIN
        PROCNUM:=ORD(CH)-ORD('0');
        READ(CH);
        IF (CH>='0') AND (CH<='9') THEN
          PROCNUM:=PROCNUM*10 + ORD(CH) - ORD('0');
        IF (PROCNUM<1) OR (PROCNUM>PROCS[0]) THEN
          BEGIN
            WRITELN(CR,'I didn''t say you had THAT procedure!');
            PROMPT;
          END
        ELSE IF NOT LEXCHECK THEN
          BEGIN
            PAGE(OUTPUT);
            WRITELN('dis-assembling procedure',PROCNUM:3,CR);
            PROCEJUR;
            PROMPT;
            CH:=' ';
          END
        ELSE
          BEGIN
```

```
            DATAPROC:=PROCNUM;
            DATASEG:=SEGNUM;
            DATASEGINFO;
            DATASEGSIZE:=DTSGSZ;
            NEW(DSSTART);
            FOR I:=1 TO ((DATASEGSIZE+19) DIV 20) DO NEW(DSSPACE);
            FILLCHAR(DSSTART^,DATASEGSIZE*2,0);
            FOR PROCNUM:=1 TO PROCS[0] DO
              BEGIN
                PROCEJUR;
                PROCLEX[PROCNUM]:=LEXLEVEL;
              END;
            CH:=CHR(7);
          END;
      END;
  UNTIL (CH='Q') OR (CH=CHR(7));
END;

PROCEDURE SEGMTGUIDE;
VAR I,J:INTEGER;
BEGIN
  REPEAT
    PAGE(OUTPUT);
    WRITELN('Segment guide:  #(of segment),Q(uit)');
    WRITELN(CR,CR,'you have these segments:');
    FOR I:=0 TO 15 DO
      BEGIN
        WRITE(I:4,'      ');
        FOR J:=1 TO 8 DO WRITE(CHR(SEGDIREC[63 + I*8 + J]));
        WRITELN;
      END;
    WRITE(CR,'which segment to look at ');
    IF LEXCHECK THEN
      WRITE('to decide on DATA SEGMENT?')
    ELSE
      WRITE('for possible DIS-ASSEMBLY?');
    READ(CH);
    IF CH<>'Q' THEN
      BEGIN
        SEGNUM:=0;
        IF (CH>='0') AND (CH<='9') THEN SEGNUM:=ORD(CH)-ORD('0');
        READ(CH);
        IF (CH>='0') AND (CH<='9') THEN
          SEGNUM:=SEGNUM*10 + ORD(CH) - ORD('0');
        IF (SEGDIREC[4*SEGNUM] + SEGDIREC[4*SEGNUM + 1]=0) OR (SEGNUM>15) THEN
          BEGIN
            WRITELN(CR,'I didn''t say you had THAT segment!');
            READ(KEYBOARD,CH);
          END
        ELSE
          BEGIN
            PROCGUIDE;
            IF CH<>CHR(7) THEN CH:='A';
          END;
      END;
  UNTIL (CH='Q') OR (CH=CHR(7));
END;

PROCEDURE LEXGUIDE;
BEGIN
```

```
  LEXCHECK:=TRUE;
  DATASEG:=-1;
  REPEAT
    SEGMTGUIDE;
    IF CH='Q' THEN
      BEGIN
        PAGE(OUTPUT);
        GOTOXY(0,10);
        WRITELN('have you changed your mind about data segment watching?');
        READ(KEYBOARD,CH);
        IF CH='Y' THEN DATAWATCH:=FALSE;
      END;
  UNTIL (CH=CHR(7)) OR (NOT DATAWATCH);
  IF DATAWATCH THEN
    FOR SEGNUM:=0 TO 15 DO
      IF SEGDIREC[4*SEGNUM] + SEGDIREC[4*SEGNUM + 1]<>0 THEN
        BEGIN
          SEGMINT; {Sets up appropiate segment}
          PROCNUM:=1;
          PROCEJUR; {Sets up procedure to determine segment's lexlevel}
          SEGLEX[SEGNUM]:=LEXLEVEL;
        END
      ELSE SEGLEX[SEGNUM]:=100;
  PAGE(OUTPUT);
  LEXCHECK:=FALSE;
END;

BEGIN (* SEGMENT DISASSEMBLE *)
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITE('          Do you wish to keep track of references',CR,
        '          to a particular procedure''s data segment?');
  READ(KEYBOARD,CH);
  DATAWATCH:=(CH='Y');
  IF DATAWATCH THEN LEXGUIDE ELSE LEXCHECK:=FALSE;
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITE('Do you wish control over dis-assembly?');
  READ(KEYBOARD,CH);
  CONTROL:=(CH='Y');
  IF CONTROL THEN
    BEGIN
      PAGE(OUTPUT);
      GOTOXY(0,7);
      WRITE(CHR(7));
      WRITE('***  WARNING  - -  STATISTICS ARE GATHERED ON DIS-ASSEMBLED');
      WRITELN(' PROCEDURES ONLY  ***');
      IF DATAWATCH THEN WRITELN(CR,CR,'              ',
                        '***   THIS INCLUDES DATA SEGMENT WATCHING   ***');
      READ(KEYBOARD,CH);
      SEGMTGUIDE;
    END
  ELSE
    BEGIN
      IF NOT CONSOLE THEN WRITE(CHR(12),CR);
      FOR SEGNUM:=0 TO 15 DO
        BEGIN
          IF NOT DISPLAY THEN WRITE(CR,'(',SEGNUM:2,')');
          IF SEGDIREC[4*SEGNUM] + SEGDIREC[4*SEGNUM + 1]<>0 THEN SEGMINT;
        END;
```

```
      PROMPT;
    END;
END;
(* $I DISASM2.TEXT*)

                        {start of DISASM2.TEXT}
        {Copyright (c) Regents of University of California at San Diego}

SEGMENT PROCEDURE GATHER;
VAR   FILENAME:STRING;

PROCEDURE WRITEHDR(VAR H:INTERACTIVE;HEADER:INTEGER);
BEGIN
  CASE HEADER OF
    1: WRITELN(H,'         Parameter one');
    2: WRITELN(H,'Bits used    Total    Percentage');
    3: WRITELN(H,'         Parameter one           Parameter two         ');
    4: WRITELN(H,'Bits used    Total    Percentage    Total    Percentage');
    5: WRITELN(H,'         Parameter one           Parameter two',
                                              '           Case table size');
    6: WRITELN(H,'Bits used    Total    Percentage    Total    Percentage',
                                                  '    Total    Percentage');
    7: WRITELN(H,'Flavor      Total    Percentage     Flavor',
                                                  '      Total    Percentage');
    8: WRITELN(H,'   #   Total    Pct   #   Total    Pct   #   Total',
                                                  '    Pct   #   Total    Pct')
  END;
END;

PROCEDURE JUMPSTUFF;
VAR   I:INTEGER;
BEGIN
  WRITELN(LISTFILE,CR,'Jump statistics on the',JUMPTOTAL:5,' Total jumps');
  IF JUMPTOTAL>0 THEN
    BEGIN
      WRITELN(LISTFILE,CR,
                '                  Positive jumps         Negative jumps');
      WRITEHDR(LISTFILE,4);
      WITH JUMPSTATS DO
        FOR I:=0 TO 15 DO
          WRITELN(LISTFILE,I:5,POS[I]:13,POS[I]/JUMPTOTAL*100:14:2,
                          NEG[I]:9,NEG[I]/JUMPTOTAL*100:14:2);
    END
  ELSE WRITELN(LISTFILE,CR,'Sorry no jumps today!');
END;

PROCEDURE PROCSTUFF;
VAR   I,J:INTEGER;
BEGIN
  WRITELN(LISTFILE,CR,'Procedure call statistics');
  FOR I:=0 TO 15 DO
    IF PROCCALL[I]<>NIL THEN
      FOR J:=1 TO MAXPROCNUM DO
        IF PROCCALL[I]^[J]>0 THEN
          WRITELN(LISTFILE,'  Segment:',I:4,'  Procedure:',J:4,
                                          '  Calls:',PROCCALL[I]^[J]:4);
END;

PROCEDURE SHORTSTUFF;
VAR  I:INTEGER;
```

```
 PROCEDURE SHORT1(VAR H:INTERACTIVE);
 BEGIN
   PCTMAX:=ROUND(SLDC/MAXOP*20);
   WRITE(H,CR,'SLDC   OPCODE: 0..127   TOTAL:',
             SLDC:8,SLDC/OPTOTAL*100:16:2,' % ');
   FOR I:=1 TO PCTMAX DO WRITE(H,'*');
   IF SLDC<>0 THEN
     BEGIN
       WRITELN(H,CR); WRITEHDR(H,8);
       FOR OP:=0 TO 31 DO
         WRITELN(H,OP:4,':',OPCODE[OP]^.TOTAL0:7,OPCODE[OP]^.TOTAL0/SLDC*100:7:2,
          OP+32:4,':',OPCODE[OP+32]^.TOTAL0:7,OPCODE[OP+32]^.TOTAL0/SLDC*100:7:2,
          OP+64:4,':',OPCODE[OP+64]^.TOTAL0:7,OPCODE[OP+64]^.TOTAL0/SLDC*100:7:2,
          OP+96:4,':',OPCODE[OP+96]^.TOTAL0:7,OPCODE[OP+96]^.TOTAL0/SLDC*100:7:2);
     END;
   PCTMAX:=ROUND(SLDL/MAXOP*20);
   WRITE(H,CR,CR,'SLDL  OPCODE: 216..231  TOTAL:',
                 SLDL:8,SLDL/OPTOTAL*100:16:2,' % ');
   FOR I:=1 TO PCTMAX DO WRITE(H,'*');
   IF SLDL<>0 THEN
     BEGIN
       WRITELN(H,CR); WRITEHDR(H,8);
       FOR OP:=216 TO 219 DO
         WRITELN(H,OP:4,':',OPCODE[OP]^.TOTAL0:7,OPCODE[OP]^.TOTAL0/SLDL*100:7:2,
          OP+4:4,':',OPCODE[OP+4]^.TOTAL0:7,OPCODE[OP+4]^.TOTAL0/SLDL*100:7:2,
          OP+8:4,':',OPCODE[OP+8]^.TOTAL0:7,OPCODE[OP+8]^.TOTAL0/SLDL*100:7:2,
          OP+12:4,':',OPCODE[OP+12]^.TOTAL0:7,OPCODE[OP+12]^.TOTAL0/SLDL*100:7:2);
     END;
 END;

 PROCEDURE SHORT2(VAR H:INTERACTIVE);
 BEGIN
   PCTMAX:=ROUND(SLDO/MAXOP*20);
   WRITE(H,CR,CR,'SLDO  OPCODE: 232..247  TOTAL:',
                 SLDO:8,SLDO/OPTOTAL*100:16:2,' % ');
   FOR I:=1 TO PCTMAX DO WRITE(H,'*');
   IF SLDO<>0 THEN
     BEGIN
       WRITELN(H,CR); WRITEHDR(H,8);
       FOR OP:=232 TO 235 DO
         WRITELN(H,OP:4,':',OPCODE[OP]^.TOTAL0:7,OPCODE[OP]^.TOTAL0/SLDO*100:7:2,
            OP+4:4,':',OPCODE[OP+4]^.TOTAL0:7,OPCODE[OP+4]^.TOTAL0/SLDO*100:7:2,
            OP+8:4,':',OPCODE[OP+8]^.TOTAL0:7,OPCODE[OP+8]^.TOTAL0/SLDO*100:7:2,

OP+12:4,':',OPCODE[OP+12]^.TOTAL0:7,OPCODE[OP+12]^.TOTAL0/SLDO*100:7:2);
     END;
   PCTMAX:=ROUND(SIND/MAXOP*20);
   WRITE(H,CR,CR,'SIND  OPCODE: 248..255  TOTAL:',
                 SIND:8,SIND/OPTOTAL*100:16:2,' % ');
   FOR I:=1 TO PCTMAX DO WRITE(H,'*');
   IF SIND<>0 THEN
     BEGIN
       WRITELN(H,CR); WRITEHDR(H,8);
       FOR OP:=248 TO 249 DO
         WRITELN(H,OP:4,':',OPCODE[OP]^.TOTAL0:7,OPCODE[OP]^.TOTAL0/SIND*100:7:2,
              OP+2:4,':',OPCODE[OP+2]^.TOTAL0:7,OPCODE[OP+2]^.TOTAL0/SIND*100:7:2,
              OP+4:4,':',OPCODE[OP+4]^.TOTAL0:7,OPCODE[OP+4]^.TOTAL0/SIND*100:7:2,

OP+6:4,':',OPCODE[OP+6]^.TOTAL0:7,OPCODE[OP+6]^.TOTAL0/SIND*100:7:2);
```

```
    END;
  WRITELN(H);
END;

BEGIN(* SHORTSTUFF *)
  SHORT1(LISTFILE);
  SHORT2(LISTFILE);
END;

PROCEDURE SHORTST;
VAR  I:INTEGER;
BEGIN
  INUM:=OPCODE[OP]^.TOTAL0;
  PCTMAX:=ROUND(INUM/MAXOP*20);
  WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
  FOR I:=1 TO PCTMAX DO WRITE('*');
  WRITELN(LISTFILE);
END;

PROCEDURE ONEST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
    BEGIN
      INUM:=TOTAL1;
      PCTMAX:=ROUND(INUM/MAXOP*20);
      WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
      IF TOTAL1<>0 THEN
        BEGIN
          FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
          WRITELN(LISTFILE,CR);
          WRITEHDR(LISTFILE,1); WRITELN(LISTFILE);
          WRITEHDR(LISTFILE,2);
          FOR I:=0 TO 7 DO
           WRITELN(LISTFILE,I:5,BYTEONE1[I]:13,BYTEONE1[I]/TOTAL1*100:14:2);
        END
      ELSE WRITELN(LISTFILE);
    END;
  END;

PROCEDURE TWOST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
    BEGIN
      PCTMAX:=ROUND(TOTAL2/MAXOP*20);
      WRITE(LISTFILE,TOTAL2:8,TOTAL2/OPTOTAL*100:16:2,' % ');
      FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
      WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,3);
      WRITELN(LISTFILE); WRITEHDR(LISTFILE,4);
      IF TOTAL2=0 THEN
        FOR I:=0 TO 7 DO
          WRITELN(LISTFILE,I:5,BYTEONE2[I]:13,0.0:14:2,BYTETWO2[I]:9,0.0:14:2)
      ELSE
        FOR I:=0 TO 7 DO
          WRITELN(LISTFILE,I:5,BYTEONE2[I]:13,BYTEONE2[I]/TOTAL2*100:14:2,
                          BYTETWO2[I]:9,BYTETWO2[I]/TOTAL2*100:14:2);
      IF OP=205 THEN
        BEGIN
          WRITELN(LISTFILE); WRITEHDR(LISTFILE,7);
```

```
          IF TOTAL2=0 THEN
            FOR I:=2 TO 15 DO
               WRITELN(LISTFILE,NAMES[56+I],FLAVOR2[I]:9,0.0:14:2,'     ',
                         NAMES[56+I+14],FLAVOR2[I+14]:9,0.0:14:2)
          ELSE
            FOR I:=2 TO 15 DO
              WRITELN(LISTFILE,NAMES[56+I],FLAVOR2[I]:9,
                        FLAVOR2[I]/TOTAL2*100:14:2,'     ',
                        NAMES[56+I+14],FLAVOR2[I+14]:9,
                        FLAVOR2[I+14]/TOTAL2*100:14:2);
        END;
    END;
END;

PROCEDURE WORDST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
    BEGIN
      INUM:=TOTAL3;
      PCTMAX:=ROUND(INUM/MAXOP*20);
      WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
      IF TOTAL3<>0 THEN
        BEGIN
          FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
          WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,1);
          WRITELN(LISTFILE); WRITEHDR(LISTFILE,2);
          FOR I:=0 TO 15 DO
            WRITELN(LISTFILE,I:5,PARMONE3[I]:13,PARMONE3[I]/TOTAL3*100:14:2);
        END
      ELSE WRITELN(LISTFILE);
    END;
END;

PROCEDURE LOPTST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
    BEGIN
      INUM:=TOTAL4;
      PCTMAX:=ROUND(INUM/MAXOP*20);
      WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
      IF TOTAL4<>0 THEN
        BEGIN
          FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
          WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,3);
          WRITELN(LISTFILE); WRITEHDR(LISTFILE,4);
          FOR I:=0 TO 7 DO
            WRITELN(LISTFILE,I:5,BYTEONE4[I]:13,BYTEONE4[I]/TOTAL4*100:14:2,
              PARMTWO4[I]:9,PARMTWO4[I]/TOTAL4*100:14:2);
          FOR I:=8 TO 15 DO
            WRITELN(LISTFILE,I:5,PARMTWO4[I]:36,PARMTWO4[I]/TOTAL4*100:14:2);
        END
      ELSE WRITELN(LISTFILE);
    END;
END;

PROCEDURE WORDSST;
VAR   I:INTEGER;
BEGIN
```

```
  WITH OPCODE[OP]^ DO
    BEGIN
      INUM:=TOTAL5;
      PCTMAX:=ROUND(INUM/MAXOP*20);
      WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
      IF TOTAL5<>0 THEN
        BEGIN
          FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
          WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,5);
          WRITELN(LISTFILE); WRITEHDR(LISTFILE,6);
          FOR I:=0 TO 15 DO
            WRITELN(LISTFILE,I:5,PARMONE5[I]:13,PARMONE5[I]/TOTAL5*100:14:2,
              PARMTWO5[I]:9,PARMTWO5[I]/TOTAL5*100:14:2,
              PARMTHREE5[I]:9,PARMTHREE5[I]/TOTAL5*100:14:2);
        END
      ELSE WRITELN(LISTFILE);
    END;
END;

PROCEDURE CMPRSSST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
    BEGIN
      PCTMAX:=ROUND(TOTAL6/MAXOP*20);
      WRITE(LISTFILE,TOTAL6:8,TOTAL6/OPTOTAL*100:16:2,' % ');
      FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
      WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,7);
      IF TOTAL6=0 THEN
        BEGIN
          FOR I:=0 TO 19 DO
            WRITELN(LISTFILE,NAMES[86+I],FLAVOR6[I]:9,0.0:14:2,'     ',
                    NAMES[106+I],FLAVOR6[I+20]:9,0.0:14:2);
          WRITELN(LISTFILE,NAMES[126]:44,FLAVOR6[40]:9,0.0:14:2);
        END
      ELSE
        BEGIN
          FOR I:=0 TO 19 DO
            WRITELN(LISTFILE,NAMES[86+I],FLAVOR6[I]:9,
              FLAVOR6[I]/TOTAL6*100:14:2,
              NAMES[106+I]:13,FLAVOR6[I+20]:9,FLAVOR6[I+20]/TOTAL6*100:14:2);
          WRITELN(LISTFILE,NAMES[126]:44,
              FLAVOR6[40]:9,FLAVOR6[40]/TOTAL6*100:14:2);
        END;
    END;
END;

PROCEDURE CMPRSS2ST;
VAR   I:INTEGER;
BEGIN
  WITH OPCODE[OP]^ DO
  BEGIN
    INUM:=TOTAL7;
    PCTMAX:=ROUND(INUM/MAXOP*20);
    WRITE(LISTFILE,INUM:8,INUM/OPTOTAL*100:16:2,' % ');
    FOR I:=1 TO PCTMAX DO WRITE(LISTFILE,'*');
    WRITELN(LISTFILE,CR); WRITEHDR(LISTFILE,7);
    FOR I:=1 TO 6 DO
      BEGIN
        IF INUM<>0 THEN
```

```
          WRITE(LISTFILE,NAMES[51+I],FLAVOR7[I]:9,FLAVOR7[I]/INUM*100:14:2,'
')
        ELSE
          WRITE(LISTFILE,NAMES[51+I],FLAVOR7[I]:9,0.0:14:2,'     ');
        IF (I MOD 2=0) THEN WRITELN(LISTFILE);
      END;
  END;
END;

PROCEDURE GINIT;
BEGIN
  MAXOP:=0;
  FOR OP:=128 TO 215 DO
    WITH OPCODE[OP]^ DO
      CASE RECTYPES[OP] OF
            ONE,CHRS,BLK:IF (TOTAL1>MAXOP) THEN MAXOP:=TOTAL1;
                     TWO:IF (TOTAL2>MAXOP) THEN MAXOP:=TOTAL2;
                WORD,OPT:IF (TOTAL3>MAXOP) THEN MAXOP:=TOTAL3;
                    LOPT:IF (TOTAL4>MAXOP) THEN MAXOP:=TOTAL4;
                   WORDS:IF (TOTAL5>MAXOP) THEN MAXOP:=TOTAL5;
                  CMPRSS:IF (TOTAL6>MAXOP) THEN MAXOP:=TOTAL6;
                 CMPRSS2:IF (TOTAL7>MAXOP) THEN MAXOP:=TOTAL7
       END;
END;

BEGIN (* SEGMENT PROCEDURE GATHER *)
  GINIT;
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITE(CHR(7),'Output file for opcode statistics (<CR> for none): ');
  READLN(FILENAME);
  DISPLAY:=(FILENAME<>'');
  CONSOLE:=(FILENAME='CONSOLE:') OR (FILENAME='#1:');
  IF DISPLAY THEN
    BEGIN
      IF (FILENAME<>LASTFILENAME) THEN
        BEGIN
          CLOSE(LISTFILE,LOCK);
          REWRITE(LISTFILE,FILENAME);
          LASTFILENAME:=FILENAME;
        END;
      PAGE(OUTPUT);
      PROCSTUFF;
      JUMPSTUFF;
      SHORTSTUFF;
      FOR OP:=128 TO 215 DO
        BEGIN
          WRITE(LISTFILE,CR,NAMES[OP],'  Opcode:',OP:4,'   Total:');
          CASE RECTYPES[OP] OF
                     SHORT:SHORTST;
                 OPT,WORD:WORDST;
              ONE,CHRS,BLK:ONEST;
                       TWO:TWOST;
                      LOPT:LOPTST;
                     WORDS:WORDSST;
                    CMPRSS:CMPRSSST;
                   CMPRSS2:CMPRSS2ST
          END;
        END;
      WRITELN(CR,CR,CR,OPTOTAL:20,'   Total operators');
```

```
    END;
END;

SEGMENT PROCEDURE DATACOUNT;
TYPE  ACTPTR=^ACTREC;
      ACTREC=RECORD
        OFFSET,TOTAL:INTEGER;
        LES,GTR:ACTPTR
      END;
VAR   TOTAL:INTEGER;
      HEAP:^INTEGER;
      TREETRUNK,ENTRY:ACTPTR;
      FILENAME:STRING;

PROCEDURE SETORDER;
VAR INDEX:INTEGER;

PROCEDURE DATASET(TREEMARK:ACTPTR);
BEGIN
  {$R-}
  IF DSSTART^[INDEX]<TREEMARK^.TOTAL THEN
    IF TREEMARK^.LES<>NIL THEN
      DATASET(TREEMARK^.LES)
    ELSE
      BEGIN
        NEW(ENTRY);
        ENTRY^.OFFSET:=INDEX;
        ENTRY^.TOTAL:=DSSTART^[INDEX];
        ENTRY^.LES:=NIL;
        ENTRY^.GTR:=NIL;
        TREEMARK^.LES:=ENTRY;
      END
  ELSE IF TREEMARK^.GTR<>NIL THEN
      DATASET(TREEMARK^.GTR)
    ELSE
      BEGIN
        NEW(ENTRY);
        ENTRY^.OFFSET:=INDEX;
        ENTRY^.TOTAL:=DSSTART^[INDEX];
        ENTRY^.LES:=NIL;
        ENTRY^.GTR:=NIL;
        TREEMARK^.GTR:=ENTRY;
      END;
  {$R+}
END;

BEGIN
  NEW(TREETRUNK);
  TREETRUNK^.TOTAL:=0;
  TREETRUNK^.LES:=NIL;
  TREETRUNK^.GTR:=NIL;
  DATAREF:=0; INDEX:=0;
  REPEAT
    {$R-}
    INDEX:=INDEX + SCAN((DATASEGSIZE-INDEX)*2,<>CHR(0),DSSTART^[INDEX]) DIV 2;
    IF DSSTART^[INDEX]>0 THEN
      BEGIN
        DATASET(TREETRUNK);
        DATAREF:=DATAREF + DSSTART^[INDEX];
        DSSTART^[INDEX]:=0;
```

```
      END;
    {$R+}
  UNTIL INDEX>=DATASEGSIZE;
END;

PROCEDURE DATAHEADER(VAR H2:INTERACTIVE);
VAR  I:INTEGER;
BEGIN
  WRITELN(H2,CR,CR,'Data Segment size:',DATASEGSIZE:6,'      Data references:',
                       DATAREF:6,'      Lex level',PROCLEX[DATAPROC]:6);
  WRITE(H2,CR,CR,'For segment ');
  FOR I:=1 TO 8 DO WRITE(H2,CHR(SEGDIREC[63 + DATASEG*8 +I]));
  WRITELN(H2,' Procedure #',DATAPROC:3);
  WRITELN(H2,'Offset(word)  Total      %');
END;

PROCEDURE PRINTDATA(TREE:ACTPTR);
BEGIN
  IF TREE^.GTR<>NIL THEN PRINTDATA(TREE^.GTR);
  TOTAL:=TREE^.TOTAL;
  IF DISPLAY THEN WRITELN(LISTFILE,
            TREE^.OFFSET:9,TOTAL:11,TOTAL/DATAREF*100:9:2);
  IF TREE^.LES<>NIL THEN PRINTDATA(TREE^.LES);
END;

BEGIN (* DATACOUNT *);
  MARK(HEAP);
  PAGE(OUTPUT);
  GOTOXY(0,10);
  WRITE(CHR(7),'Output file for data segment statistics(<CR> for none): ');
  READLN(FILENAME);
  DISPLAY:=(FILENAME<>'');
  CONSOLE:=(FILENAME='CONSOLE:') OR (FILENAME='#1:');
  IF DISPLAY AND (FILENAME<>LASTFILENAME) THEN
    BEGIN
      CLOSE(LISTFILE,LOCK);
      REWRITE(LISTFILE,FILENAME);
      LASTFILENAME:=FILENAME;
    END;
  PAGE(OUTPUT);
  SETORDER;
  IF DISPLAY THEN DATAHEADER(LISTFILE);
  IF DATAREF>0 THEN
    PRINTDATA(TREETRUNK^.GTR)
  ELSE
    BEGIN
      IF DISPLAY THEN WRITELN(LISTFILE,CR,CR,
                'sorry but there were no accesses',
                ' to this data segment from dis-assembled procedures');
    END;
  PROMPT;
  RELEASE(HEAP);
END;

PROCEDURE PROMPT;
VAR  CH:CHAR;
BEGIN
  WRITE(CHR(7),CR,CR,'press spacebar to continue...');
  REPEAT  READ(CH)  UNTIL CH=' ';
  WRITELN;
```

```
 END;


 BEGIN(*MAIN STUFF*)
   INIT;
   DISASSEMBLE;
   IF DATAWATCH THEN DATACOUNT;
   GATHER;
   IF DISPLAY AND NOT CONSOLE THEN CLOSE(LISTFILE,LOCK);
 END.

{ +-------------------------------------------------------------------+
  |                                                                   |
  |                     F     I     N     I     S                     |
  |                                                                   |
  +-------------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 Disassembler
```

```
##############################################################################
### FILE: UCSD Pascal 1.5 Interp 3.MAC
##############################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "3.MAC")

 EIS=1
 FPI=1
 LSI=1


; +--------------------------------------------------------------------+
; |                                                                    |
; |                      F     I     N     I     S                     |
; |                                                                    |
; +--------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp 3.MAC
```

```
##############################################################################
### FILE: UCSD Pascal 1.5 Interp CopyRite
##############################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "copyr.mac")

          ;
          ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
          ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
          ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
          ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
          ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
          ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
          ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
          ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
          ; PUBLISHER.
          ;

; +------------------------------------------------------------------+
; |                                                                  |
; |                     F     I     N     I     S                    |
; |                                                                  |
; +------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp CopyRite
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp DL.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "dl.mac")

        .TITLE  DL-11 INPUT HANDLER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       SYSTEM TABLE CONTENTS                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT
.=120
        DLRINT          ; READ LOCATION IN VECTOR
        200
        DLXINT          ; TRANSMIT LOCATION IN VECTOR
        200

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   8.
        .BLKW   3
        .ENDR
        .WORD   INBIT!OUTBIT,DLSTRT,DLABRT
        .PAGE
        .CSECT  DLDRVR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;              DL-11 INPUT, OUTPUT HANDLER                             ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



;       THESE APPLY TO BOTH READ AND TRANSMIT HANDLERS
DLRUNT: .WORD   0               ; DRIVER STATUS WORDS
DLXUNT: .WORD   0
DLRBFA: .WORD                   ; USER BUFFER ADDRESSES
DLXBFA: .WORD
DLRLNG: .WORD                   ; USER BUFFER LENGTHS
DLXLNG: .WORD
DLRCSR: .WORD   177520          ; READ CONTROL STATUS REGISTER
DLRBUF: .WORD   177522          ; SERIAL PORT READ BUFFER
```

```
DLXCSR: .WORD   177524          ; TRANSMIT CONTROL STATUS REGISTER
DLXBUF: .WORD   177526          ; SERIAL PORT TRANSMIT BUFFER
DLOFST  = 4                     ; STACK OFFSET AFTER ENTERING DRIVER

DLSTRT: TST     (R3)+           ; CHECK FOR READ OR WRITE OP
        BNE     DLRSTR          ; IF READ OP THEN JMP TO READ HANDLER
DLXSTR: TST     DLXUNT          ; SEE IF AN IO ALREADY IN PROGRESS
        BNE     DLXSTR          ; IF SO LOOP UNTIL THE IO IS COMPLETE
        MTPS    #200            ; NO INTERRUPTS PLEASE
        MOV     R1,DLXUNT       ; MARK HANDLER AS BUSY
        BIS     #BSYBIT,(R1)    ; MARK LOGICAL UNIT AS BUSY
        CLR     (SP)            ; SET UP RETURN STUFF ON STACK...PR-0 PS
        MOV     R3,-(SP)        ; NOW THE RETURN ADDRESS
        MOV     <UBUFFR+DLOFST>(SP),DLXBFA      ; GRAB USER BUFFER ADDR
        MOV     <URLENG+DLOFST>(SP),DLXLNG      ; AND REQUESTED IO LENGTH
DLXINT: BIC     #100,@DLXCSR    ; DISABLE INTERRUPTS
        TST     DLXUNT          ; ANY IO'S IN PROGRESS
        BEQ     DLEXIT          ; IF NOT JUST FORGET IT
        TST     DLXLNG          ; ANY CHARS LEFT TO BE SENT?
        BEQ     DLXQT           ; IF NOT THEN FINISH UP IO
        MOVB    @DLXBFA,@DLXBUF ; SEND CHAR TO DL
        BIS     #100,@DLXCSR    ; ALLOW INTERRUPT
        INC     DLXBFA          ; BUMP BUFFER POINTER TO NEXT CHAR
        DEC     DLXLNG          ; ALSO REFLECT ONE FEWER CHAR TO SEND
        JMP     @#INTRTN        ; THIS STRUCTURE IMPLIES AN IO IS NOT
                                ; DONE UNTIL THE LAST INTERRUPT IS RECEIVED

DLXQT:  BIC     #BSYBIT,@DLXUNT ; CLEAR BUSY BIT IN IO UNIT TABLE
        CLR     DLXUNT          ; MARK HANDLER AS NOT BUSY NOW
DLEXIT: JMP     @#INTRTN        ; BACK TO WHEREVER

;       THESE APPLY ONLY TO THE READ HANDLER
BUFSIZ  = 216                   ; CIRCULAR BUFFER SIZE IS 142
CIRBUF: .BLKB   BUFSIZ          ; CIRCULAR I/0 BUFFER
DLHEAD: .WORD   CIRBUF          ; NEXT OUT CHAR AVAILABLE IN CIRBUF
DLTAIL: .WORD   CIRBUF          ; NEXT AVAILABLE OPENING IN CIRBUF
CHARS:  .WORD   0               ; NUMBER OF CHARS PRESENT IN CIRBUF
ERRFLG: .WORD   0               ; TEMP STORAGE OF ERROR TYPE IN FLAG
OVFERR  = 17                    ; RING BUFFER OVERFLOW ERROR
DLERR   = 4                     ; UNDEFINED HARDWARE ERROR AT DL

DLRSTR: TST     DLRUNT          ; SEE IF AN IO ALREADY IN PROGRESS
        BNE     DLRSTR          ; IF SO LOOP UNTIL THE IO IS COMPLETE
        MTPS    #200            ; NO INTERRUPTS PLEASE
        MOV     R1,DLRUNT       ; MARK HANDLER AS BUSY
        BIS     #BSYBIT,(R1)    ; MARK LOGICAL UNIT AS BUSY
        CLR     (SP)            ; SET UP RETURN STUFF ON STACK...PR-0 PS
        MOV     R3,-(SP)        ; NOW THE RETURN ADDRESS
        MOV     <UBUFFR+DLOFST>(SP),DLRBFA      ; GRAB USER BUFFER ADDR
        MOV     <URLENG+DLOFST>(SP),DLRLNG      ; AND REQUESTED IO LENGTH
        BEQ     DLRQT           ; IF ZERO LENGTH READ FORGET IT
        TST     CHARS           ; CHECK FOR AVAILABLE CHARS
        BNE     GET             ; IF ANY THEN GET THEM
        JMP     @#INTRTN        ; ELSE RETURN AND WAIT FOR INTERUPT

DLRINT: MOV     R0,-(SP)        ; STASH REGISTER
        .IF     DF,TERAK        ; IF A TERAK THEN
        MOV     @DLRCSR,R0      ; GET ERROR BITS
        BMI     DLRERR          ; IF ERROR AT DL THEN QUIT
        MOVB    @DLRBUF,@DLTAIL ; GRAB CHAR FROM BUFFER
```

```
        .IFF
        MOV     @DLRBUF,R0      ; ELSE GRAB CHAR FROM DL
        BMI     DLRERR          ; IF ERROR BIT ON THEN EXIT
        MOVB    R0,@DLTAIL      ; ELSE STORE CHAR IN CIRBUF
        .ENDC
        BICB    #200,@DLTAIL    ; CLEAR RANDOM BIT
        INC     CHARS           ; SHOW ONE MORE CHAR IN CIRBUF
        CMP     CHARS,#BUFSIZ   ; CHECK FOR TOO MANY CHARS IN CIRBUF
        BEQ     OFLWER          ; IF CHARS GREATER THEN OVERFLOW ERROR
        INC     DLTAIL          ; ADVANCE POINTER TO CIRBUF
        CMP     DLTAIL,#CIRBUF+BUFSIZ   ; CHK FOR END OF CIRBUF
        BNE     1$              ; IF NOT THEN FORGET IT
        MOV     #CIRBUF,DLTAIL  ; ELSE RESET DLTAIL
1$:     MOV     (SP)+,R0        ; RESTORE REGISTER
GET:    TST     DLRUNT          ; CHECK FOR ANY I/O'S
        BEQ     ENDIT           ; IF NONE THEN FORGET IT
        MOVB    @DLHEAD,@DLRBFA ; STUFF CHAR INTO USER BUF
        INC     DLHEAD          ; ADVANCE POINTER TO CIRBUF
        CMP     DLHEAD,#CIRBUF+BUFSIZ   ; END OF CIRBUF?
        BNE     2$              ; IF NOT FORGET IT
        MOV     #CIRBUF,DLHEAD  ; ELSE RESET TO CIRBUF START
2$:     DEC     CHARS           ; REDUCE # CHARS
        DEC     DLRLNG          ; SHOW ONE LESS SPACE TO FILL
        BEQ     DLRQT           ; IF NO MORE NEEDED THEN LEAVE
        INC     DLRBFA          ; ADVANCE POINTER TO USER BUFFER
        MTPS    #0              ; ALLOW INTERUPT
        MTPS    #200            ; REGAIN CONTROL
        TST     CHARS           ; SEE IF MORE CHARS AVAILABLE
        BNE     GET             ; ELSE CONTINUE TO FILL USER BUFFER
ENDIT:  JMP     @#INTRTN        ; BACK TO WHEREEVER

DLRERR: MOVB    #DLERR,ERRFLG   ; MOV ERROR NUMBER TO ERROR FLAG WORD
        TSTB    @DLRBUF         ; READ FROM RBUF TO CLEAR DL
        BR      ERROR           ; SKIP OVER OTHER ERROR TYPE
OFLWER: DEC     CHARS           ; CAN'T BE MORE THAN 129 CHARS
        MOVB    #OVFERR,ERRFLG  ; MOV ERROR TYPE TO ERROR FLAG WORD
ERROR:  MOV     (SP)+,R0        ; RESTORE REGISTER
        TST     DLRUNT          ; CHECK IF I/O PENDING
        BEQ     ENDIT           ; IF NOT THEN END INTERUPT
DLRQT:  MOVB    ERRFLG,@DLRUNT  ; MOVE ANY ERRORS TO IORESULT
        CLR     ERRFLG          ; NOW SHOW NO ERROR
        BIC     #BSYBIT,@DLRUNT ; CLEAR BUSY BIT IN IO UNIT TABLE
        CLR     DLRUNT          ; MARK HANDLER AS NOT BUSY
        JMP     @#INTRTN        ; BACK TO WHEREVER

DLABRT: MTPS    #200            ; DISABLE INTERUPTS
        MOV     #CIRBUF,DLHEAD  ; RESET CIRCULAR BUFFER POINTERS
        MOV     DLHEAD,DLTAIL
        CLR     CHARS           ; ZERO CHAR COUNT IN CIRBUF
        TST     DLRUNT          ; CHECK IF I/0 IN PROGRESS
        BEQ     1$              ; IF NOT THEN FORGET IT
        BIC     #BSYBIT,@DLRUNT ; CLEAR BSYBIT IN UNIT I/O TABLE
        CLR     DLRUNT          ; MARK HANDLER AS NOT BUSY NOW
1$:     TST     DLXUNT          ; CHECK IF I/O IN PROGRESS
        BEQ     2$              ; IF NOT THEN FORGET IT
        BIC     #BSYBIT,@DLXUNT ; CLEAR BSYBIT IN UNIT I/O TABLE
        CLR     DLXUNT          ; MARK HANDLER AS NOT BUSY NOW
2$:     MOV     @#4,-(SP)       ; SAVE OLD TRAP ADDRESS
        MOV     #OFFLNE,@#4     ; LOAD A TEMP TRAP ADDRESS
```

```
        BIS     #100,@DLRCSR    ; ENABLE INTERUPTS (WHICH ACCESSES DL ADDRESS)
        BR      ABTOUT          ; IF NORMAL RETURN FROM OP THEN QUIT

 ;      THIS CODE IS EXECUTED WHEN A DL IS NOT ON LINE
 OFFLNE: MOV    #11,R5          ; VOLUME NOT ON LINE ERROR
        CMP     (SP)+,(SP)+     ; POP PS&PC FROM TRAP OFF STACK

 ABTOUT: MOV    (SP)+,@#4       ; RELOAD ORIGINAL TRAP ADDRESS
        MTPS    #0              ; ALLOW INTERUPTS
        RTS     PC

        .END

; +-------------------------------------------------------------------+
; |                                                                   |
; |                     F     I     N     I     S                     |
; |                                                                   |
; +-------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp DL.MAC
```

```
##############################################################################
### FILE: UCSD Pascal 1.5 Interp EIS.MAC
##############################################################################



; UCSD PASCAL I.5 INTERPRETER (FILE "eis.mac")

 EIS=1
 FPI=1

; +--------------------------------------------------------------------+
; |                                                                    |
; |                      F     I     N     I     S                     |
; |                                                                    |
; +--------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp EIS.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp IOTR.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "iotrap.mac")

        .TITLE  INTERRUPT AND TRAP SUBSYSTEM
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       TRAP VECTOR CONTENTS                           ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=0
        TRAP    NOTIMP  ; ILLEGAL OPCODE MOST LIKELY, NO FP
        0
        T4$INT          ; 4-TRAP HANDLER
        0
        T10$INT         ; 10-TRAP HANDLER
        0
.=24
        173000          ; POWER UP LOCATION
        0
.=34
        TP$INT          ; TRAP HANDLER
        0               ; PR-0
        BACK            ; ENTRY POINT FOR BOOT LOADER
.=60
        TR$INT          ; KEYBOARD INTERRUPT HANDLER
        200             ; PR-4
        TX$INT          ; CONSOLE PRINTER HANDLER
        200             ; PR-4
.=100
        KW$INT          ; KW-11 (MAYBE REFRESH HARDWARE!) CLOCK HANDLER
        301             ; PR-6...CARRY SET FOR ADC OP
.=244
        FP$INT          ; FLOATING POINT EXCEPTIONS
        0
        .PAGE

.=300
        .ASCII  'COPYRIGHT (C) 1978, REGENTS OF UNIV OF CALIF,SD'
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;                                                                       ;
;                       SYSCOM CONTENTS                                 ;
;                                                                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.=400

IORSLT: .WORD   0
XEQERR: .WORD   0
SYSUNT: .WORD   0
BUGSTA: .WORD   0
GDIRP:  .WORD   NIL
BOMBP:  .WORD   NIL
STKBAS: .WORD   NIL
LASTMP: .WORD   NIL
JTAB:   .WORD   NIL
SEG:    .WORD   NIL
MEMTOP: .WORD   157776
BOMBIPC:.WORD   0
HLTLIN: .WORD   NIL
BRKPTS: .BLKW   4
        .BLKW   10.
LOTIME: .WORD   0
HITIME: .WORD   0
NOBREAK=100     ;NO BREAK BIT IN MISCINFO
MSCNFO: .WORD   ^B0000000
CRTTYP: .WORD   3       ;TELETYPE
CRTESC: .BYTE   0,CR,0,0,0,0,0
FILCNT: .BYTE   10.
        .BLKB   4
CRTNFO: .WORD   0.,72.
        .BYTE   0,0,0,0
EOF:    .BYTE   3
FLUSH:  .BYTE   6
BREAK:  .BYTE   0
STOP:   .BYTE   23
CHRDEL: .BYTE   '_
BADCHR: .BYTE   '?
LINDEL: .BYTE   177
ALTMOD: .BYTE   33
        .BLKB   6
SEGTBL: .BLKW   3*<MAXSEG+1>
        .PAGE
        .CSECT  IOTRAP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                       ;
;               KW-11, 4 & 10 TRAP HANDLERS                             ;
;                                                                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

T4$INT: TRAP    BADMEM

T10$INT:TRAP    NOTIMP  ; CROAK SYSTEM FOR ILLEGAL OPCODE

FP$INT: TRAP    FPIERR  ; EXECUTION ERROR!

KW$INT: ADC     @#LOTIME        ; IN 60-TH'S...CARRY IS SET IN TRAP VECTOR
        ADC     @#HITIME
        RTI
```

```
INTRTN: ; ALL IO DRIVERS MUST USE JUMP HERE INSTEAD OF
        ; DOING THEIR OWN RTI...THE SYSTEM MAY HANG IN THE WAIT
        ; INSTRUCTION OF UNITIO IF THIS IS NOT DONE!!
        CMP     @(SP),(PC)+     ; IS THE NEXT INSTRUCTION A WAIT??
        WAIT
        BNE     1$              ; IF NOT THEN DO IT
        ADD     #2,@SP          ; ELSE SKIP THE WAIT OPCODE AND THEN
1$:     RTI                     ; RETURN
        .PAGE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                             ;
;               "TRAP"  INTERRUPT  HANDLER                                    ;
;                                                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TP$INT: ; ENTRY POINT FOR "TRAP" INSTRUCTION INTERRUPTS.  THESE
        ; ARE USED FOR EXECUTION ERRORS AND SOME SYSTEM REQUESTS.
        MOV     R1,-(SP)        ; R1 IS USED FOR DETERMINING TRAP TYPE
        MOV     2(SP),R1        ; GRAB OLD PC OFF THE STACK
        MOV     -(R1),R1        ; NOW R1 HAS TRAP INSTRUCTION FROM CORE
        MOVB    R1,R1           ; ISOLATE LOW BYTE WITH SIGN EXTEND
        BLT     TPRQST          ; A MINUS PARAM IS A SYSTEM REQUEST
        BGT     XQ.ERR          ; GREATER THAN IS EXECUTION ERROR
        RESET                   ; ZERO IS FATAL SYSTEM ERROR...REBOOT!
        JMP     @24             ; USE POWER-UP VECTOR FOR BOOT ADDR
XQ.ERR: ; HERE WE ARE FOR AN EXECUTION ERROR...RESTORE A VALID
        ; ENVIRONMENT FOR THE SYSTEM AND CXP 0,2...EXECERROR
        MOV     LASTMP,MP
        MOV     STKBAS,BASE
        MOV     #BACK,BK
        MOV     (PC)+,@BK       ; ENSURE OP FETCH IS OK...STOP BREAKING
        GETNEXT
        MOVB    R1,XEQERR       ; SET UP PARAMS IN SYSCOM TO ERR HANDLERS
        MOV     SP,BOMBP        ; SET UP BOMB MCSWP FOR DEBUGGER
        SUB     #MSDLTA+4,BOMBP
        MOV     IPC,BOMBIPC
        CLR     -(SP)
        MOV     BK,-(SP)
        MOV     #CXP0.2,IPC
        RTI
CXP0.2: .BYTE   77.+128.,0,2,326

TPRQST: ASL     R1              ; DOUBLE FOR WORD ADDRESSING
        SUB     R1,PC           ; CASE STMT, R1 NEGATIVE...REALLY ADDS R1
        TRAP    SYSERR          ; SHOULD NEVER DO THIS
        BR      TTYOUT          ; -1 IS TTYOUT REQUEST

TTYOUT: TST     TXCHAR          ; SEE IF ANY CHAR WAITING ALREADY
        BPL     TTYOUT          ; >=0 -> BUSY...HANG UNTIL NEG
        MOV     R0,TXCHAR       ; PLACE THE CHAR IN HIGH PRIOR BUFFER
        MOV     (SP)+,R1        ; RESTORE REG
        TSTB    @TXCSR          ; SEE IF DL-11 IS READY FOR A CHAR
        BPL     1$              ; IF NOT THEN RETURN ELSE
        JMP     TX$INT          ; MAKE TX THINK AN IO IS COMPLETED
1$:     RTI

        .PAGE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                             ;
```

```
;               CRT INPUT-OUTPUT HANDLER                                ;
;                                                                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


TRCSR:  .WORD   177560
TRBUF:  .WORD   177562
TRBFSZ  =  64.                  ; BYTES IN TT INPUT BUFFER
TRBUFR: .BLKB   TRBFSZ          ; RING BUFFER FOR TTY INPUT
TRHEAD: .WORD   0               ; INDEX OF CURRENT QUEUE HEAD
TRTAIL: .WORD   0               ; INDEX OF CURRENT QUEUE TAIL
TRQUED: .WORD   0               ; CHARS IN QUEUE WAITING TO BE READ
TRUNIT: .WORD   0               ; UNIT # OF CURRENT IO (ZERO MEANS NONE)
TRBUFA: .WORD   0               ; ADDRESS OF USERS BUFFER (BYTE ADDR)
TRLENG: .WORD   0               ; NUMBER OF BYTES USER WANTS READ IN
TROFST  =  6                    ; OFFSET FROM MY STACK TO PARAMS
BUZZ    = 07
SFLAG:  .WORD   0               ; NON-ZERO IF OUTPUT TO BE STOPPED
OFLAG:  .WORD   0               ; NON-ZERO IF OUTPUT TO BE FLUSHED
ALOCK:  .WORD   0               ; ZERO -> ALPHA LOCK ON

TR$INT: ; ENTRY POINT FOR KEYBOARD INTERRUPTS
        MOV     R0,-(SP)        ; SAVE R0, R0 IS USED FOR CHAR
        MOVB    @TRBUF,R0       ; GRAB CHAR FROM DL-11
        BIC     #177600,R0      ; CHUCK PARITY AND ANY JUNK BITS
        CMPB    R0,BREAK        ; IS IT THE STANDARD BREAK CHAR
        BNE     1$
        BIT     #NOBREAK,MSCNFO ; BREAK DISABLED??
        BNE     TREXIT          ; IF SO THEN NO BLOWUP
        MOV     (PC)+,R0        ; STICK TRAP OP INTO R0
        TRAP    UBREAK
        MOV     R0,@#BACK       ; STICK TRAP INTO FETCH SEQUENCE
        BR      TREXIT
1$:     CMPB    R0,STOP         ; IS IT A CONTROL S? (START-STOP)
        BNE     2$
        COM     SFLAG           ; FLIP STATE OF STOPPED FLAG (0 MEANS GOING)
        BR      TREXIT          ; AND SPLIT
2$:     CMPB    R0,FLUSH        ; A CONTROL F? (FLUSH REQUEST?)
        BNE     3$
        COM     OFLAG           ; FLIP FLUSH STATE
        CLR     TXLENG          ; HALT ANY IO IN PROGRESS
        BR      TREXIT          ; AND THEN GO AWAY
3$:     CMPB    R0,#DC2         ; ALPHALOC/K SWAP?
        BNE     TRQCHR
        COM     ALOCK
        BR      TREXIT
TRQCHR: CMP     TRQUED,#TRBFSZ  ; OVERFLOW BUFFER?
        BLT     1$              ; IF NOT THEN OK TO QUEUE CHAR
        .TTYOUT #BUZZ           ; TELL THE USER HIS INPUT WAS CHUCKED
        BR      TREXIT          ; AND FORGET IT (LIFE'S A BITCH)
1$:     TST     ALOCK           ; CHECK OUT ALHPA LOCK TOGGLE
        BNE     5$              ; ZERO SIGNALS ALOCK ON
        CMPB    R0,#'A!40       ; COMPARE TO A LOWER CASE A
        BLT     5$              ; IF LESS THEN PASS ON THOUGH
        CMPB    R0,#'Z!40       ; NOW SEE IF LEQ LC Z
        BHI     5$              ; IF > Z THEN PASS THROUGH
        BIC     #40,R0          ; ELSE ZAP LC BIT FOR 'A..'Z
5$:     MOV     R3,-(SP)        ; SAVE R3 FOR USE AS SCRATCH
        MOV     TRTAIL,R3       ; POINTER TO TAIL OF INPUT QUEUE
        MOVB    R0,TRBUFR(R3)   ; PLACE THE CHAR INTO TAIL OF QUEUE
```

```
        INC     R3              ; POINT AT NEXT POSITION IN QUEUE
        BIC     #TRBFSZ,R3      ; CHECK FOR WRAPAROUND
        INC     TRQUED          ; COUNT THE CHAR AS QUEUED
        MOV     R3,TRTAIL       ; AND SAVE FOR NEXT INPUT
        MOV     (SP)+,R3        ; RESTORE R3
        TST     TRUNIT          ; ANY IO'S IN PROGRESS TO KEYBOARD?
        BEQ     TREXIT          ; EQUAL ZERO IF NOT...QUIT
TRFLBF: MOV     R3,-(SP)        ; WELL, PUT SOME CHARS INTO INPUT BUFFER
        MOV     TRHEAD,R3       ; POINTER TO HEAD OF INPUT QUEUE
        MOVB    TRBUFR(R3),R0   ; MOVE A QUEUED CHAR TO USER BUFFER
        MOVB    R0,@TRBUFA      ; BUT WE WANT IT IN R0 FOR ECHO TOO
        INC     R3              ; BUMP QUEUE HEAD POINTER
        BIC     #TRBFSZ,R3      ; WRAPAROUND AGAIN (MAYBE)
        MOV     R3,TRHEAD       ; AND STASH NEW HEAD POINTER
        DEC     TRQUED          ;ONE LESS CHAR IN INPUT QUEUE
        MOV     (SP)+,R3        ; AND RESTORE R3
        CMP     TRUNIT,#UNITBL+6 ; SEE IF WE WANT TO ECHO AS WE READ IN
        BNE     2$              ; IF NOT INPUT UNIT, SKIP ECHO STUFF
        CMPB    R0,EOF          ; INPUT EOF CHAR?
        BNE     3$              ; IF NOT THEN SKIP
        MOV     TRLENG,R0       ; USE R0 FOR LOOP COUNTER
1$:     CLRB    @TRBUFA         ; ELSE NULL FILL INPUT BUFFER
        INC     TRBUFA
        SOB     R0,1$           ; FOR ALL REMAINING CHARS
        BR      TRQUIT          ; AND CONSIDER IO COMPLETE
3$:     CMPB    R0,#177         ; A DEL???
        BEQ     2$              ; DONT ECHO IT...TERAK BUG
        CMPB    R0,CRTESC       ; IS IT THE ESCAPE CHAR?
        BEQ     2$              ; DONT ECHO...MAY MESS UP FORMATTING
        BIC     #100,@TRCSR     ; PREVENT INPUT WHILE ECHOING (FUNNY WINDOW)
        .TTYOUT ; SEND R0 TO TTY AS ECHO
2$:     BIS     #100,@TRCSR     ; RESTORE INPUT ENABLE
        INC     TRBUFA          ; BUMP BUFFER ADDRESS
        DEC     TRLENG          ; ONE FEWER CHARS TO READ INTO BUFFER
        BEQ     TRQUIT          ; WE ARE DONE IF IT GOES TO ZERO
        TST     TRQUED          ; WE CAN TRANSFER MORE CHARS IF ANY IN BUFFER
        BGT     TRFLBF          ; SO GO TO FILL BUFFER LOCATION
        BR      TREXIT          ; WELL, NO CHARS, RETURN TO USER NOT DONE
TRQUIT: BIC     #BSYBIT,@TRUNIT ; MARK OUR UNIT AS NOT BUSY
        CLR     TRUNIT          ; AND NOW HANDLER IS NOT BUSY EITHER
TREXIT: MOV     (SP)+,R0        ; RESTORE R0
        JMP     INTRTN          ; AND RETURN TO WHEREVER

TRSTRT: ; THIS IS THE ENTRY POINT FOR STARTING I-O'S TO THE
        ; KEYBOARD DEVICE.  NOTE THE REASON THIS CODE IS HERE
        ; IS TO MAKE BRANCHING EASY.  ALSO PLEASE FORGIVE
        ; THE DIDLING TO SAVE REGISTERS ACROSS TRFLBF ETC.
        TST     TRUNIT          ; SEE IF ANY IO'S IN PROGRESS
        BNE     TRSTRT          ; AND HANG IF SO
        MTPS    #200            ; PREVENT ANY INTERRUPTS NOW
        MOV     R1,TRUNIT       ; MARK AS IO IN PROGRESS
        BIS     #BSYBIT,(R1)    ; MARK UNIT AS BUSY
        CLR     (SP)            ; ON RTI, BE AT PR0
        TST     (R3)+           ; GET RETURN ADDRESS AND
        MOV     R3,-(SP)        ; PLACE ON STACK FOR RTI AT TREXIT
        MOV     R0,-(SP)        ; NOW STACK LOOKS LIKE INSIDE OF TRINT
        MOV     <UBUFFR+TROFST>(SP),TRBUFA      ; SAVE USER BUF ADDR
        MOV     <URLENG+TROFST>(SP),TRLENG      ; AND REQUESTED TRANSFER LENG
        BEQ     TRQUIT          ; IF NOTHING, THEN QUIT NOW
        TST     TRQUED          ; IF SOME CHARS WAITING, THEN GET EM
```

```
        BNE     TRFLBF          ; AND PUT EM IN USERS BUFFER
        BR      TREXIT          ; ELSE RETURN TO SYSTEM

TRMSTRT:; THIS ROUTINE JUST FIGURES IF INPUT OR OUTPUT
        ; REQUEST AND TRANSFERS TO THE PROPER HANDLER.
        TST     @R3             ; ZERO MEANS A WRITE REQUEST...SEE UWRITE
        BEQ     TXSTRT          ; SO START TERM TRANSMIT
        BR      TRSTRT          ; ELSE A READ START

TRMABRT:; ENTERED TO CANCEL ANY IO'S PENDING
        ; OR IN PROGRESS ON CRT DEVICE
        MTPS    #200
        CLR     SFLAG
        CLR     OFLAG
        CLR     TRQUED
        MOV     TRTAIL,TRHEAD
        TST     TRUNIT
        BEQ     1$
        CLR     TRLENG
        BIC     #BSYBIT,@TRUNIT
        CLR     TRUNIT
1$:     TST     TXUNIT
        BEQ     2$
        CLR     TXLENG
        BIC     #BSYBIT,@TXUNIT
        CLR     TXUNIT
2$:     MTPS    #0
        RTS     PC

TXCSR:  .WORD   177564
TXBUF:  .WORD   177566
DLEFLG: .WORD   0       ; BLANK COMP EXPANSION FLAG
TXUNIT: .WORD   0       ; UNIT TABLE ADDRESS OF IO IN PROGRESS
TXLENG: .WORD   0       ; NUMBER OF BYTES LEFT TO BE SENT TO TERMINAL
TXBUFA: .WORD   0       ; BYTE ADDRESS OF NEXT CHAR TO SEND
TXCHAR: .WORD   -1      ; HIGH PRIORITY CHAR TO SEND...FROM TTYOUT

TXOFST  = 4
TXSTRT: ; THIS CODE STARTS IO'S TO THE CONSOLE DEVICE
        ; ACCORDING TO STANDARD IO.OPS PROTOCOL
        TST     TXUNIT          ; SEE IF AN IO ALREADY IN PROGRESS
        BNE     TXSTRT          ; IF SO LOOP UNTIL THE IO IS COMPLETE
        MTPS    #200            ; NO INTERRUPTS PLEASE
        MOV     R1,TXUNIT       ; MARK HANDLER AS BUSY
        BIS     #BSYBIT,(R1)    ; MARK LOGICAL UNIT AS BUSY
        CLR     (SP)            ; SET UP RETURN STUFF ON STACK...PR-0 PS
        TST     (R3)+           ; SKIP R3 OVER IO INFO WORD
        MOV     R3,-(SP)        ; NOW THE RETURN ADDRESS
        CMP     R1,#UNITBL+14   ; IS THE WRITE TO SYSTERM??
        BNE     1$              ; IF NOT THEN LEAVE OUTPUT FLAGS ALONE
        CLR     SFLAG           ; ELSE CLEAR THE STOP FLAG
        CLR     OFLAG           ; AND FLUSH FLAG
1$:     TST     OFLAG           ; IF OUTPUT TO BE FLUSHED?
        BNE     TXQUIT          ; IF SO THEN MARK IO AS COMPLETE NOW
        MOV     <UBUFFR+TXOFST>(SP),TXBUFA      ; GRAB USER BUFFER ADDR
        MOV     <URLENG+TXOFST>(SP),TXLENG      ; AND REQUESTED IO LENGTH
TX$INT: BIC     #100,@TXCSR     ; NO INTERRUPTS PLEASE
        TST     TXCHAR          ; SEE IF ANY CHARS TO SEND RIGHT NOW
        BMI     NOCHAR          ; IF NEG THEN NO CHAR TO SEND
        MOV     R0,-(SP)        ; STASH A REG FOR NULLING
```

```
        MOV     TXCHAR,R0       ; GRAB THE ACTUAL CHAR
        MOVB    R0,@TXBUF
        CMPB    R0,#CR          ; IS IT ACR (MUST FILL AND LF)
        BNE     1$
        JSR     PC,NULLER       ; SEND STUFF FOR FILL COUNT
1$:     COM     TXCHAR          ; FLIP NEG BIT...NO DATA ANYMORE
        MOV     (SP)+,R0        ; RESTORE
        BIS     #100,@TXCSR     ; OK TO INTERRUPT NOW
        JMP     INTRTN          ; AND GO ON AS IF NOTHING HAPPENED
NOCHAR: TST     TXUNIT          ; ANY IO'S IN PROGRESS
        BEQ     TXEXIT          ; IF NOT JUST FORGET IT
        TST     TXLENG          ; ANY CHARS LEFT TO BE SENT?
        BEQ     TXQUIT          ; IF NOT THEN FINISH UP IO
        TST     SFLAG           ; SEE IF OUTPUT IS STOPPED
        BEQ     1$              ; IF NOT THEN SKIP THE WAITING STATE
        MTPS    #0              ; LEAVE CRITICAL REGION FOR DEVICE
        MTPS    #200            ; BACK UP TO PR-4
        BR      NOCHAR          ; AND GO TEST SFLAG AGAIN
1$:     MOV     R0,-(SP)        ; STASH REG
        CLR     R0
        BISB    @TXBUFA,R0      ; GRAB CHAR FROM USER BUFFER
        TST     DLEFLG          ; IS THIS CHAR THE DLE BLANK COUNT?
        BEQ     3$
        CLR     DLEFLG
        SUB     #32.,R0         ; NORMALIZE TO ACTUAL BLANK COUNT
        BLE     5$              ; DO NOTHING...SEND NULL
        MTPS    #0              ; BUSY WAIT LOOP
6$:     TSTB    @TXCSR
        BPL     6$
        MOVB    #' ,@TXBUF
        SOB     R0,6$
        MTPS    #200
3$:     CMPB    R0,#DLE         ; IS CHAR = DLE?
        BNE     4$
        COM     DLEFLG
5$:     CLR     R0
4$:     MOVB    R0,@TXBUF       ; SEND CHAR TO DL
        CMPB    R0,#CR          ; IS IT A CR??
        BNE     2$              ; IF NOT THEN....SKIP
        JSR     PC,NULLER       ; SEND FILL AND LF
2$:     MOV     (SP)+,R0        ; RESTORE TEMP REG
        BIS     #100,@TXCSR     ; ENABLE FOR NEXT COMPLETE
        INC     TXBUFA          ; BUMP BUFFER POINTER TO NEXT CHAR
        DEC     TXLENG          ; ALSO REFLECT ONE FEWER CHAR TO SEND
        JMP     INTRTN          ; THIS STRUCTURE IMPLIES AN IO IS NOT
                                ; DONE UNTIL THE LAST INTERRUPT IS RECEIVED
TXQUIT: BIC     #BSYBIT,@TXUNIT ; CLEAR BUSY BIT IN IO UNIT TABLE
        CLR     TXUNIT          ; MARK HANDLER AS NOT BUSY NOW
TXEXIT: JMP     INTRTN          ; AND BACK NOW TO WHEREVER

NULLER: ; HANDY SUBROUTINE FOR NULL FILLING AND LF AFTER CR
        ; ASSUME R0 SCRATCH (WELL...=CR) AND INTERRUPTS DISABLED
        MTPS    #0
        .IF     NDF,TERAK
        MOVB    FILCNT,R0       ; GRAB NILL COUNT (IF ANY)
        BEQ     2$
1$:     TSTB    @TXCSR          ; HANG UNTIL DL READY
        BPL     1$
        CLRB    @TXBUF
        SOB     R0,1$
```

```
        .ENDC
 2$:    TSTB    @TXCSR          ; HANG UNTIL READY FOR LF SEND
        BPL     2$
        MOVB    #LF,@TXBUF
        MTPS    #200
        RTS     PC
        .PAGE
        .CSECT  TABLES

        .BLKW   128.    ; ROOM FOR OP XFER TABLE

 UNITBL: .WORD  0,0,0   ; UNIT 0 NOT USED
        .WORD   INBIT!OUTBIT,TRMSTRT,TRMABRT
        .WORD   INBIT!OUTBIT,TRMSTRT,TRMABRT
        .REPT   <MAXUNT-2>
        .WORD   0,0,0
        .ENDR

        .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                    F     I     N     I     S                    |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp IOTR.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp LP.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "lp.mac")

        .TITLE  LP-11 PRINTER HANDLER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       SYSTEM TABLE CONTENTS                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=200
        LP$INT
        200

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   6
        .BLKW   3
        .ENDR
        .WORD   OUTBIT,LPSTRT,LPABRT
        .PAGE
        .CSECT  LPDRVR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;               PRINTER OUTPUT HANDLER                                 ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LPUNIT: .WORD   0
LPBUFA: .WORD
LPLENG: .WORD
DLEFLG: .WORD   0
LP.COL: .WORD   0
LPCSR:  .WORD   177514
LPBUF:  .WORD   177516
CRFLG:  .BYTE   0
LFFLG:  .BYTE   0

LPOFST  = 4
LPSTRT: ; THIS CODE STARTS IO'S TO THE PRINTER DEVICE
```

```
        TST     LPUNIT          ; SEE IF AN IO ALREADY IN PROGRESS
        BNE     LPSTRT          ; IF SO LOOP UNTIL THE IO IS COMPLETE
        MTPS    #200            ; NO INTERRUPTS PLEASE
        MOV     R1,LPUNIT       ; MARK HANDLER AS BUSY
        BIS     #BSYBIT,(R1)    ; MARK LOGICAL UNIT AS BUSY
        CLR     (SP)            ; SET UP RETURN STUFF ON STACK...PR-0 PS
        TST     (R3)+           ; SKIP R3 OVER IO INFO WORD
        MOV     R3,-(SP)        ; NOW THE RETURN ADDRESS
        MOV     <UBUFFR+LPOFST>(SP),LPBUFA      ; GRAB USER BUFFER ADDR
        MOV     <URLENG+LPOFST>(SP),LPLENG      ; AND REQUESTED IO LENGTH
LP$INT: BIC     #100,@LPCSR     ; DISABLE INTERRUPTS
        TST     LPUNIT          ; ANY IO'S IN PROGRESS
        BEQ     LPEXIT          ; IF NOT JUST FORGET IT
        TST     LPLENG          ; ANY CHARS LEFT TO BE SENT?
        BEQ     LPQUIT          ; IF NOT THEN FINISH UP IO
        MOV     R0,-(SP)        ; STASH REG
        CLR     R0
        TST     CRFLG           ; IF LFFLG AND CRFLG NOT SET THEN
        BEQ     9$              ; NO EXTRA LF OR NULL IS SENT.
        BPL     8$              ; IF LFFLG SET THEN
        CLRB    LFFLG           ; CLEAR LFFLG AND
        MOVB    #LF,R0          ; SEND A LF.
        BR      7$
8$:     CLRB    CRFLG           ; CRFLG SET SO CLEAR CRFLG AND
        MOVB    #00,R0          ; SEND A NULL.
7$:     DEC     LPBUFA          ; NO CHARACTER IS TAKEN FROM UBUFFR SO
        INC     LPLENG          ; ADJUST THESE TWO VALUES ACCORDINGLY.
        BR      2$
9$:     BISB    @LPBUFA,R0      ; GRAB CHAR FROM USER BUFFER
        BEQ     3$              ; A NULL? RESET TABS STOPS
        TST     DLEFLG          ; DLE EXPANSION IN PROGRESS?
        BEQ     10$             ; IF =0 THEN NORMAL STATE
        CLR     DLEFLG
        SUB     #32.,R0
        BLE     12$
11$:    TSTB    @LPCSR
        BPL     11$
        MOVB    #' ,@LPBUF
        INC     LP.COL
        SOB     R0,11$
        BR      2$
10$:    CMPB    R0,#DLE
        BNE     15$
        COM     DLEFLG
12$:    CLR     R0
        BR      2$
15$:    CMPB    R0,#FF          ; FORM - FEED?
        BEQ     3$
        CMPB    R0,#HT
        BNE     4$
        JSR     PC,LPTABR
4$:     INC     LP.COL
        CMPB    R0,#CR          ; AN END-OF-LINE CHAR?
        BNE     1$
        COMB    LFFLG           ; SET LFFLG AND
        MOVB    #CR,R0          ; SEND A CR.
        BR      3$
1$:     CMPB    R0,#21          ; A DC1??  (USED TO DO UNDERLINING)
        BNE     2$
        COMB    CRFLG           ; SET CRFLG AND
```

```
         MOVB    #CR,R0          ; SEND A CR.
 3$:     CLR     LP.COL
 2$:     TSTB    @LPCSR          ; TEST DONE BIT
         BPL     2$              ; AND HANG TIL READY
 13$:    TST     @LPCSR          ; TEST ERROR BIT
         BMI     13$             ; AND HANG TIL READY
         MOVB    R0,@LPBUF       ; SEND CHAR TO DL
         MOV     (SP)+,R0        ; RESTORE TEMP REG
         BIS     #100,@LPCSR     ; ALLOW INTERRUPT
         INC     LPBUFA          ; BUMP BUFFER POINTER TO NEXT CHAR
         DEC     LPLENG          ; ALSO REFLECT ONE FEWER CHAR TO SEND
         JMP     @#INTRTN        ; THIS STRUCTURE IMPLIES AN IO IS NOT
                                 ; DONE UNTIL THE LAST INTERRUPT IS RECEIVED
 LPQUIT: BIC     #BSYBIT,@LPUNIT ; CLEAR BUSY BIT IN IO UNIT TABLE
         CLR     LPUNIT          ; MARK HANDLER AS NOT BUSY NOW
 LPEXIT: JMP     INTRTN          ; AND BACK NOW TO WHEREVER

 LPTABR: ; LITTLE SUBROUTINE TO TAB
         MOV     R1,-(SP)
         MOV     #' ,R0
         MOV     LP.COL,R1
         BIS     #7,LP.COL
         SUB     LP.COL,R1
         BEQ     3$
 1$:     MOVB    R0,@LPBUF
 2$:     TSTB    @LPCSR
         BPL     2$
         INC     R1
         BNE     1$
 3$:     MOV     (SP)+,R1
         RTS     PC

 LPABRT: MTPS    #200
         TST     LPUNIT
         BEQ     1$
         BIC     #BSYBIT,@LPUNIT
         CLR     LPUNIT
 1$:     MTPS    #0
         RTS     PC

         .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                    F     I     N     I     S                    |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp LP.MAC
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 Interp LSI.MAC
##########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "lsi.mac")

 LSI=1

; +---------------------------------------------------------------------+
; |                                                                     |
; |                      F     I     N     I     S                      |
; |                                                                     |
; +---------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp LSI.MAC
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 Interp Macs.MAC
##########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "macros.mac")


        .NLIST
        .NLIST  CND
        .NLIST  TTM
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;
        .LIST

;***********************************************;
;*                                             *;
;*    UCSD PASCAL INTERPRETER FOR PDP-11'S     *;
;*                                             *;
;*    INSTITUTE FOR INFORMATION SYSTEMS        *;
;*    UC SAN DIEGO, LA JOLLA, CA  92093        *;
;*                                             *;
;*    KENNETH L. BOWLES, DIRECTOR              *;
;*                                             *;
;***********************************************;

        .NLIST
PC=%7                   ;PDP-11 PROGRAM COUNTER
SP=%6                   ;PDP-11 AND P-11 STACK POINTER
IPC=%4                  ;P-11 PROGRAM COUNTER
MP=%5                   ;BASE OF LOCAL DATA SEGMENT
BASE=%3                 ;BASE OF GLOBAL DATA SEG
BK=%2                   ;USED TO GET TO BACK FOR NEXT OPCODE FETCH

MAXUNT = 12.    ; MAX LEGAL UNIT # IN SYSTEM
MAXSEG = 15.    ; MAX SEGMENT NUMBER ALLOWED
NP = 50         ; INITIAL HEAP TOP...SET BY LINKER MAGICALLY

.IF     DF,TERAK
        LSI=1
        EIS=1
        FPI=1
        .GLOBL  DRAWLINE,DRAWBLOCK
.ENDC

.IF     NDF,FPI
        .GLOBL  $ADR,$SBR,$MLR,$DVR,$CMR
.ENDC
```

```
        .GLOBL  HLTLIN,BRKPTS,BUGSTA
        .GLOBL  $IR,$RI,ALOG,ALOG10,EXP,SIN,COS,ATAN,SQRT
        .GLOBL  ENTFP,XITFP,LOTIME,HITIME
        .GLOBL  MEMTOP,CRTNFO,GDIRP,INTRTN
        .GLOBL  BACK,SYSUNT,DIV,MLI,STKBAS,IORSLT
        .GLOBL  JTAB,SEGTBL,LASTMP,SEG,UNITBL,BITTER
        .GLOBL  SBROPS,UBROPS


BLANK=40
BLANKS=20040    ;TWO ASCII BLNKS
BS=10           ;BACKSPACE
CR=15
LF=12
HT=11
EM=31
FS=34
US=37
FF=14
GS=35
VT=13
RS=36
DLE=20
DC1=21
DC2=22

; TRAP PARAMETERS ( >=0 ARE EXECERR, <0 ARE SYSTEM REQUESTS)
SYSERR=0
INVNDX=1
NOPROC=2
NOEXIT=3
STKOVR=4
INTOVR=5
DIVZER=6
BADMEM=7
UBREAK=10
SYIOER=11
UIOERR=12
NOTIMP=13
FPIERR=14
S2LONG=15
HLTBPT=16
BRKPNT=17
TTXREQ=-1

NIL = 170001

;;;; CODE SEGMENT FORMAT DEFINES
; R@JTAB IS PROC# (LOW BYTE) AND LL (HIGH BYTE)

ENTRIC  = -2    ; JTAB INDEX OF ENTRY OFFSET
EXITIC  = -4    ;   "    "    "   EXIT POINT
PARMSZ  = -6    ;   "    "    "  # WORDS OF PARAMS TO COPY AT ENTRY TIME
DATASZ  = -10   ;   "    "    "  # WORDS TO OPEN IN STACK

;;;; MARK STACK CONTROL WORD FORMAT
; THESE OFFSETS ARE RELATIVE TO THE STAT LINK WORD!

MSSTAT = 0      ; STATIC LINK...POINTS TO PARENTS STAT LINK
MSDYN  = 2      ; DYNAMIC LINK...POINTS TO CALLERS STAT LINK
```

```
MSIPC  = 10     ; ABSOLUTE MEM ADDR OF NEXT OPCODE IN CALLER
MSSEG  = 6      ;    "      "    "  OF SEG TABLE OF CALLER (LIKELY = SEG)
MSJTAB = 4      ;    "      "    " OF CALLER JTAB (PROCEDURE CODE INFO ETC)
MSSP   = 12     ; VALUE TO SET SP TO UPON EXIT
MSBASE = -2     ; BASE REG...ONLY IN BASE MSCW'S
MSDLTA = 12     ; SIZE OF MSCW - 2

;;;; IO SUBSYSTEM STUFF
; BIT FIELDS IN UNITBL
; IO RESULTS GIVEN BY IO ROUTINES

PARERR  = 1
UNTERR  = 2
MODERR  = 3
INBIT   =  20000
OUTBIT  =  40000
BSYBIT  = 100000
UNOWAIT = 0
UBLOCK  = 2
URLENG  = 4
UBUFFR  = 6
UUNIT   = 10

.MACRO  GETNEXT STUFF
        .IF     B,<STUFF>
        MOVB    (IPC)+,R0               ;GET A BYTE FROM CODE
        .IFF
        MOVB    (IPC)+,STUFF            ;AND PUT IT IN R0 OR
        .ENDC                   ;IN STUFF IF STUFF I S NON-BLANK
.ENDM   GETNEXT

.MACRO  GETBYTE STUFF
        .IF     B,<STUFF>
        CLR     R0
        BISB    (IPC)+,R0
        .IFF
        CLR     STUFF
        BISB    (IPC)+,STUFF
        .ENDC
.ENDM   GETBYTE

.MACRO  GETBIG  STUFF,?NOTBIG
        .IF     B,<STUFF>
        GETNEXT
        BPL     NOTBIG
        BIC     #SIGNWIPE,R0
        SWAB    R0
        BISB    (IPC)+,R0
        .IFF
        GETNEXT STUFF
        BPL     NOTBIG
        BIC     #SIGNWIPE,STUFF
        SWAB    STUFF
        BISB    (IPC)+,STUFF
        .ENDC
NOTBIG:
.ENDM   GETBIG

R0=%0   ; DEFINE WORKING REGISTERS
R1=%1
```

```
 R2=%2
 R3=%3
 R4=%4
 R5=%5

 .MACRO  MORE
         MOV     BK,PC   ;SET PC TO BACK LABEL ADDR
 .ENDM   MORE

 .MACRO  WORDBOUND
         INC     IPC             ;BUMP IPC
         BIC     #1,IPC          ;THEN ROUND
 .ENDM   WORDBOUND

 SIGNWIPE=177600
 CLREXT=177400

 .IF     NDF,LSI
 .MACRO  MTPS    NEWPS,?L
         MOV     NEWPS,-(SP)
         MOV     #L,-(SP)
         RTI
 L:
 .ENDM   MTPS

 .IF     NDF,EIS
 .MACRO  SOB     REG,LABEL
         DEC     REG     ; THIS IS AN SOB OPERATOR
         BNE     LABEL   ; AS IN LSI-11 OR 11-40
 .ENDM   SOB
 .ENDC
 .ENDC

 .MACRO  .TTYOUT .CHAR
         .IIF    NB,<.CHAR>,     MOVB    .CHAR,R0
         TRAP    TTXREQ
 .ENDM   .TTYOUT

         .LIST

; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp Macs.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp MainOp
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "mainop.mac")


        .TITLE  MAIN OPERATORS
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;
        .CSECT  MAINOP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       MAIN OPERATORS                                 ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; FIRST ARE THE SHORT FORM LDL,LDO, AND IND OPS.
; THESE ARE THE MOST COMMON OPS (EXCEPT LDCI) AND RUN
; VERY FAST.  EACH DISP VALUE FOR THESE IS A NEW OPCODE

; SHORT LOCAL LOADS...16 OF THEM
SLDLS:  .IRP    N,<1,2,3,4,5,6,7,10,11,12,13,14,15,16,17,20>
        MOV     <N+N>+MSDLTA(MP),-(SP)
        BR      BACK
        .ENDR

; SHORT LOAD GLOBALS...16 OF THEM
SLDOS:  .IRP    N,<1,2,3,4,5,6,7,10,11,12,13,14,15,16,17,20>
        MOV     <N+N>+MSDLTA(BASE),-(SP)
        BR      BACK
        .ENDR

; SHORT IND OPS...8 OF THEM
SINDS:  MOV     @(SP)+,-(SP)
        BR      BACK
        .BLKW   2       ; FUNNY BUSINESS FOR EXTRA FAST IND 0
        .IRP    N,<1,2,3,4,5,6,7>
        ADD     #<N+N>,@SP
        MOV     @(SP)+,-(SP)
        BR      BACK
        .ENDR
        .PAGE
;;;;;;;;;;;;;;;;;;;;;;
; MAIN INTERPRETER LOOP
; GO HERE FOR OPCODE
; FETCH SEQUENCE
```

```
;;;;;;;;;;;;;;;;;;;;;;

SLDCI:  MOV     R0,-(SP)        ; PUSH THE LIT VALUE AND FALL INTO NEXT OP
BACK:   GETNEXT                 ; GET NEXT INSTRUCTION BYTE
        BPL     SLDCI           ; IF POSITIVE THEN A SHORT LDCI
        ASL     R0              ; DOUBLE FOR WORD INDEXING
        MOV     XFRTBL(R0),PC   ; TRANSFER CONTROL TO PROPER OP

ABI:    ; INTEGER ABSOLUTE VALUE
        TST     @SP
        BPL     1$
        NEG     @SP
        BPL     1$
        CLR     @SP
1$:     MORE

ABR:    ; REAL ABSOLUTE VALUE
        BIC     #100000,@SP
        MORE

ADI:    ; ADD INTEGER
        ADD     (SP)+,@SP
        MORE

ADR:    ; ADD REAL
        .IF     DF,FPI
        FADD    SP
        MORE
        .IFF
        JSR     R4,ENTFP
        .WORD   $ADR,XITFP
        .ENDC

AND:    ; LOGICAL AND
        COM     @SP
        BIC     (SP)+,@SP
        MORE

BPT:    ; CONDITIONAL HALT (BREAKPOINT)
        GETBIG                  ; LINE IN LIST FILE
        MOV     R0,HLTLIN
        CMP     BUGSTA,#3
        BGE     BPTTRP
        ; NOT IN STEPPING MODE, SO SEE IF MATCHES A BREAKPOINT
        MOV     #BRKPTS,R1
        CMP     (R1)+,R0
        BEQ     BPTTRP
        CMP     (R1)+,R0
        BEQ     BPTTRP
        CMP     (R1)+,R0
        BEQ     BPTTRP
        CMP     (R1),R0
        BEQ     BPTTRP
        MORE
BPTTRP: TRAP    BRKPNT

DIF:    ; SET DIFFERENCE
        JSR     PC,SETADJ
        BEQ     2$
1$:     BIC     (SP)+,(R0)+
```

```
        SOB     R1,1$

2$:     MORE

DVI:    ; INTEGER DIVIDE
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        JSR     PC,DIV
        MOV     R0,-(SP)
        MORE

DVR:    ; REAL DIVIDE
        .IF     DF,FPI
        FDIV    SP
        MORE
        .IFF
        JSR     R4,ENTFP
        .WORD   $DVR,XITFP
        .ENDC

CHK:    ; CHECK INDEX OR RANGE
        CMP     (SP)+,2(SP)     ; CHECK MAXIMUM VALUE
        BLT     CHKERR
        CMP     (SP)+,@SP       ; CHECK MINIMUM VALUE
        BGT     CHKERR
        MORE
CHKERR: TRAP    INVNDX

FLO:    ; FLOAT NEXT TO TOP-OF-STACK
        MOV     (SP)+,FLO1      ; SAVE REAL ON TOS
        MOV     (SP)+,FLO0
        JSR     R4,ENTFP
        .WORD   $IR,FIXTOS,XITFP
FIXTOS: MOV     (PC)+,-(SP)
FLO0:   .WORD
        MOV     (PC)+,-(SP)
FLO1:   .WORD
        JMP     @(R4)+

FLT:    ; FLOAT TOP-OF-STACK
        JSR     R4,ENTFP
        .WORD   $IR,XITFP

INN:    ; SET INCLUSION
        MOV     (SP)+,BK        ; GET SET SIZE FROM STACK
        MOV     SP,R0           ; NOW POINT R0 AT THE SCALAR VAL
        ADD     BK,R0           ; BY SKIPPING IT ABOVE
        ADD     BK,R0           ; THE SET
        MOV     @R0,R1          ; R1 HAS THE VALUE TO TEST FOR NOW
        BMI     NOTINN          ; NO NEGATIVE SET INDEXES
        .IF     DF,EIS
        ASH     #-4,R1
        .IFF
        ASR     R1
        ASR     R1
        ASR     R1
        ASR     R1
        .ENDC

        CMP     R1,BK           ; CHECK IF ENOUGH WORD ARE IN SET
```

```
        BGE     NOTINN          ; TO ACCOMODATE THE VALUE IN R1
        ASL     R1              ; IF THERE ARE, POINT R1 AT THE WORD
        ADD     SP,R1           ; WHICH HAS THE BIT IN IT
        MOV     @R1,BK          ; PLACE THE WORD INTO BK FOR LATER
        MOV     @R0,R1          ; GET THE SCALAR AGAIN
        BIC     #177760,R1      ; CHUCK ALL BUT LOW 4 BITS
        ASL     R1              ; MAKE A WORD INDEX INTO BITTER
        BIT     BITTER(R1),BK   ; TEST IF THE BIT IN QUESTION IS ON
        BEQ     NOTINN
        MOV     R0,SP           ; FOUND IT...CUT BACK STACK
        MOV     #1,@SP          ; PUT A TRUE ON TOP
XITINN: MOV     #BACK,BK        ; RESTORE REGISTER
        MORE
NOTINN: MOV     R0,SP           ; CUT BACK HERE TOO
        CLR     @SP             ; EXCEPT PUSH A FALSE
        BR      XITINN


INT:    ; SET INTERSECTION
        JSR     PC,SETADJ
        MOV     R1,TOPSIZ       ; SAVE TOP SET SIZE
        BEQ     2$
1$:     COM     @SP
        BIC     (SP)+,(R0)+
        SOB     R1,1$
2$:     MOV     @SP,R1          ; GET FINAL SET SIZE
        SUB     TOPSIZ,R1       ; SUBTRACT THE TOP SIZE...R1 = DIFF
        BEQ     4$              ; IF NO LEFTOVER WORDS THEN EXIT
3$:     CLR     (R0)+           ; ELSE CLEAR EXTRA WORDS IN FINAL SET
        SOB     R1,3$
4$:     MORE
TOPSIZ: .WORD           ; SIZE OF TOP SET (TEMP)

IOR:    ; LOGICAL OR
        BIS     (SP)+,@SP
        MORE

MOD:    ; INTEGER REMAINDER DIVIDE
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        JSR     PC,DIV
        MOV     R1,-(SP)
        MORE

MPI:    ; INTEGER MULTIPLY
        MOV     (SP)+,R0
        MOV     (SP)+,R1
        JSR     PC,MLI
        MOV     R0,-(SP)
        MORE

MPR:    ; REAL MULTIPLY
        .IF     DF,FPI
        FMUL    SP
        MORE

        .IFF
        JSR     R4,ENTFP
        .WORD   $MLR,XITFP
        .ENDC
```

```
NGI:    ; INTEGER NEGATION
        NEG     @SP
        MORE

NGR:    ; REAL NEGATION
        TST     @SP
        BEQ     1$
        ADD     #100000,@SP
1$:     MORE

NOT:    ; LOGICAL NOT
        COM     @SP
        MORE

SRS:    ; BUILD SUBRANGE SET
        MOV     (SP)+,R0        ; GRAB HIGHER VALUE J OF I..J
        MOV     (SP)+,R1        ; AND LOWER VALUE I
        BMI     NULSET          ; IF I IS NEG THEN NULL SET TIME
        CMP     R1,R0           ; IF I > J THEN
        BGT     NULSET          ; ALSO A NULL SET
        MOV     #1,SETWDS       ; FINAL SET SIZE...START WITH 1 WORD
        MOV     #177777,-(SP)   ; OF ALL ONES
        MOV     R0,BK           ; CLEAR HIGH BITS 15 DOWNTO J
        BIC     #177760,BK      ; USE LOW BITS IN BK FOR CLRMSK INX
        ASL     BK              ; DOUBLE FOR WORDS INDEX
        BIC     CLRMSK+2(BK),@SP ; HIGH ORDER BITS GONE NOW
        BIS     #17,R0          ; FIND WORDS TO PUT BETWEEN I..J
        SUB     R1,R0           ; HAVE DIFFERENCE NOW * 16
        .IF     DF,EIS
        ASH     #-4,R0          ; DIV 16...NUMBER WORDS FROM I..J
        .IFF
        ASR     R0
        ASR     R0
        ASR     R0
        ASR     R0
        .ENDC
        BEQ     2$              ; IF ZERO, THEN 1 WORD IS ENOUDH
        ADD     R0,SETWDS       ; ELSE BUMP SET SIZE COUNTER
1$:     MOV     #177777,-(SP)   ; AND PUSH ALL BIT SET WORDS
        SOB     R0,1$           ; FOR NUMBER WORDS DIFFERENCE
2$:     MOV     R1,BK           ; NOW ZAP LOW BITS ON TOS WORD
        BIC     #177760,BK      ; THAT ARE LESS THAN I VALUE
        ASL     BK              ; WORD INDEX
        MOV     CLRMSK(BK),BK   ; GRAB HIGH ORDER CLEARING BIT MASK
        COM     BK              ; CHANGE TO LOW ORDER MASK
        BIC     BK,@SP          ; NOW THE ON BITS IN SET ARE OK
        .IF     DF,EIS
        ASH     #-4,R1          ; DIV 16...# OF ZERO TO PUSH NOW
        .IFF
        ASR     R1
        ASR     R1
        ASR     R1
        ASR     R1
        .ENDC
        BEQ     4$              ; IF NO MORE ZEROES THEN SKIP
        ADD     R1,SETWDS       ; ELSE ADD ON ZERO COUNT TO SET SIZE
3$:     CLR     -(SP)           ; AND LOOP ADDING ON ZEROES
        SOB     R1,3$
4$:     MOV     SETWDS,-(SP)    ; PUSH SET SIZE...NOW GOOD, CLEAN SET ON STACK
```

```
        MOV     #BACK,BK
        MORE
SETWDS: .WORD   ; SIZE OF SET BUILD ABOVE STUCK HERE

SBI:    ; INTEGER SUBTRACT
        SUB     (SP)+,@SP
        MORE

SBR:    ; REAL SUBTRACT
        .IF     DF,FPI
        FSUB    SP
        MORE
        .IFF
        JSR     R4,ENTFP
        .WORD   $SBR,XITFP
        .ENDC

        ; SGS IS BELOW THE SQUARE OP

SQI:    ;  SQUARE INTEGER
        MOV     @SP,-(SP)
        BR      MPI

SQR:    ; SQUARE REAL
        MOV     2(SP),-(SP)
        MOV     2(SP),-(SP)
        BR      MPR

NULSET: CLR     -(SP)           ; ZERO WORD SET SIZE
        MORE

SGS:    ; MAKE SINGLETON SET
        MOV     (SP)+,R0        ; GET THE SCALAR VALUE WANTED
        BMI     NULSET          ; IF NEGATIVE THEN GO BUILD A NULL SET
        CLR     -(SP)           ; PUT A WORD TO SET BIT INN
        MOV     R0,R1           ; NOW SET PROPER BIT IN TOS
        BIC     #177760,R1      ; ZAP ALL BUT LOW 4 BITS
        ASL     R1              ; MAKE A WORD INDEX IN BITTER
        BIS     BITTER(R1),@SP  ; NOW WE HAVE PROPER BIT SET
        BIC     #170017,R0      ; ZAP ALL BUT WORD BITS
        BEQ     2$              ; IF NO ZEROES NEEDED THEN DONE
        .IF     DF,EIS
        ASH     #-4,R0
        .IFF
        ASR     R0
        ASR     R0
        ASR     R0
        ASR     R0
        .ENDC
        MOV     R0,R1           ; SAVE WORD COUNT FOR LATER PUSH
1$:     CLR     -(SP)           ; CLEAR A STACK WORD
        SOB     R1,1$
2$:     INC     R0              ; SET R0 TO TOTAL SET SIZE
        MOV     R0,-(SP)        ; AND PUSH IT FINALLY
        MORE

STO:    ; STORE INDIRECT
        MOV     (SP)+,@(SP)+
        MORE
```

```
IXS:    ; STRING INDEX...DYNAMIC RANGE CHECK
        MOV     @SP,R0          ; GRAB INDEX VALOUE
        BEQ     IXSERR          ; ZERO INDEX IS AN ERROR
        CMP     R0,#255.        ; CHECK IF WAY TOO BIG
        BHI     IXSERR          ; BOMB IF SO
        CMPB    R0,@2(SP)       ; CHECK INDEX AGAINST STRING LENGTH
        BHI     IXSERR          ; AND BOMB FOR THAT TOO
        ADD     (SP)+,@SP       ; OK...ADD THE INDEX TO ADDR ON TOS
        MORE
IXSERR: TRAP    INVNDX

UNI:    ; SET UNION
        JSR     PC,SETADJ
        BEQ     2$
1$:     BIS     (SP)+,(R0)+
        SOB     R1,1$
2$:     MORE

S2P:    ; STRING TO PACKED ARRAY CONVERT
        INC     2(SP)
        MORE

LDCN:   ; LOAD CONSTANT NIL
        .IIF    EQ,NIL,         CLR     -(SP)
        .IIF    NE,NIL,         MOV     #NIL,-(SP)
        MORE

ADJ:    ; SET ADJUST
        GETBYTE                 ; GRAB REQUESTED SET SIZE
        MOV     (SP)+,R1        ; GET SET SIZE FROM TOS
        CMP     R1,R0           ; COMPARE SET SIZE TO REQ SIZE
        BLT     EXPAND          ; IF SET TOO SMALL THEN EXPAND IT
        BGT     CRUNCH          ; IF TOO BIG THEN CRUNCH THE SET
        MORE                    ; ELSE ALL'S OK...NEXT INSTRUCTION
CRUNCH: MOV     R0,BK           ; SAVE REQUESTED LENGTH
        ASL     R0              ; NOW POINT R0 AT TOP OF VALID PART OF SET
        ADD     SP,R0
        ASL     R1              ; POINT R1 ABOVE ENTIRE SET...IS DEST
        ADD     SP,R1           ; FOR FUTURE MOVES TO CRUNCH OUT JUNK
1$:     MOV     -(R0),-(R1)     ; COPY THE WORDS OF GOOD SEOT PART
        SOB     BK,1$
        MOV     R1,SP           ; R1 IS NEW TOS...CUT BACK STUFF
        BR      XITADJ
EXPAND: MOV     SP,BASE         ; REMEMBER TOP OF SMALL SET
        SUB     R1,R0           ; R0 HAS SET SIZE DIFFERENCE NOW
        MOV     R0,BK           ; SAVE DIFF FOR LATER ZEROING
        ASL     R0              ; DOUBLE FOR WORD COUNT
        SUB     R0,SP           ; ADD JUNK ONTO STACK POINTER FOR ZERO FILL
        MOV     SP,R0           ; NOW DEST FOR SET COPYING
        TST     R1              ; CHECK IF OLD SET SIZE = 0!!
        BEQ     2$              ; IF SO THEN DONT DO LOOP...SYSBOMB!
1$:     MOV     (BASE)+,(R0)+   ; COPY THE SET NOW
        SOB     R1,1$
2$:     CLR     (R0)+           ; NOW ZERO IN THE REST OF SET
        SOB     BK,2$
        MOV     STKBAS,BASE     ; RESTORE SCRATCH REG
XITADJ: MOV     #BACK,BK        ; RESTORE THIS TOO
        MORE

        ; FJP IS UP AHEAD WITH UJP
```

```
INC:    ; INCREMENT TOS BY PARAM
        GETBIG
        ADD     R0,@SP
        MORE

IND:    ; INDIRECT LOAD
        GETBIG
        ASL     R0
        ADD     R0,@SP
        MOV     @(SP)+,-(SP)
        MORE

IXA:    ; INDEX ARRAY
        GETBIG  R1              ; GET # WORDS PER ELEMENT
        MOV     (SP)+,R0        ; GRAB USER'S INDEX VALUE
        BEQ     2$              ; IF ZERO, THEN DONE ALREADY!
        CMP     R1,#1           ; CHECK IF 1 WORD ELS
        BEQ     1$              ; IF SO THEN NO MULTIPLY
        JSR     PC,MLI
1$:     ASL     R0              ; NOW DOUBLE INDEX VALUE FOR WORDS
        ADD     R0,@SP          ; NEW ADDRESS OFO ARRAY ELEMENT NOW
2$:     MORE

LAO:    ; LOAD GLOBAL ADDRESS
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     BASE,R0
        MOV     R0,-(SP)
        MORE

LCA:    ; LOAD CONSTANT (STRING) ADDRESS
        MOV     IPC,-(SP)
        GETBYTE                 ; GRAB STRING LENGTH
        ADD     R0,IPC          ; AND SKIP IPC PAST STRING
        MORE

LDO:    ; LOAD GLOBAL
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     BASE,R0
        MOV     @R0,-(SP)
        MORE

MOV:    ; MOVE WORDS
        GETBIG  BK              ; GRAB # WORDS TO MOVE (ALWAYS > 0)
        MOV     (SP)+,R0        ; SOURCE ADDRESS
        MOV     (SP)+,R1        ; DESTINATION ADDRESS
1$:     MOV     (R0)+,(R1)+     ; COPY EACH WORD
        SOB     BK,1$
        MOV     #BACK,BK
        MORE

MVB:    ; MOVE BYTES
        GETBIG  BK              ; GRAB # BYTES TO MOVE (ALWAYS > 0)
        MOV     (SP)+,R0        ; SOURCE ADDRESS
        MOV     (SP)+,R1        ; DESTINATION ADDRESS
1$:     MOVB    (R0)+,(R1)+     ; COPY EACH BYTE
```

```
        SOB     BK,1$
        MOV     #BACK,BK
        MORE

SAS:    ; STRING ASSIGNMENT
        MOV     (SP)+,R0        ; GET SOURCE STRING ADDRESS
        CMP     R0,#255.        ; CHECK IF ITS REALLY A CHAR
        BHI     1$              ; IF NOT THEN SKIP TRICKYNESS
        MOVB    R0,LITCHR+1     ; LIT CHAR...MAKE IT A STRING
        MOV     #LITCHR,R0      ; NOW R0 HAS GOOD ADDRESS
1$:     CMPB    @R0,(IPC)+      ; CHEOCK IF MAXLENG IS EXCEEDED BY SRC LENG
        BHI     SASERR          ; BOMB OUT IF SO
        MOV     (SP)+,R1        ; GRAB DESTINATION ADDRESS
        CLR     BK              ; SET UP LOOP COUNTER WITH SOURCE LENGTH
        BISB    @R0,BK          ; NOW BK HAS LENGTH COUNT OF SOURCE
        INC     BK              ; INCLUDE LENGTH BYTE IN LOOP COUNT
2$:     MOVB    (R0)+,(R1)+     ; COPY EACH BYTE
        SOB     BK,2$           ; LOOP FOR CHARS+LENGTH BYTE
        MOV     #BACK,BK        ; RESTORE
        MORE
LITCHR: .WORD   1               ; DUMMY STRING OF LENGTH 1
SASERR: TRAP    S2LONG

SRO:    ; STORE GLOBAL
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     BASE,R0
        MOV     (SP)+,@R0
        MORE

XJP:    ; INDEX JUMP
        WORDBOUND
        MOV     (SP)+,R0        ; GRAB INDEX VALUE FROM TOS
        MOV     (IPC)+,R1       ; GET MIN CASE INDEX FROM CODE
        CMP     R0,R1           ; SEE IF INDEX IS TOO SMALL
        BLT     MINERR          ; SKIP OUT IF NOT IN RANGE
        CMP     R0,(IPC)+       ; CHECK IF LEQ MAX VALUE
        BGT     MAXERR          ; SKIP OUT HERE TOO
        TST     (IPC)+          ; SKIP OVER ELSE JUMP WORD
        SUB     R1,R0           ; ADJUST INDEX TO 0..N
        ASL     R0              ; DOUBLE INDEX FOR WORD STUFF
        ADD     R0,IPC          ; POINT IPC AT PROPER JUMP TABLE INDEX
        SUB     @IPC,IPC        ; NOW IPC POINTS AT STATEMENT SELECTED
        MORE
MINERR: TST     (IPC)+          ; SKIP IPC TO ELSE JUMP LOCATION
MAXERR: MORE                    ; IPC POINTS AT ELSE JUMP...ONWARD

COMPAR: ; COMPARE COMPLEX THINGS
        ; RELOPS EQU, GRT, GEQ, LEQ, LES, & NEQ
        GETNEXT R1              ; GRAB COMPARISON TYPE
        MOV     CMPTBL(R1),PC   ; NOW TRANSFER TO PROPER CODE

REALCMP:; COMPARE REAL
        MOV     SBROPS(R0),1$
        .IF     DF,FPI
        FSUB    SP
1$:     NOP
        BR      2$
        TST     (SP)+
```

```
        MOV     #1,@SP
        MORE
2$:     TST     (SP)+
        CLR     @SP
        MORE
        .IFF
        JSR     R4,ENTFP
        .WORD   $CMR,1$,XITFP
1$:     NOP
        BR      2$
        MOV     #1,-(SP)
        JMP     @(R4)+
2$:     CLR     -(SP)
        JMP     @(R4)+
        .ENDC

STRGCMP:; COMPARE STRINGS
        MOV     UBROPS(R0),NOTEQL       ; SELF-MODIFY UNSIGNED BRANCH
        MOV     2(SP),R0        ; GET LEFT OPERAND ADDRESS
        CMP     R0,#255.        ; BUT IT MAY BE A CHAR!
        BHI     1$              ; IF SO, THEN PUT IN LITCHR TRICK
        MOVB    R0,LITCHR+1     ; TO GET A STRING OF 1 LENGTH
        MOV     #LITCHR,R0      ; AND POINT REGISTER AT IT
        MOV     R0,2(SP)        ; BE SURE TO FIX STACK TOO
1$:     MOV     @SP,R1          ; GRAB RIGHT SIDE ADDRESS
        CMP     R1,#255.        ; SAME LITCHR BUSINESS
        BHI     2$              ; AS ABOVE
        MOVB    R1,LITCHR+1
        MOV     #LITCHR,R1
        MOV     R1,@SP
2$:     CLR     BK              ; NOW GET LENG = MIN(LENGTH(R0),LENGTH(R1))
        CMPB    (R0)+,(R1)+     ; CHECK MIN LENG, POINT REGS AT TEXT
        BHIS    3$              ; IF LENG(R0) < LENG(R1) THEN
        BISB    -1(R0),BK       ; BK := LENGTH(R0)
        BR      4$
3$:     BISB    -1(R1),BK       ; ELSE BK := LENG(R1)
4$:     BEQ     EQUALS          ; BR IF RUN OFF END OF STRINGS (BK = 0)
        CMPB    (R0)+,(R1)+     ; COMPARE STRING CONTENTS
        BNE     NOTEQL          ; ANY NEQ CHAR STOPS CMP
        DEC     BK
        BR      4$              ; LOOP UNTIL OFF END
EQUALS: ; WELL, STRINGS = FOR MIN LENGTH...CMP LENGTHS
        CMPB    @2(SP),@0(SP)   ; LONGER STRING IS GREATER!
NOTEQL: NOP     ; CORRECT BR OP GOES HERE
        BR      2$              ; JUMP TO FALSE CASE
        MOV     #1,2(SP)        ; PLACE A TRUE IN STACK
1$:     TST     (SP)+           ; FINALLY RETURN
        MOV     #BACK,BK
        MORE
2$:     CLR     2(SP)           ; FALSE
        BR      1$

WORDCMP:; COMPARE WORDS
        GETBIG  BK
        ASL     BK
        BR      CMP.IT
BYTECMP:; COMPARE BYTE STRING
        GETBIG  BK
CMP.IT: MOV     UBROPS(R0),2$   ; PUT IN PROPER CMP OPERATOR
        MOV     (SP)+,R1        ; RIGHT HAND EXPRESSION ADDR
```

```
        MOV     (SP)+,R0        ; LEFT EXPRESSION
1$:     CMPB    (R0)+,(R1)+     ; COMPARE BYTES
        BNE     2$              ; ANY NEQ STOPS LOOP
        SOB     BK,1$
2$:     NOP
        BR      4$
        MOV     #1,-(SP)
3$:     MOV     #BACK,BK
        MORE
4$:     CLR     -(SP)
        BR      3$


BOOLCMP:; COMPARE BOOLEAN OPERANDS
        BIC     #177776,@SP
        BIC     #177776,2(SP)
        MOV     XFRTBL+40.(R0),PC       ; DO INTEGER COMPARE

POWRCMP:; COMPARE SETS
        JSR     PC,SETADJ       ; ENSURE SETS MAKE SENSE
        MOV     -(R0),BK        ; GET LOWER SET SIZE
        ADD     (R0)+,BK        ; DOUBLE FOR BYTE SIZE
        ADD     R0,BK           ; NOW BK POINTS AT FINAL TOP OF STACK
        MOV     BK,NEWSP
        MOVB    -2(IPC),BK      ; GRAB ORIGINAL INSTRUCTION BYTE
        ASL     BK              ; DOUBLE IT!! WORD INDEX IN XFRSET
        MOV     XFRSET(BK),-(SP) ; STASH TRANSFER ADDRESS...
        MOV     -2(R0),BK       ; ACTUAL OPS EXPECT BK=LOWER SET SIZE
        MOV     (SP)+,PC        ; TRANSFER NOW TO PROPER COMPARE OP

EQUS:   ; COMPARE SETS EQUAL
        TST     R1              ; NUMBER OF WORDS IN TOP SERT
        BEQ     CHKZER
1$:     CMP     (SP)+,(R0)+
        BNE     SFALSE
        DEC     BK
        SOB     R1,1$
CHKZER: TST     BK
        BEQ     STRUE
1$:     TST     (R0)+
        BNE     SFALSE
        SOB     BK,1$
        BR      STRUE

LEQS:   ; LESS THAN OR EQUAL SET COMPARE
        TST     R1
        BEQ     CHKZER
1$:     BIC     (SP)+,(R0)+
        BNE     SFALSE
        DEC     BK
        SOB     R1,1$
        BR      CHKZER

GEQS:   ; GREATER OR EQUAL SET COMPARE
        TST     R1
        BEQ     STRUE
1$:     BIC     (R0)+,(SP)+
        BNE     SFALSE
        SOB     R1,1$
        BR      STRUE
```

```
NEQS:   ; NOT EQUAL SET COMPARE
        TST     R1
        BEQ     2$
1$:     CMP     (SP)+,(R0)+
        BNE     STRUE
        DEC     BK
        SOB     R1,1$
2$:     TST     BK
        BEQ     SFALSE
3$:     TST     (R0)+
        BNE     STRUE
        SOB     BK,3$
SFALSE: MOV     NEWSP,SP
        CLR     -(SP)
XITPWR: MOV     #BACK,BK
        MORE
STRUE:  MOV     NEWSP,SP
        MOV     #1,-(SP)
        BR      XITPWR
NEWSP:  .WORD

LDA:    ; LOAD INTERMEDIATE ADDRESS
        GETNEXT                 ; THE DELTA LEX LEVEL
        MOV     MP,R1           ; POINT R1 AT STAT LINKS
1$:     MOV     @R1,R1          ; LINK DOWN NOW UNTIL
        SOB     R0,1$           ; DELTA LL = 0 (NEVER START AT 0)
        GETBIG                  ; GET DISPLACMENT
        ASL     R0              ; DOUBLE FOR WORD INDEXING
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     R1,R0           ; NOW R0 HAS ADDRESS
        MOV     R0,-(SP)        ; PUSH IT
        MORE

LDC:    ; LOAD MULTIWORD CONSTANT
        GETNEXT                 ; NUMBER OF WORDS TO LOAD (ALWAYS > 0)
        WORDBOUND
1$:     MOV     (IPC)+,-(SP)
        SOB     R0,1$
        MORE

LOD:    ; LOAD INTERMEDIATE VALUE
        GETNEXT                 ; THE DELTA LEX LEVEL
        MOV     MP,R1           ; POINT R1 AT STAT LINKS
1$:     MOV     @R1,R1          ; LINK DOWN NOW UNTIL
        SOB     R0,1$           ; DELTA LL = 0 (NEVER START AT 0)
        GETBIG                  ; GET DISPLACMENT
        ASL     R0              ; DOUBLE FOR WORD INDEXING
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     R1,R0           ; NOW R0 HAS ADDRESS
        MOV     @R0,-(SP)       ; COPY VALUE FROM STACK
        MORE

STR:    ; STORE INTERMEDIATE VALUE
        GETNEXT                 ; THE DELTA LEX LEVEL
        MOV     MP,R1           ; POINT R1 AT STAT LINKS
1$:     MOV     @R1,R1          ; LINK DOWN NOW UNTIL
        SOB     R0,1$           ; DELTA LL = 0 (NEVER START AT 0)
        GETBIG                  ; GET DISPLACMENT
        ASL     R0              ; DOUBLE FOR WORD INDEXING
```

```
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     R1,R0           ; NOW R0 HAS ADDRESS
        MOV     (SP)+,@R0       ; SAVE VALUE INTO STACK
        MORE

NOJUMP: INC     IPC             ; GO HERE IF A TRUE WAS ON STACK
        MORE

EFJ:    ; INTEGER = THEN FJP
        SUB     (SP)+,(SP)+
        BEQ     NOJUMP
        BR      UJP

NFJ:    ; INTEGER <> THEN FJP
        SUB     (SP)+,(SP)+
        BNE     NOJUMP
        BR      UJP

FJP:    ; BRANCH IF FALSE ON TOS
        ROR     (SP)+
        BCS     NOJUMP
        ; NOW FALL INTO UJP

UJP:    ; BRANCH UNCONDITIONAL
        GETNEXT                 ; GET BRANCH PARAM
        BMI     1$              ; IF < 0 THEN A LONG JUMP
        ADD     R0,IPC          ; ELSE JUST A BYTE OFFSET FORWARD
        MORE
1$:     MOV     JTAB,IPC        ; POINT IPC AT JTAB ENTRY SO OFFSET
        ADD     R0,IPC          ; IS GOOD...R0 IS < 0   REALLY A SUBTRACT
        SUB     @IPC,IPC        ; POINT IPC AT NEW OBJECT CODE
        MORE

LDP:    ; LOAD PACKED FIELD
        MOV     @4(SP),R0       ; GET WORD WHICH HAS FIELD IN IT INTO R0
        MOV     (SP)+,R1        ; GET FIELD RIGHT-MOST BIT NUMBER
        .IF     DF,EIS
        NEG     R1
        ASH     R1,R0
        .IFF
        BEQ     NOASR           ; IF ZERO THEN NO SHIFTS NEEDED
1$:     ASR     R0              ; SHIFT R0 UNTIL FIELD IN LOW BITS
        SOB     R1,1$
NOASR:
        .ENDC
        MOV     (SP)+,R1        ; GRAB FIELD WIDTH FROM STACK
        ASL     R1              ; DOUBLE IT FOR WORD INDEXING
        BIC     CLRMSK(R1),R0   ; CLEAR SHIT BITS IN WORD
        MOV     R0,@SP          ; NOW PUT FIELD ON STACK
        MORE

STP:    ; STORE PACKED FIELD
        MOV     4(SP),R1        ; GRAB FIELD WIDTH
        ASL     R1              ; DOUBLE FOR WORD INDEX
        MOV     CLRMSK(R1),R1   ; NOW WE HAVE A CLEARING MASK IN R1
        MOV     (SP)+,R0        ; GRAB INSERT VALUE FROM STACK
        BIC     R1,R0           ; ZAP JUNK BITS IN INSERT VALUE
        COM     R1              ; NOW R1 WILL ZAP THE FIELD ITSELF
        MOV     (SP)+,BK        ; GET FIELD RIGHT-MOST BIT
        .IF     DF,EIS
```

```
        ASH     BK,R0
        ASH     BK,R1
        .IFF
        BEQ     NOASL           ; IF IN RIGHT-MOST BIT THEN NO SHIFT
1$:     ASL     R0              ; SHIFT INSERT VALUE BY ONE
        ASL     R1              ; AND SHIFT CLEAR MASK
        SOB     BK,1$           ; AND DO SO UNTIL LINED UP WITH FIELD
NOASL:
        .ENDC
        TST     (SP)+           ; FORGET THE OLD FIELD WIDTH
        MOV     (SP)+,BK        ; BK NOW HAS ADDRESS OF PACKED FIELD WORD
        BIC     R1,@BK          ; SET FIELD IN WORD TO ZEROES
        BIS     R0,@BK          ; NOW OR IN THE INSERT VALUE
        MOV     #BACK,BK        ; RESTORE SCRATCH REG
        MORE

LDM:    ; LOAD MULTIPLE WORDS
        MOV     (SP)+,R1        ; GET WORD LIST ADDRESS
        GETBYTE                 ; AND GET WORD COUNT
        BEQ     NOLOAD          ; MAY HAPPEN SOMEDAY
        ADD     R0,R1           ; SKIP LIST ADDRESS TO UPPER END
        ADD     R0,R1           ; R1 NOW POINTS ABOVE DATA BLOCK
1$:     MOV     -(R1),-(SP)
        SOB     R0,1$
NOLOAD: MORE

STM:    ; STORE MULTIPLE WORDS
        GETBYTE                 ; GET NUMBER OF WORDS
        BEQ     NOSTOR
        MOV     SP,R1           ; POINT R1 AT DATA BLOCK ON STACK
        ADD     R0,R1           ; SKIP R1 PAST THE DATA TO GET THE
        ADD     R0,R1           ; STORE ADDRESS BELOW IT
        MOV     @R1,R1          ; GET STORE ADDRESS NOW
1$:     MOV     (SP)+,(R1)+
        SOB     R0,1$
NOSTOR: TST     (SP)+           ; CHUCK ADDRESS WORD
        MORE

LDB:    ; LOAD BYTE
        MOV     @SP,R0
        CLR     @SP
        BISB    @R0,@SP
        MORE

STB:    ; STORE BYTE
        MOVB    (SP)+,@(SP)+
        MORE

IXP:    ; INDEX PACKED ARRAY
        GETNEXT R1              ; GET # ELEMENTS PER WORD
        MOV     (SP)+,R0        ; GET USER'S INDEX VALUE
        JSR     PC,DIV          ; NOW DIVIDE OUT WORD INX AND BIT INX
        ADD     R0,@SP          ; ADD WORD INDEX TO BASE ADDR ON TOS
        ADD     R0,@SP          ; TO BUILD WORD ADDRESS FOR LDP
        GETNEXT                 ; GET ELEMENT WIDTH
        MOV     R0,-(SP)        ; NOW PUSH EL WIDTH FOR LDP STUFF
        CLR     -(SP)           ; NOW THE RIGHT-MOST BIT
1$:     ASR     R1              ; NOW A SHORT MULTIPLY FOR SMALL VALUES
        BCC     2$              ; SKIP IF THE MULTIPLICAND BIT IS OFF
        ADD     R0,@SP
```

```
2$:     ASL     R0              ; DOUBLE ADDEND
        TST     R1              ; ANY MULTIPLICATION AT ALL?
        BNE     1$              ; IF SO THEN KEEP LOOPING
        MORE

EQUI:   ; INTEGER EQUAL COMPARE
        SUB     (SP)+,@SP
        BEQ     PSHTRU
PSHFLS: CLR     @SP
        MORE
PSHTRU: MOV     #1,@SP
        MORE

GEQI:   ; INTEGER GREATER OR EQUAL COMPARE
        SUB     (SP)+,@SP
        BGE     PSHTRU
        BR      PSHFLS

GRTI:   ; INTEGER GREATER THAN COMPARE
        SUB     (SP)+,@SP
        BGT     PSHTRU
        BR      PSHFLS

LLA:    ; LOAD LOCAL ADDRESS
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     MP,R0
        MOV     R0,-(SP)
        MORE

LDCI:   ; LOAD LONG INTEGER CONSTANT
        MOVB    (IPC)+,-(SP)
        MOVB    (IPC)+,1(SP)
        MORE

LEQI:   ; INTEGER LESS THAN OR EQUAL COMPARE
        SUB     (SP)+,@SP
        BLE     PSHTRU
        BR      PSHFLS

LESI:   ; INTEGER LESS THAN COMPARE
        SUB     (SP)+,@SP
        BLT     PSHTRU
        BR      PSHFLS

LDL:    ; LOAD LOCAL
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     MP,R0
        MOV     @R0,-(SP)
        MORE

NEQI:   ; INTEGER NOT EQUAL COMPARE
        SUB     (SP)+,@SP
        BNE     PSHTRU
        BR      PSHFLS

STL:    ; STORE LOCAL
```

```
        GETBIG
        ASL     R0
        .IIF    NE,MSDLTA,      ADD     #MSDLTA,R0
        ADD     MP,R0
        MOV     (SP)+,@R0
        MORE

S1P:    ; STRING TO PACKED ON TOS
        INC     @SP
        MORE

IXB:    ; INDEX BYTE ARRAY
        ADD     (SP)+,@SP
        MORE

BYT:    ; CONVERT WORD TO BYTE ADDR
        MORE

; EQUAL FJP AND NOT EQUAL FJP ARE AT FJP

XIT:    ; EXIT SYSTEM
        HALT
        TRAP    SYSERR

NOP:    ; NO OPERATION
        MORE

ENTFP:  ; THIS SUBROUTINE STARTS THE THREADED CODE
        ; SEQUENCE A-LA FPMP $POLSH.  THE DIFFERENCE IS
        ; WE SAVE IPC REGISTER (R4)
        MOV     (SP)+,FPIPC     ; IPC MUST BE R4!!!
        JMP     @(R4)+          ; THREAD IT

FPIPC:  .WORD   ; SAVE R4 (IPC) REG HERE

XITFP:  ; HERE IS WHERE WE EXIT FROM FPMP BUSINESS
        MOV     LASTMP,MP
        MOV     #BACK,BK
        MOV     STKBAS,BASE
        MOV     FPIPC,IPC
        MORE

SETADJ: ; THIS IS A SUBROUTINE CALLED BY SET OPERATIONS
        ; TO MASSAGE SET SIZES AND REGISTERS...SEE THOSE OPS
        MOV     (SP)+,RETADR    ; SAVE RETURN ADDRESS
TRYAGN: MOV     (SP)+,R1        ; GRAB SET SIZE
        MOV     SP,R0           ; NOW POINT R0 AT NEXT SET
        ADD     R1,R0
        ADD     R1,R0
        CMP     (R0)+,R1        ; COMPARE FIRST SET SIZE WITH SECOND (TOP) SIZE
        BGE     SETSOK          ; QUIT IF SIZES ARE OK
        MOV     R1,-(SP)        ; ELSE EXPAND LOWER SET BY SHOVING IN 0-S
        MOV     -(R0),BK        ; GET SMALLER SET SIZE
        MOV     R1,@R0          ; CHANGE IT TO FINAL SIZE AFTER EXPAND
        MOV     R1,R0           ; CALCULATE NUMBER OF EXTRA ZEROES NEEDED
        SUB     BK,R0           ; R0 = TOPSIZE-LOWERSIZE
        MOV     R0,ZEROES       ; STASH IT FOR LATER USE
        ADD     BK,R1           ; NOW SET R1 TO TOTAL NUMBER OF WORDS TO COPY
        ADD     #2,R1           ; BE SURE TO INCLUDE SIZE WORDS
        MOV     SP,BK           ; POINT BK AT OLD TOS
```

```
        ASL     R0              ; DOUBLE SIZE DIF TO BYTES
        SUB     R0,SP           ; AND BUMP STACK TO MAKE ROOM
        MOV     SP,R0           ; NOW R0 IS DEST POINTER FOR COPY
1$:     MOV     (BK)+,(R0)+     ; COPY EACH WORD IN STACK
        SOB     R1,1$           ; LOOP FOR TOTAL SET SIZES
        MOV     ZEROES,R1       ; NOW COPY IN ZEROES BELOW SETS
2$:     CLR     (R0)+
        SOB     R1,2$
        MOV     #BACK,BK        ; RESTORE REG
        BR      TRYAGN          ; RESET REGISTERS AND EXIT
ZEROES: .WORD   ; TEMP FOR ABOVE EXPAND
SETSOK: TST     R1              ; LEAVE CC WITH R1 VALUE
        JMP     @(PC)+          ; BACK TO CALLER...LEAVE CC ALONE
RETADR: .WORD

        .PAGE
        .CSECT  TABLES
        .GLOBL  XFRTBL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                           ;
;                       OPERATOR TRANSFER TABLES                            ;
;                                                                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

XFRTBL  = . + 400       ; USE NEGATIVE INDEXES TO GET TO OPS
        .WORD   ABI
        .WORD   ABR
        .WORD   ADI
        .WORD   ADR
        .WORD   AND
        .WORD   DIF
        .WORD   DVI
        .WORD   DVR
        .WORD   CHK
        .WORD   FLO
        .WORD   FLT
        .WORD   INN
        .WORD   INT
        .WORD   IOR
        .WORD   MOD
        .WORD   MPI
        .WORD   MPR
        .WORD   NGI
        .WORD   NGR
        .WORD   NOT
        .WORD   SRS
        .WORD   SBI
        .WORD   SBR
        .WORD   SGS
        .WORD   SQI
        .WORD   SQR
        .WORD   STO
        .WORD   IXS
        .WORD   UNI
        .WORD   S2P
        .BLKW   1
        .WORD   LDCN
        .WORD   ADJ
        .WORD   FJP
        .WORD   INC
```

```
.WORD   IND
.WORD   IXA
.WORD   LAO
.WORD   LCA
.WORD   LDO
.WORD   MOV
.WORD   MVB
.WORD   SAS
.WORD   SRO
.WORD   XJP
.BLKW   2
.WORD   COMPAR
.WORD   COMPAR
.WORD   COMPAR
.WORD   LDA
.WORD   LDC
.WORD   COMPAR
.WORD   COMPAR
.WORD   LOD
.WORD   COMPAR
.WORD   STR
.WORD   UJP
.WORD   LDP
.WORD   STP
.WORD   LDM
.WORD   STM
.WORD   LDB
.WORD   STB
.WORD   IXP
.BLKW   2       ; CBP RNP
.WORD   EQUI
.WORD   GEQI
.WORD   GRTI
.WORD   LLA
.WORD   LDCI
.WORD   LEQI
.WORD   LESI
.WORD   LDL
.WORD   NEQI
.WORD   STL
.BLKW   3.
.WORD   S1P
.WORD   IXB
.WORD   BYT
.WORD   EFJ
.WORD   NFJ
.WORD   BPT
.WORD   XIT
.WORD   NOP
.LIST   ME
.IRP    N,<1,2,3,4,5,6,7,10,11,12,13,14,15,16,17,20>
.WORD   SLDLS+<6*<N-1>>
.ENDR
.IRP    N,<1,2,3,4,5,6,7,10,11,12,13,14,15,16,17,20>
.WORD   SLDOS+<6*<N-1>>
.ENDR
.IRP    N,<0,1,2,3,4,5,6,7>
.WORD   SINDS+<10*N>
.ENDR
.NLIST  ME
```

```
        .BLKW   3*<MAXUNT+1>    ; UNIT TABLE IN IOTRAP

CMPTBL: .WORD   0
        .WORD   REALCMP
        .WORD   STRGCMP
        .WORD   BOOLCMP
        .WORD   POWRCMP
        .WORD   BYTECMP
        .WORD   WORDCMP

XFRSET  = . + 242
        .WORD   EQUS
        .WORD   GEQS
        .WORD   0,0,0
        .WORD   LEQS
        .WORD   0,0
        .WORD   NEQS

SBROPS  = . + 242
        BEQ     .+4
        BGE     .+4
        BGT     .+4
        TRAP    SYSERR
        TRAP    SYSERR
        BLE     .+4
        BLT     .+4
        TRAP    SYSERR
        BNE     .+4

UBROPS  = . + 242
        BEQ     .+4
        BHIS    .+4
        BHI     .+4
        TRAP    SYSERR
        TRAP    SYSERR
        BLOS    .+4
        BLO     .+4
        TRAP    SYSERR
        BNE     .+4

        .RADIX  2.

BITTER: 0000000000000001
        0000000000000010
        0000000000000100
        0000000000001000
        0000000000010000
        0000000000100000
        0000000001000000
        0000000010000000
        0000000100000000
        0000001000000000
        0000010000000000
        0000100000000000
        0001000000000000
        0010000000000000
        0100000000000000
        1000000000000000
```

```
 CLRMSK: 1111111111111111
         1111111111111110
         1111111111111100
         1111111111111000
         1111111111110000
         1111111111100000
         1111111111000000
         1111111110000000
         1111111100000000
         1111111000000000
         1111110000000000
         1111100000000000
         1111000000000000
         1110000000000000
         1100000000000000
         1000000000000000
         0000000000000000

         .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp MainOp
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp ProcOp
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "procop.mac")


        .TITLE  PROCEDURE OPERATORS
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;
        .CSECT  PROCOP
        .GLOBL  CSPTBL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       PROCEDURE OPERATORS                            ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


MEMADR: .WORD   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
USGCNT: .WORD   -1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

OLDSEG: .WORD   ; SEG VALUE TO BE SAVED IN MSCW
OLDSP:  .WORD   ; SP VALUE ABOVE LOADED CODE IN READIT

READIT: ; END UP HERE IF SEGMENT IS NOT IN CORE...MAKE ROOM
        ; IN THE STACK AND READ IT.
        MOV     (SP)+,RTNTMP    ; SAVE RETURN ADDRESS
        MOV     R0,SEGNDX       ; PRESERVE R0
        ADD     R1,R0           ; MULTIPLY BY 6 TO INDEX INTO SEGTBL
        ASL     R0
        TST     SEGTBL+4(R0)    ; CHECK IF THERE IS CODE IN SEG
        BNE     GOTCODE         ; IF SO THEN WE CAN READ IT IN
CROAKM: TRAP    NOPROC          ; ELSE BOMB FOR SYSTEM ERROR
GOTCODE:SUB     SEGTBL+4(R0),SP ; OPEN UP GAP LARGE ENOUGH FOR CODE
        MOV     SP,R1           ; REMEMBER MEM ADDR FOR HANDLERS
        MOV     SEGTBL(R0),-(SP)        ; PUSH UNIT FOR HANDLERS
        MOV     R1,-(SP)                ; PUSH BUFFER
        MOV     SEGTBL+4(R0),-(SP)      ; PUSH LENGTH
        MOV     SEGTBL+2(R0),-(SP)      ; PUSH BLOCK
        CLR     -(SP)                   ; PUSH A ZERO,D  ALL ABOVE FOR HANDLERS
        MOV     R3,R3TEMP       ; AND R3 (ALL OTHERS SAVED BY CONVENTION)
        MOV     UUNIT(SP),R1    ; GET UNIT #
        ASL     R1              ; MULTIPLY BY 6
        ADD     UUNIT(SP),R1
        ASL     R1              ; FOR UNIT(*) INDEX
        ADD     #UNITBL,R1      ; R1 SHOULD BE ABS ADDR OF UNIT ENTRY
```

```
        JSR     R3,@2(R1)       ; ENTER HANDLER FOR PARTICULAR UNIT
        .WORD   1               ; 1 SINCE READ ONLY
3$:     TST     (R1)            ; NOW WAIT UNTIL UNIT IS NOT BUSY
        BMI     3$              ; BUSY WAIT UNTIL IO IS COMPLETE
        TSTB    @R1             ; CHECK IO RESULT FOR UNIT
        BEQ     2$
        TRAP    SYIOER          ; BOMB SYSTEM IO ERROR
2$:     MOV     R3TEMP,R3       ; RESTORE R3
        ADD     #12,SP          ; CHUCK PARAMETERS
        MOV     OLDSP,R1        ; RETRIEVE POINTER AT PARAM ON STACK
        TST     -(R1)           ; NOW POINT R1 AT TOP WORD IN PROCTBL
        MOV     SEGNDX,R2
        MOV     R1,MEMADR(R2)   ; SAVE THE ADDRESS OF THIS SEGMENT
        MOV     SP,RSEGNM       ; SAVE THE SEGMENT RELOCATION VALUE
        MOV     R1,-(SP)        ; SAVE R1
        MOV     R3,-(SP)        ; SAVE R3
RELOC:  MOVB    1(R1),NPROCS    ; SAVE THE NUMBER OF PROCEDURES IN THE SEGMENT
        BEQ     CROAKM          ; IF SEGMENT EMPTY THEN CROAK
        MOV     R1,R2
        MOV     R1,R3
        TST     -(R2)           ; LOOK AT SELF RELATIVE POINTER TO FIRST PROC.
        SUB     @R2,R2          ; LOOK AT JTAB OF FIRST PROC.
        CMP     #1,(R2)         ; IF NOT A BASE LEVEL OUTER BLOCK THEN
        BNE     1$              ; USE STKBAS AS BASE RELOCATION VALUE ELSE
        MOV     SP,R0           ; CALCULTE A NEW BASE RELOCATION VALUE
        ADD     #4,R0                 ; MAKE UP FOR REG SAVE CODE ABOVE
        SUB     PARMSZ(R2),R0         ; MAKE ROOM FOR PARAMETERS
        SUB     DATASZ(R2),R0         ; MAKE ROOM FOR DATA
        SUB     #MSDLTA+2,R0          ; ROOM FOR MARK STACK CONTROL BLOCK
        MOV     R0,RBASNM       ; THIS IS THE NEW BASE RELOCATION VALUE
        BR      RBEGIN
1$:     MOV     STKBAS,RBASNM
RBEGIN: TST     -(R3)           ; GET SELF RELATIVE POINTER TO PROCEDURE
        BEQ     CROAKM          ; FORWARD DECLARED PROCEDURE'S BODY MISSING
        MOV     R3,R1
        SUB     @R1,R1          ; SUBTRACT POINTER FROM ADDRESS OF POINTER
        TSTB    (R1)            ; IF PROC # <> 0 THEN
        BNE     RNEXT           ; P-CODE PROCEDURE SO NO RELOCATION ELSE
        TST     -(R1)           ; LOOK AT SELF RELATIVE POINTER TO ENTRY POINT
        MOV     R1,R2           ;   OF ASSEMBLY LANGUAGE PROCEDURE AND
        SUB     @R2,R2          ; CALCULATE ABSOLUTE ADDRESS OF ENTRY POINT
        MOV     R2,RLOCNM       ; THIS IS THE LOCAL RELOCATION VALUE
BASREL: MOV     -(R1),R0        ; GET THE NUMBER OF BASE RELOCATABLE ITEMS
        BEQ     SEGREL          ; IF NONE THEN SKIP TO SEGMENT RELOCATION
1$:     TST     -(R1)           ; GET SELF RELATIVE POINTER INTO ASSM CODE
        MOV     R1,R2           ; GET ADDRESS OF POINTER
        SUB     @R2,R2          ; SUBTRACT POINTER VALUE FROM ADDRESS
        ADD     RBASNM,(R2)     ; ADD BASE REL VALUE TO POINTED AT WORD
        SOB     R0,1$           ; REPEAT FOR EACH BASE RELOCATABLE ITEM

SEGREL: MOV     -(R1),R0        ; NUMBER OF SEGMENT RELOCATABLE ITEMS
        BEQ     LOCREL          ; IF NONE THEN SKIP TO LOCAL RELOCATION
1$:     TST     -(R1)
        MOV     R1,R2
        SUB     @R2,R2
        ADD     RSEGNM,(R2)     ; UPDATE EACH POINTED AT LOCATION
        SOB     R0,1$           ; REPEAT FOR ALL ITEMS

LOCREL: MOV     -(R1),R0        ; NUMBER OF BASE RELOCATABLE ITEMS
        BEQ     RNEXT           ; IF NONE THEN DONE WITH THIS PROCEDURE
```

```
1$:     TST     -(R1)
        MOV     R1,R2
        SUB     @R2,R2
        ADD     RLOCNM,(R2)     ; UPDATE THE POINTED AT LOCATION
        SOB     R0,1$           ; REPEAT FOR EACH LOCAL ITEM

RNEXT:  DEC     NPROCS          ; DECREMENT THE NUMBER OF PROCS TO BE CHECKED
        BNE     RBEGIN          ; IF ANY LEFT THEN DO IT AGAIN
        MOV     (SP)+,R3        ; RESTORE R3
        MOV     (SP)+,R1        ; RESTORE R1
TMPLBL: MOV     SEGNDX,R0       ; RESTORE R0
        MOV     RTNTMP,PC       ; RETURN TO CALLING PROCEDURE
R3TEMP: .WORD   0
NPROCS: .WORD   0
RBASNM: .WORD   0
RSEGNM: .WORD   0
RLOCNM: .WORD   0
SEGNDX: .WORD   0
RTNADR: .WORD   0
RTNTMP: .WORD   0

GETSEG: MOV     (SP)+,RTNADR    ; PUT RETURN ADDRESS IN R1
        MOV     (SP)+,R0        ; PUT SEG # IN R0
        MOV     SP,OLDSP
        MOV     R0,R1
        ASL     R0              ; SHIFT FOR WORD INDEX
        TST     USGCNT(R0)      ; CLEARS CARRY
        BGT     1$              ; SEGMENT ALREADY IN MEMORY
        BEQ     2$              ; SEGMENT IS NOT IN MEMORY SO READ IT
        MOV     @#MEMTOP,MEMADR(R0)  ; SPECIAL HANDLING FOR FIRST OP SYS CALL
        MOV     #1,USGCNT(R0)
        BR      1$
2$:     JSR     PC,READIT
        SEC                     ; CARRY SET INDICATES IO DONE, DO NOT INCLUDE
                                ; ANY INSTRUCTIONS WHICH WILL CHANGE THE CARRY
                                ; BETWEEN HERE AND THE BCC IN CXP.
1$:     INC     USGCNT(R0)
        MOV     RTNADR,PC

RELSEG: MOV     (SP)+,R1        ; PUT RETURN ADDRESS IN R1
        MOV     (SP)+,R0        ; PUT SEG # IN R0
        ASL     R0              ; DOUBLE FOR WORD INDEXING
        DEC     USGCNT(R0)      ; DECREMENT THE USAGE COUNT
        BPL     1$              ; BRANCH IF OK
        TRAP    SYSERR          ; SEGMENT HAS BEEN RELEASED TOO MANY TIMES
1$:     JMP     @R1

CXP:    ; CALL EXTERNAL (OTHER SEGMENT) PROCEDURE
        GETNEXT                 ; GRAB SEGMENT # OF CALLED PROC
        MOV     SEG,OLDSEG      ; SAVE SEG #
        CMPB    R0,@SEG         ; IS THE CALLED PROCEDURE IN SAME SEGMENT?
        BEQ     CIP             ; YES SO BRANCH TO CIP ELSE
        MOV     SP,OLDSP        ; SAVE THE STACK POINTER
        MOV     R0,-(SP)        ; PUSH NEW SEG #
        JSR     PC,GETSEG       ; GET SEGMENT
        MOV     MEMADR(R0),SEG
        BCC     2$              ; IF CARRY CLEAR THEN NO IO DONE
        CLR     BK              ; NOW OPEN EXTRA STACK SPACE FOR PARAMS...
        BISB    @IPC,BK         ; GET PROCEDURE NUMBER FROM CODE
        ASL     BK              ; DOUBLE FOR WORD INDEXING
```

```
        SUB     BK,R1           ; R1 NOW POINTS AT PROCTBL(P#)
        SUB     @R1,R1          ; R1 NOW POINTS AT JTAB FOR CALLED PROC
        SUB     PARMSZ(R1),SP   ; OPEN SOME SPACE FOR DUPLICATE PARAMS
2$:     MOV     OLDSP,R0
        MOV     #ENDCIP,BK      ; RETURN TO CIP (VERY GENERAL PROC CALLS)
        BR      XCLP            ; AND CALL LOCAL PROC

CALLAL: ; CALL USER ASSEMBLY LANGUAGE ROUTINE
        ADD     #ENTRIC,R1      ; POINT R1 AT ENTRIC IN SHORT JTAB
        SUB     @R1,R1          ; NOW R1 POINTS AT PDP-11 CODE
        JSR     PC,@R1          ; ENTER USER ROUTINE
        MOV     #BACK,BK        ; RESTORE THIS SCRATCH REG.
        MORE

CLPERR: TSTB    @SEG            ; CHECK IF CALLING EXECERROR...
        BNE     1$              ; IF NOT SEG 0 THEN CANT BE
        CMPB    @R1,#2          ; PROCEDURE # 2?
        BEQ     NOCARE          ; IF SO THEN DONT CARE ABOUT STCK OVER
1$:     ADD     DATASZ(R1),SP   ; RESTORE STACK W/O DAMAGE HOPEFULLY
        TRAP    STKOVR

CLP:    ; CALL LOCAL PROCEDURE
        MOV     SEG,OLDSEG      ; NO SEG CHANGE...SET UP TO SAVE CUR SEG
        MOV     SP,R0           ; NO CODE...LEAVE R0 AT PARAM LIST
XCLP:   ; ENTER HERE FOR EXTERNAL CALLS...R0 AND OLDSEG DIFFERENT
        GETBYTE R1              ; GET PROCEDURE #
        ASL     R1              ; CHANGE FOR WORD INDEXING
        NEG     R1              ; ENSURE NEGATIVE SINCE SEGP IS ABOVE TABLE
        ADD     SEG,R1          ; NOW R1 POINT AT SEGTABLE ENTRY FOR PROC
        SUB     @R1,R1          ; NOW R1 POINTS AT JTAB FOR PROC
        TSTB    @R1             ; IS PROC#=0?  (ASSEMBLY ROUTINE?)
        BEQ     CALLAL          ; IF SO CALL ASSEMBLY LANGUAGE CODE
        SUB     DATASZ(R1),SP   ; OPEN UP HOLE IN STACK FOR LOCAL VARS
        CMP     SP,NP           ; SEE IF WE ARE OVERFLOWING INTO HEAP
        BLOS    CLPERR          ; AAAAUUUUGGGGHHH STACK OVERFLOW!!!
NOCARE: TST     -(SP)           ; HOLE FOR FUTURE SP SAVE
        MOV     IPC,-(SP)       ; SAVE PROCESSOR STATE REGS
        MOV     OLDSEG,-(SP)    ; THUS BUILDING MSCW
        MOV     JTAB,-(SP)
        MOV     MP,-(SP)
        MOV     MP,-(SP)
        MOV     PARMSZ(R1),IPC  ; NOW COPY PARAMS (IF ANY)
        BEQ     2$              ; IF NONE, THEN SKIP MESSINESS
        ASR     IPC             ; WAS NUMBER OF BYTES...NOW WORDS
        MOV     SP,MP           ; SET UP MP TO PARAM COPY PLACE
        ADD     #MSDLTA+2,MP    ; MP NOW POINTS ABOVE MSCW...
1$:     MOV     (R0)+,(MP)+     ; LOOP AND COPY EACH PARAM WORD
        SOB     IPC,1$
2$:     MOV     SP,MP           ; NOW FINALLY POINT MP AT STAT LINK
        MOV     MP,LASTMP       ; SAVE THIS FOR EXECUTION ERROR
        MOV     R0,MSSP(MP)     ; STASH OLD SP VALUE
        MOV     R1,JTAB         ; NEW JUMP TABLE POINTER
        MOV     R1,IPC          ; SET UP CODE ENTRY POINT
        ADD     #ENTRIC,IPC     ; POINT IPC AT ENTRY OFFSET WORD
        SUB     @IPC,IPC        ; NOW IPC POINTS AT FIRST CODE BYTE
        MORE                    ; RETURN NOW

CGP:    ; CALL GLOBAL PROCEDURE
        MOV     #ENDCGP,BK      ; SET UP MAGIC RETURN
        BR      CLP             ; AND CALL LOCAL PROC
```

```
ENDCGP: MOV     BASE,@MP        ; CHANGE STAT LINK TO BASE
        MOV     #BACK,BK        ; RESTORE REGS
        MORE

CBP:    ; CALL BASE PROCEDURE
        MOV     #ENDCBP,BK
        BR      CLP
ENDCBP: MOV     BASE,-(SP)      ; ADD ON EXTRA MSCW WORD
        MOV     @BASE,@MP       ; POINT STAT LINK AT OUTER BLOCK
        MOV     MP,BASE         ; SET BASE REG TO THIS NEW PROC
        MOV     BASE,STKBAS     ; BE SURE TO UPDATE PERM BASE REG
        MOV     #BACK,BK        ; RESTORE
        MORE

CIP:    ; CALL INTERMEDIATE PROCEDURE
        MOV     #ENDCIP,BK
        BR      CLP
ENDCIP: MOVB    1(R1),BK        ; GRAB LEX LEVEL OF CALLED PROC
        BLE     ENDCBP          ; IF <= 0 THEN A BASE PROC CALL
        MOV     MP,R0           ; NOW SEARCH DOWN DYN LINKS FOR PARENT
1$:     MOV     MSJTAB(R0),R1   ; GRAB JTAB SAVED IN MSCW
        CMPB    1(R1),BK        ; COMPARE LEX LEVELS
        BLT     2$              ; IS IT LOWER? IF SO THEN FOUND PARENT
        MOV     MSDYN(R0),R0    ; ELSE LINK DOWN TO CALLER OF CURRENT
        BR      1$              ; AND LOOP UNTIL FOUND
2$:     MOV     @R0,@MP         ; SET UP FOUND STAT LINK
        MOV     #BACK,BK        ; RESTORE AND
        MORE

RBP:    ; RETURN FROM BASE LEVEL PROCEDURE
        MOV     MSBASE(MP),BASE ; GET BASE FROM MSCW
        MOV     BASE,STKBAS     ; AND SAVE IN PERM WORD
RNP:    ; RETURN FROM NORMAL PROCEDURE
        CMPB    @MSSEG(MP),@SEG ; ARE WE RETURNING TO THE SAME SEGMENT?
        BEQ     3$              ; YES SO BRANCH OTHERWISE
        CLR     -(SP)
        MOVB    @SEG,@SP        ; PUT SEGMENT NUMBER ON TOP OF STACK
        JSR     PC,RELSEG       ; RELEASE SEGMENT
3$:     MOV     MSSP(MP),R0     ; POP OLD SP VALUE
        GETNEXT R1              ; GRAB # OF WORDS TO RETURN
        BEQ     2$              ; IF NONE THEN SKIP RETURN CODE
        ADD     #MSDLTA+2,MP
        ADD     R1,MP           ; POINT MP ABOVE FUNCTION VALUE
        ADD     R1,MP           ; R1 IS WORDS
1$:     MOV     -(MP),-(R0)     ; PUSH RETURN WORDS ONTO STACK
        SOB     R1,1$           ; AND LOOP FOR TOTAL WORD COUNT
        MOV     LASTMP,MP       ; RESTORE OLD MP VALUE
2$:     MOV     MP,R1           ; NOW RESTORE STATE FROM MSCW
        TST     (R1)+           ; CHUCK STAT LINK
        MOV     (R1)+,MP        ; DYNAMIC LINK
        MOV     (R1)+,JTAB
        MOV     (R1)+,SEG
        MOV     (R1)+,IPC
        MOV     MP,LASTMP
        MOV     R0,SP           ; NOW BACK IN STATE AT CALL TIME
        MORE

CSP:    ; CALL STANDARD PROCEDURE
        GETNEXT                 ; GET STANDARD PROC #
        ASL     R0              ; SET FOR WORD INDEXING
```

```
        MOV     CSPTBL(R0),PC   ; TRANSFER TO PROPER SUBROUTINE

IOC:    ; IO CHECK
        TST     @#IORSLT
        BEQ     1$
        TRAP    UIOERR
1$:     MORE

NEW:    ; ALLOCATE DYNAMIC MEMORY
        CMP     @#GDIRP,#NIL    ; IS GLOB DIR NIL?
        BEQ     2$
        MOV     @#GDIRP,@#NP    ; RELEASE ITS SPACE
        MOV     #NIL,@#GDIRP    ; ZAP CURRENT DIRECTORY BUFFER
2$:     MOV     (SP)+,R1        ; GET NUMBER OF WORDS INTO R1
        MOV     @#NP,R0         ; GET CURRENT HEAP TOP IN R0
        MOV     R0,@(SP)+       ; SET POINTER PARAM TO NEW MEM SPACE
        ADD     R1,R0           ; POINT R0 ABOVE DYN MEM AREA
        ADD     R1,R0           ; BYTE WISE
        MOV     SP,R1           ; NOW CHECK FOR STK OVERFLOW
        SUB     #40.,R1         ; GIVE A 20 WORD BUFFER ZONE
        CMP     R0,R1           ; CHECK IF OVERLAPPING
        BLOS    1$              ; IF NEW HEAP TOP LOWER THEN OK
        TRAP    STKOVR          ; ELSE BOMB FOR STACK OVERFLOW
1$:     MOV     R0,@#NP         ; SAVE NEW HEAP TOP
        MORE

FLC:    ; FILL CHAR INTRIN...KB GROSSNESS
        MOVB    @SP,1(SP)       ; DUP LOW BYTE IN UPPER BYTE
        MOV     (SP)+,R1        ; CHAR TO FILL WITH
        MOV     @SP,BK          ; # CHARS TO FILL
        BLE     NOMOVE          ; LEAVE TWO THINGS ON STACK IN THIS CASE
        BIS     2(SP),@SP       ; OR ADDR AND BYTE COUNT
        ROR     (SP)+           ; CHUCK RESULT EXCEPT LOW BIT IN C
        MOV     (SP)+,R0        ; GRAB DEST ADDR, LEAVE C-BIT ALONE
        BCS     CHRFIL          ; IF ODD THEN MUST CHAR FILL ELSE
        CMP     R0,#160000      ; IS ADDR IN IO PAGE? (EG TERAK SCREEN)
        BHIS    CHRFIL
        ASR     BK
1$:     MOV     R1,(R0)+        ; MUCH FASTER!
        SOB     BK,1$
        BR      XITMOV
CHRFIL: MOVB    R1,(R0)+        ; FILL EACH CHAR W/ CHAR PARAM
        SOB     BK,CHRFIL
        BR      XITMOV

MVL:    ; MOVE LEFT MEMORY BLOCK
        MOV     (SP)+,BK        ; GRAB # BYTES TO MOVE
        BLE     NOMOVE          ; QUIT IF LENGTH <= 0
        MOV     (SP)+,R1        ; GET DESTINATION ADDR
        MOV     @SP,R0          ; GRAB SOURCE ADDR
        BIS     R1,@SP          ; CHECK FOR ODD COUNT IN ANY OPERAND
        BIS     BK,@SP          ; IN HOPES OF WORD MOVE
        ROR     (SP)+           ; OR-ED LOW BIT IN CARRY NOW
        BCS     1$              ; IF C SET THEN SOMETHING IS ODD
        CMP     R0,#160000      ; ADDR IN IO PAGE? (EG TERAK SCREEN)
        BHIS    1$
        CMP     R1,#160000
        BHIS    1$
        ASR     BK              ; ELSE WE CAN WORD MOVE!
2$:     MOV     (R0)+,(R1)+
```

```
        SOB     BK,2$
        BR      XITMOV
1$:     MOVB    (R0)+,(R1)+     ; COPY BYTES
        SOB     BK,1$
        BR      XITMOV

NOMOVE: ; GO HERE FOR A BAD MOVE REQUEST
        CMP     (SP)+,(SP)+     ; CHUCK ADDRESSES ON STACK
XITMOV: MOV     #BACK,BK
        MORE

MVR:    ; MOVE RIGHT BYTES
        MOV     (SP)+,BK        ; GRAB # BYTES TO MOVE RIGHT
        BLE     NOMOVE          ; QUIT IF <= 0
        MOV     (SP)+,R1        ; DESTATION ADDR
        MOV     (SP)+,R0        ; SOURCE ADDR
        ADD     BK,R0           ; POINT SOURCE AND DESTINATION
        ADD     BK,R1           ; AT END OF THE ARRAYS
1$:     MOVB    -(R0),-(R1)     ; BYTE COPY BACKWARDS
        SOB     BK,1$
        BR      XITMOV

XIT:    ; EXIT PROCEDURE
        MOV     JTAB,IPC        ; FIRST SET IPC TO EXIT FROM CURRENT
        ADD     #EXITIC,IPC     ; PROC ... GET INFO FROM CUR JTAB
        SUB     @IPC,IPC        ; NOW IPC IS SET TO EXIT MY CALLER
        CMPB    @JTAB,@SP       ; IS IT THE PROC # TO EXIT ANYWAY?
        BNE     XCHAIN          ; IF NOT THEN CHAIN DYN LINKS TO FIND
        CMPB    @SEG,2(SP)      ; IF PROC OK, HOW ABOUT SEG#?
        BNE     XCHAIN          ; IF WRONG, THEN CHAIN DYN TOO
        CMP     (SP)+,(SP)+     ; ELSE CHUCK STACK STUFF
        MORE                    ; AND DO THE RETURN CODE
XCHAIN: MOV     MP,R0           ; OK...START EXITING STACKED PROCS
XLOOP:  CMP     R0,@BASE        ; ARE WE ABOUT TO EXIT SYSTEM BLOCK?
        BEQ     XBOMB           ; IF SO THEN BIG BOOBOO
        MOV     MSJTAB(R0),R1   ; ELSE OK...GRAB JTAB AND FUDGE MS IPC
        ADD     #EXITIC,R1      ; TO EXIT CODE RATHER THAN NORMAL REENTRY
        SUB     @R1,R1          ; R1 NOW HAS EXIT POINT IPC
        MOV     R1,MSIPC(R0)    ; SO PLACE IN STACK FRAME
        CMPB    @MSJTAB(R0),@SP ; IS THIS THE PROC# TO EXIT FROM?
        BNE     1$              ; IF NOT THEN GO TO NEXT CALLED PROC
        CMPB    @MSSEG(R0),2(SP)        ; AND RIGHT SEG#
        BNE     1$
        CMP     (SP)+,(SP)+     ; WELL, FOUND IT...CHUCK PARAMS
        MORE                    ; AND FALL OUT OF PROC
1$:     MOV     MSDYN(R0),R0    ; CHAIN DOWN DYNAMIC LINKS!
        BR      XLOOP
XBOMB:  TRAP    NOEXIT

        ;TREESEARCH (TREEROOTP, VAR FOUNDP, VAR TARGETNAME)
        ;-SEARCHS A BINARY TREE, EACH OF WHOSE NODES CONTAIN
        ; AT LEAST THE FOLLOWING COMPONENTS, IN ORDER SHOWN:
        ;       A)  CODEWD: ALPHA (8 CHAR NODE NAME)
        ;       B)  RLINK: CTP  (POINTER TO RIGHT SUBTREE)
        ;       C)  LLINK: CTP  (POINTER TO LEFT SUBTREE)

        ;-RETURNS POINTER TO TARGET NODE THROUGH CALL BY NAME PARA-
        ; METER AND DESCRIPTION OF SEARCH RESULTS AS INTEGER FUNCTION
        ; VALUE WITH 3 POSSIBLE VALUES:
        ;       A)  0:  TARGET NAME WAS FOUND; FOUNDP POINTS TO IT
```

```
        ;       B)  1:  NO MATCH; TARGET > LEAF NODE; FOUNDP => LEAF
        ;       C) -1:  NO MATCH; TARGET < LEAF NODE; FOUNDP => LEAF
        ;-ROOT POINTER ASSUMED TO BE NON NIL.

TRS:    MOV     (SP)+,R0        ; GET ADDR OF TARGET NAME
        MOV     2(SP),R1        ;GET ROOT OF TREE
TRLOOP: CMP     @R0,@R1         ;FIRST WORD COMPARE
        BNE     TRNEXT
        CMP     2(R0),2(R1)
        BNE     TRNEXT
        CMP     4(R0),4(R1)
        BNE     TRNEXT
        CMP     6(R0),6(R1)
        BNE     TRNEXT
        MOV     R1,@(SP)+       ;FOUND IT!  TELL USER WHERE
        CLR     @SP             ;RETURN ZERO VALUE
        MORE

TRNEXT: BHI     TRRIGHT         ;WHICH SUBTREE NEXT?
        CMP     #NIL,12(R1)     ;LEFT- IS IT NIL?
        BNE     NEXTL           ;NOPE, CARRY ON
        MOV     R1,@(SP)+       ;YES- RETURN POINTER
        MOV     #177777,(SP)    ;AND FUNCTION VALUE
        MORE
NEXTL:  MOV     12(R1),R1       ;ON TO POSTERITY
        BR      TRLOOP

TRRIGHT:CMP     #NIL,10(R1)     ;RIGHT TREE NIL?
        BNE     NEXTR
        MOV     R1,@(SP)+       ;POINTER
        MOV     #1,(SP)         ;AND FUNCTION VALUE
        MORE

NEXTR:  MOV     10(R1),R1       ;POSTERITY AGAIN...
        BR      TRLOOP

        ;IDSEARCH(SYMCURSUR[START OF SYM INFO BUFF],SYMBUF[SOURCE BUF])
        ;ORDER OF SYMBOL INFO BLOCK IS
        ;       A) SYMCURSUR    (POINTER IN SYMBOLIC BUFFER)
        ;       B) SY           (SYMBOL)
        ;       C) OP           (OPERATOR)
        ;       D) IDCODE       (8 CHAR ID NAME)
        ;IDSEARCH EXITS WITH SYMCURUSR UPDATED TO POINT TO THE END OF
        ;NEXT ID. SY AND OP DESCRIBE THE TOKEN FOUND, AND IDCODE CON-
        ;TAINS THE FIRST 8 CHARACTERS (BLANK FILLED)  CONVERTED TO UPPERCASE.
        ;ON ENTRY, SYMCURUSR POINTS TO FIRST CHARACTER OF  ID, WHICH
        ;IS ASSUMED TO BE ALPHABETIC.  ALSO ON ENTRY, TOS-1 IS ADDRESS OF
        ;SYMCURSUR AND TOS IS ADDR OF SYMBUF

IDS:    MOV     (SP)+,R0
        MOV     (SP),R1
        MOV     R3,-(SP)        ; SAVE OLD R3
        MOV     R4,-(SP)        ; SAVE OLD R4
        MOV     (R1),R4         ; GET VALUE OF SYMCURSOR
        ADD     R4,R0           ; GET ADDRESS OF SYMBOL
        ADD     #6,R1           ; GET ADDRESS OF IDCODE
        MOV     R1,-(SP)        ; SAVE ADDRESS OF IDCODE
        MOV     #400,R3         ; SET SHIFT REGISTER FOR 8 CHARS

CHLOOP: MOVB    (R0)+,R2        ; GET SOURCE CHARACTER
```

```
        INC     R4              ; BUMP SYMCURSOR
        CMPB    #137,R2         ; IS IT AN UNDERSCORE ? IGNORE IF SO
        BEQ     CHLOOP
        CMPB    R2,#'0          ; IS IT LESS THAN A '0' ?
        BLO     GOTRW
        CMPB    R2,#'9          ; IS IT LESS THAN A '9' ?
        BLOS    GOTCH           ; IF SO, IT'S OK
        BIC     #40,R2          ; MAKE SURE IT'S UPPERCASE
        CMPB    R2,#'A          ; IS IT LESS THAN AN 'A' ?
        BLO     GOTRW
        CMPB    R2,#'Z          ; IS IT GREATER THAN A 'Z' ?
        BHI     GOTRW
GOTCH:  ASR     R3              ; HAVE WE RUN OUT THE 8 CHARACTERS ?
        BEQ     CHLOOP          ; IF SO, DON'T MOVE SYMBOL INTO IDCODE
        MOVB    R2,(R1)+        ; MAKE CHARACTER PART OF ID BUFFER
        BR      CHLOOP

GOTRW:  SUB     #2,R4           ; POINT SYMCURSOR AT LAST IDENTIFIER CHAR
        MOV     #40,R2          ;   OF IDCODE BUFFER
1$:     ASR     R3              ; DECREMENT COUNT
        BEQ     2$              ; RUN OUT OF PLACES ??
        MOVB    R2,(R1)+        ; NOT YET, BLANK IT
        BR      1$
2$:     MOVB    @(SP),R2        ; GET INDEX OF
        ASL     R2              ;   RESWORD TO START
        MOV     RESTBL-'A-'A(R2),R1     ; GET TO INDEX OF LETTER
        MOV     RESTBL-'A-'A+2(R2),R3   ; GET INDEX OF NEXT LETTER
        SUB     R1,R3           ; GET NUMBER OF SYMBOLS TO CHECK
        ASL     R3              ; MAKE INTO WORD OFFSET
        MOV     BITTER(R3),R3   ; TURN COUNT INTO SHIFT REGISTER
        ASL     R1              ; MULTIPLY BY 12
        ASL     R1
        MOV     R1,-(SP)
        ASL     R1
        ADD     (SP)+,R1
        ADD     #RESTBL+54.,R1  ; GET ABSOLUTE ADDRESS OF START

RWLOOP: ASR     R3              ; DECREMENT RECORD COUNT
        BEQ     RWBAD           ; HAVE WE RUN OUT OF CHOICES ??
        MOV     @SP,R0          ; GET ADDRESS OF IDCODE
        CMP     (R0)+,(R1)+     ; IS FIRST WORD EQUAL ?
        BNE     1$
        CMP     (R0)+,(R1)+     ; IS SECOND WORD EQUAL ?
        BNE     2$
        CMP     (R0)+,(R1)+     ; IS THIRD WORD EQUAL ?
        BNE     3$
        CMP     (R0)+,(R1)+     ; IS FOURTH (AND LAST) WORD EQUAL ?
        BNE     4$
        MOV     (R1)+,R0        ; FOUND A MATCH, R0:=SY
        MOV     (R1)+,R1        ; R1:=OP
        BR      RWDONE          ; FINISH UP
1$:     ADD     #2,R1           ; OFFSET
2$:     ADD     #2,R1           ;   TO NEXT
3$:     ADD     #2,R1           ;     ID RECORD
4$:     ADD     #4,R1           ; GO TO NEXT RECORD
        BR      RWLOOP          ;  AND TRY TRY AGAIN

RWBAD:  CLR     R0              ; SY:=0
        MOV     #15.,R1         ; OP:=15  (NOOP)
RWDONE: MOV     R4,@6(SP)       ; SYMCURSOR:=^LAST CHAR OF SYMBOL
```

```
        MOV     (SP)+,R4        ; WASTE POINTER TO IDCODE
        MOV     (SP)+,R4        ; GET OLD R4 BACK
        MOV     (SP)+,R3        ; GET OLD R3 BACK
        MOV     (SP)+,R2        ; GET ADDRESS OF SYMCURSOR
        ADD     #2,R2           ; GET TO ADDRESS OF SY
        MOV     R0,(R2)+        ; SY:=R0
        MOV     R1,(R2)         ; OP:=R1
        MOV     #BACK,BK        ; GO FOR IT
        MORE                    ;  ... AND PRAY

        .EVEN

RESTBL: .WORD   0,2,3,5,8.,11.,15.,16.,16.,20.,20.,20.
        .WORD   21.,22.,23.,25.,28.,28.,30.,33.,36.
        .WORD   39.,40.,42.,42.,42.,42.

.MACRO  RW      NAME,SY,OP
        .ASCII  /NAME/
        .WORD   SY,OP
.ENDM   RW

        RW      <AND     >,39.,2
        RW      <ARRAY   >,44.,15.
        RW      <BEGIN   >,19.,15.
        RW      <CASE    >,21.,15.
        RW      <CONST   >,28.,15.
        RW      <DIV     >,39.,3
        RW      <DO      >,6  ,15.
        RW      <DOWNTO  >,8. ,15.
        RW      <ELSE    >,13.,15.
        RW      <END     >,9. ,15.
        RW      <EXTERNAL>,53.,15.
        RW      <FOR     >,24.,15.
        RW      <FILE    >,46.,15.
        RW      <FORWARD >,34.,15.
        RW      <FUNCTION>,32.,15.
        RW      <GOTO    >,26.,15.
        RW      <IF      >,20.,15.
        RW      <IMPLEMEN>,52.,15.
        RW      <IN      >,41.,14.
        RW      <INTERFAC>,51.,15.
        RW      <LABEL   >,27.,15.
        RW      <MOD     >,39.,4
        RW      <NOT     >,38.,15.
        RW      <OF      >,11.,15.
        RW      <OR      >,40.,7
        RW      <PACKED  >,43.,15.
        RW      <PROCEDUR>,31.,15.
        RW      <PROGRAM >,33.,15.
        RW      <RECORD  >,45.,15.
        RW      <REPEAT  >,22.,15.
        RW      <SET     >,42.,15.
        RW      <SEGMENT >,33.,15.
        RW      <SEPARATE>,54.,15.
        RW      <THEN    >,12.,15.
        RW      <TO      >,7  ,15.
        RW      <TYPE    >,29.,15.
        RW      <UNIT    >,50.,15.
        RW      <UNTIL   >,10.,15.
        RW      <USES    >,49.,15.
```

```
        RW      <VAR     >,30.,15.
        RW      <WHILE   >,23.,15.
        RW      <WITH    >,25.,15.
        .WORD   0
        .EVEN


TIM:    ; RETURN TIME OF DAY WORDS
        MOV     LOTIME,@(SP)+
        MOV     HITIME,@(SP)+
        MORE

SCN:    ; SCAN ARRAY
        TST     (SP)+           ; EXTRA MASK PARAM...NOT USED YET
        MOV     @SP,R0          ; GRAB ADDR TO START SCAN
        MOV     2(SP),BK        ; CHAR TO SCAN FOR
        MOV     6(SP),R1        ; LENGTH TO SCAN FOR
        BEQ     NOTFND          ; IF NULL SCAN THEN RETURN 0
        BMI     BCKSCN          ; IF NEGATIVE THEN BACKWARD SCAN
        TST     4(SP)           ; ELSE FORWARD SCAN...CHECK RELOP
        BNE     2$              ; NEQ 0 MEANS NEQ SCAN
1$:     CMPB    (R0)+,BK        ; ELSE EQUAL COMPARE BYTES
        BEQ     3$              ; UNTIL ONE IS EQUAL
        SOB     R1,1$
        BR      NOTFND
2$:     CMPB    (R0)+,BK        ; DO NEQ COMPARE
        BNE     3$
        SOB     R1,2$
        BR      NOTFND
3$:     DEC     R0              ; POINT R0 AT CHAR FOR FIX.R0
FIX.R0: SUB     (SP)+,R0        ; MAKE R0 THE DISPLACEMENT FROM SCAN START
        CMP     (SP)+,(SP)+     ; CHUCK CHAR & RELOP PARAMS
        MOV     R0,@SP          ; RETURN DISP ON TOS
        MOV     #BACK,BK
        MORE
BCKSCN: NEG     R1              ; MAKE A NUMBER SUITABLE FOR SOB OP
        INC     R0              ; PRE-DEC SETTUP
        TST     4(SP)           ; CHECK OP TYPE
        BNE     2$
1$:     CMPB    -(R0),BK        ; SCAN BACKWARD EQUAL COMPARE
        BEQ     FIX.R0          ; WHEN FOUND THEN RETURN DISP
        SOB     R1,1$
        BR      NOTFND
2$:     CMPB    -(R0),BK
        BNE     FIX.R0
        SOB     R1,2$
NOTFND: MOV     6(SP),R0        ; RETURN SCAN LENGTH IN THIS CASE
        ADD     @SP,R0          ; THAT SIGNIFIES UNSUCCESSFUL SCAN
        BR      FIX.R0

TRC:    ; REAL TRUNCATE
        JSR     R4,ENTFP
        .WORD   $RI,XITFP

RND:    ; REAL ROUND
        MOV     @SP,R0          ; GET SIGN WORD OF PARAM TO ADD + OR - .5
        CLR     -(SP)           ; LOW ORDER REAL 0.5
        MOV     #100000,-(SP)   ; HIGH ORDER SHIFTED ONE LEFT
        ROL     R0              ; SHIFT SIGN OF PARAM INT.O C-BIT
        ROR     @SP             ; AND PLACE IN SIGN OF THE 0.5
```

```
        .IF     DF,FPI
        FADD    SP
        .ENDC
        JSR     R4,ENTFP
        .IF     NDF,FPI
        .WORD   $ADR
        .ENDC
        .WORD   $RI,XITFP

SINCSP: ; REAL SINE
        JSR     R4,ENTFP
        .WORD   CALJR5,SIN

COSCSP: ; REAL COSINE
        JSR     R4,ENTFP
        .WORD   CALJR5,COS

LOGCSP: ; BASE-10 LOGARITHM
        JSR     R4,ENTFP
        .WORD   CALJR5,ALOG10

ATNCSP: ; REAL ARCTANGENT
        JSR     R4,ENTFP
        .WORD   CALJR5,ATAN

LNCSP:  ; NATURAL LOGARITHM
        JSR     R4,ENTFP
        .WORD   CALJR5,ALOG

EXPCSP: ; EXPONENTIAL FUNCTION
        JSR     R4,ENTFP
        .WORD   CALJR5,EXP

SQTCSP: ; REAL SQUARE ROOT
        JSR     R4,ENTFP
        .WORD   CALJR5,SQRT

CALJR5: ; THIS SUBROUTINE MAGICALLY CALLS FPMP STUFF
        MOV     SP,1$           ; PUT REAL PARAM ADDR INTO CODE
        JSR     R5,@(R4)+       ; ENTER THE ROUTINE DESIRED
        BR      2$              ; PLEASE SEE CALL SEQUENCE IN FPMP DOC
1$:     .WORD   ; ADDR OF PARAM GOES HERE
2$:     MOV     R1,2(SP)        ; PUT LOW ORDER RESULT IN STACK
        MOV     R0,@SP          ; AND THEN HIGH ORDER
        JMP     XITFP           ; FINALLY EXIT

GSEG:   JSR     PC,GETSEG
        MOV     #BACK,BK
        MORE

RSEG:   JSR     PC,RELSEG
        MORE

MRK:    ; MARK HEAP
        CMP     @#GDIRP,#NIL    ; IS THE GLOB DIR NIL?
        BEQ     1$
        MOV     @#GDIRP,@#NP
        MOV     #NIL,@#GDIRP
1$:     MOV     @#NP,@(SP)+     ; SAVE TOP OF HEAP IN POINTER PARAM
        MORE
```

```
RLS:    ; RELEASE HEAP
        MOV     @(SP)+,@#NP     ; CUT BACK HEAP POINTER
        MOV     #NIL,@#GDIRP    ; ZAP GLOBAL DIR THING
        MORE

IOR:    ; RETURN IO RESULT
        MOV     @#IORSLT,-(SP)
        MORE

;. BUILD A POWER OF TEN TABLE
EXPON = 0
.MACRO  PWR10   EXP
        .FLT2   1.0E'EXP
.ENDM
TENTBL: .REPT   38.
        PWR10   EXPON
        EXPON = EXPON+1
        .ENDR

POT:    ; POWER OF TEN
        MOV     (SP)+,R0        ; GET POWER DESIRED
        BMI     BADPOT          ; NO NEGATIVE POWER ALLOWED
        CMP     R0,#EXPON       ; SEE IF INDEX IS TOO BIG
        BGE     BADPOT          ; CROAK FOR THAT TOO
        ASL     R0              ; ELSE MAKE A REAL ARRAY INDEX
        ASL     R0              ; MULTIPLY BY 4
        MOV     TENTBL+2(R0),-(SP)      ; LOW ORDER WORD
        MOV     TENTBL(R0),-(SP)        ; AND HIGH ORDER WORD
        MORE
BADPOT: TRAP    INVNDX

HLT:    ; HALT AND/OR BREAKPOINT...EXECERROR KNOWS
        MOV     (PC)+,@BK       ; STASH TRAP HLTBPT INTO OP FETCH
        TRAP    HLTBPT
        MORE

MEM:    ; RETURN # WORD OF FREE MEM
        MOV     SP,R0           ; TOP OF FREE MEM
        SUB     NP,R0           ; R0 NOW # BYTES
        CLC                     ; SET C-BIT TO 0
        ROR     R0              ; MAKE # WORDS, CLEAR SIGN
        MOV     R0,-(SP)
        MORE

UBUSY:  JSR     R4,ENTFP
        .WORD   BSYSTRT,IOSTRT,BSYTST,CHKERR,IODONE

UWAIT:  JSR     R4,ENTFP
        .WORD   WATSTRT,IOSTRT,BSYWAIT,CHKERR,IODONE

UCLEAR: JSR     R4,ENTFP
        .WORD   WATSTRT,IOSTRT,CLRUNT,IODONE

UREAD:  JSR     R4,ENTFP
        .WORD   IOSTRT,INMODE,BSYWAIT,CHKERR,STRTIN
        .WORD   CHKWAIT,BSYWAIT,CHKERR,IODONE

UWRITE: JSR     R4,ENTFP
        .WORD   IOSTRT,OUTMODE,BSYWAIT,CHKERR,STRTOUT
```

```
        .WORD   CHKWAIT,BSYWAIT,CHKERR,IODONE

        ; BELOW ARE THE THREAD MODULES FOR THE ABOVE
        ; OPERATIONS..  IT IS SUGGESTED THAT YOU LOOK
        ; HERE BEFORE TRYING TO FIGURE OUT THE INTERRUPT
        ; HANDLER INTERFACE TO THIS SECTION.

BSYSTRT:MOV     (SP),-(SP)      ; DUPL UNIT# PARAM
        CLR     2(SP)           ; SHOVE A FALSE INTO STACK FOR RETURN
WATSTRT:SUB     #8.,SP          ; MAKE STACK LOOK OK FOR IODONE
        JMP     @(R4)+          ; AND ONWARD WE GO

BSYTST: TST     (R1)            ; SEE IF UNIT IS IN FACT BUSY
        BPL     THRUR4          ; IF NOT, CONTINUE SEQUENCE
        INC     <UUNIT+2>(SP)   ; SET RETURN VALUE TO 1 (TRUE)
        BR      IODONE          ; AND QUIT NOW

CLRUNT: JSR     PC,@4(R1)
        CLRB    @R1
        JMP     @(R4)+

IOSTRT: CLR     R5              ; ERROR REGISTER, NO ERROR YET
        MOV     UUNIT(SP),R1    ; GRAB RAW UNIT #
        BLE     1$              ; IF <= ZERO, GIVE BADUNIT ERROR
        CMP     R1,#MAXUNT      ; SEE IF NUMBER IS TOO BIG
        BGT     1$              ; UNITBL INDEXED 1..MAXUNT
        ASL     R1              ; ITS OK, MULTIPLY BY 6
        ADD     UUNIT(SP),R1
        ASL     R1              ; TO GET AN ACTUAL ADDR IN
        ADD     #UNITBL,R1      ; UNITBL, R1 NOW IS ABS ADDR OF UNIT
        BIT     #INBIT!OUTBIT,@R1
        BEQ     1$              ; IF NOT IO ALLOWED AT ALL THEN ERROR
        JMP     @(R4)+          ; SO CONTINUE WITH SEQUENCE
1$:     MOV     #UNTERR,R5      ; ERROR RESULT FOR JUNK UNIT #
                                ; AND FALL INTO IO DONE
IODONE: MOV     R5,@#IORSLT     ; GIVE ANY ERROR RESULTS TO SYSTEM
        ADD     #10.,SP         ; GET RID OF PARAMS ON STACK
        JMP     XITFP           ; AND RETURN TO PROGRAM

INMODE: BIT     #INBIT,(R1)     ; SEE IF INPUT ALLOWED ON THE UNIT
MODTST: BNE     THRUR4          ; IF ONE BIT, THEN GO AHEAD
        MOV     #MODERR,R5      ; ELSE GIVE BAD MODE ERROR
        BR      IODONE

OUTMODE:BIT     #OUTBIT,(R1)    ; SEE IF OUTPUT ALLOWED ON UNIT
        BR      MODTST          ; AND SKIP TO ACTUAL TEST CODE

BSYHANG:MTPS    #0              ; ENSURE LOW PRIORITY BEFORE WAIT
        WAIT                    ; WAIT UNTIL AN INTERRUPT OCCURS
BSYWAIT:MTPS    #340            ; NO INTERRUPTS IN HERE...TIMING PROBS
        TST     (R1)            ; HIGH ORDER BIT TELLS IF BUSY
        BMI     BSYHANG         ; SO WAIT AROUND UNTIL THE BIT IS OFF
        MTPS    #0              ; OK...ALLOW INTERRUPTS
THRUR4: JMP     @(R4)+          ; CONVENIENT LOCATION FOR CONDITIONAL JMP

CHKERR: TSTB    (R1)            ; LOW BYTE IS HARD IO RSLT
        BEQ     THRUR4          ; IF NO ERROR, THEN KEEP GOING
        MOVB    (R1),R5         ; ELSE GIVE TO IORSLT AND QUIT NOT
        CLRB    (R1)            ; BE SURE TO CLEAR UNIT OR SYSTEM BOMB
        BR      IODONE
```

```
CHKWAIT:BIT     #1,UNOWAIT(SP)  ; SEE IF USER WANTS TO WAIT FOR IO
        BNE     IODONE          ; IF PARAM IS TRUE, THEN GO BACK TO CALLER
        JMP     @(R4)+          ; ELSE D.O BUSYWAIT ETC

STRTIN: JSR     R3,@2(R1)       ; JUMP INTO INTERRUPT HANDLER TO START IO
        .WORD   1               ; ONE HERE SAYS READ OP
        JMP     @(R4)+

STRTOUT:JSR     R3,@2(R1)       ; JUMP TO INTERRUPT HANDLER
        .WORD   0               ; ZERO MEANS WRITE OP
        JMP     @(R4)+          ; AND CONTINUE

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; HERE WE STICK A FEW MISCELLANEOS THINGS

DIV:    .IF     DF,EIS
        MOV     R1,DENOM        ; STASH DENOM INTO OP FIELD
        ASHC    #-16.,R0        ; SHIFT R0 INTO R1 WITH SIGN EXT
        DIV     (PC)+,R0        ; PERFORM DIVID OP
DENOM:  .WORD   0               ; DENOMINTOR GOES HERE
        BCC     1$              ; C-BIT IS ON FOR DIV BY ZERO
        TRAP    DIVZER
1$:     RTS     PC
        .IFF
        CLR     -(SP)           ;CLEAR SIGN FLAG
        TST     R1              ;EXAMINE DENOMINATOR
        BGT     1$              ;PLUS
        BNE     3$              ;GIVE EXECERR IF DIV 0
        TRAP    DIVZER
3$:     INC     (SP)            ;REMEMBER IF NEGATIVE
        NEG     R1              ;AND MAKE IT POS
1$:     TST     R0              ;TEST NUMERATOR
        BGT     2$              ;PLUS?
        BNE     4$              ;NOT ZERO, THEN HAVE TO DO WORK
        CLR     BK              ;MAKE REMAINDER ZERO
        TST     (SP)+           ;THROW AWAY SIGN INFORMATION
        BR      DONED           ;AND THEN JUMP TO END
4$:     INC     (SP)            ;ELSE NEGATIVE
        NEG     R0
2$:     MOV     #8.,-(SP)       ;8 ITERATIONS
        CLR     BK              ;HIGH ORDER DIVIDEND
        SWAB    R0              ;ANY HIGH ORDER NUMERATOR?
        BEQ     DIVD            ;NO, THEN PROCEED TO DIVIDE
        ASL     @SP             ;ELSE NEED 16 ITERATIONS
        SWAB    R0              ;AND RESTORE NUMERATOR
DIVD:   ASL     R0              ;DOUBLE DIVIDEND
        ROL     BK
        BEQ     LOP             ;JUMP IF NO CHANCE THIS TIME
        INC     R0              ;QUOTIENT BIT
        SUB     R1,BK           ;TRIAL STEP
        BHIS    LOP             ;OK
        ADD     R1,BK           ;DIVIDEND NOT BIG ENOUGH
        DEC     R0              ;RETRACT QUOTIENT BIT
LOP:    DEC     @SP             ;COUNT THIS LOOP
        BGT     DIVD            ;CONTINUE TIL DONE
        NEG     R0              ;NEGMAX CHECK
        TST     (SP)+
```

```
        ASR     (SP)+           ;GET SIGN OF QUOTIENT
        BCS     DONED           ;JUMP IF NEG
        NEG     R0              ;ANSWER POSITIVE
        BVS     OVR             ;GIVE OVERFLOW ERROR
DONED:  MOV     BK,R1           ;REMAINDER IN R1
        MOV     #BACK,BK
        RTS     PC
        .ENDC

        .IF     DF,EIS
MLI:    MUL     R0,R1
        MOV     R1,R0           ; EXPECTS RESULTS IN R0
        RTS     PC
        .IFF
OVR:    TRAP    INTOVR

MLI:    CLR     -(SP)                   ;SIGN STORAGE
        TST     R1                      ;CHECK MULTIPLICAND
        BGT     1$                      ;SKIP FOLLOWING IF +
        BEQ     ZEROM                   ;ANSWER IS ZERO
        INC     @SP                     ;REMEMBER -
        NEG     R1
1$:     TST     R0                      ;TEST MULTIPLIER
        BGT     2$
        BEQ     ZEROM
        INC     @SP
        NEG     R0
2$:     MOV     #8.,-(SP)       ; SET UP ITERATION COUNT
        CMP     R1,R0                   ;MAKE SURE
        BGE     CLR                     ;MULTIPLIER
        MOV     R1,BK                   ;IS
        MOV     R0,R1                   ;SMALLER
        MOV     BK,R0
CLR:    CLR     BK                      ;CLEAR HIGH ORDER PRODUCT
MUL:    ROR     BK                      ;SHIFT PRODUCT
        ROR     R0
        BCC     CYC                     ;MULTIPLIER BIT = 0?
        ADD     R1,BK                   ;NO,ADD IN MULTIPLICAND
CYC:    DEC     @SP                     ;COUNT LOOP
        BGT     MUL
        TST     (SP)+
        TSTB    R0                      ;TEST HIGH MULTI
;       BNE     OVR                     ;ERROR .IF MULTIPLIER NOT GONE
        BISB    BK,R0                   ;MOVE PRODECT RIGHT
        SWAB    R0
        CLRB    BK
        SWAB    BK
        ASR     BK                      ;ONE MROE SHIFT
;       BNE     OVR                     ;PRODUCT EXCEEDED 15 BITS
        ROR     R0
        NEG     R0                      ;MAKE NEG
;       BPL     OVR                     ;TOO BIG
        ROR     (SP)+                   ;DETERMINE SIGN OF PRODUCT
        BCS     OUTM
        NEG     R0                      ;SHOULD BE +
;       BVS     OVR
OUTM:   MOV     #BACK,BK
        RTS     PC

ZEROM:  CLR     R0
```

```
        TST     (SP)+
        BR      OUTM                    ;AND CLEAN UP
        .ENDC

CSPTBL: .WORD   IOC
        .WORD   NEW
        .WORD   MVL
        .WORD   MVR
        .WORD   XIT
        .WORD   UREAD
        .WORD   UWRITE
        .WORD   IDS
        .WORD   TRS
        .WORD   TIM
        .WORD   FLC
        .WORD   SCN
        .IF     DF,TERAK
        .WORD   DRAWLINE
        .WORD   DRAWBLOCK
        .IFF
        .WORD   0,0
        .ENDC
        .WORD   0,0,0,0,0,0,0
        .WORD   GSEG
        .WORD   RSEG
        .WORD   TRC
        .WORD   RND
        .WORD   SINCSP
        .WORD   COSCSP
        .WORD   LOGCSP
        .WORD   ATNCSP
        .WORD   LNCSP
        .WORD   EXPCSP
        .WORD   SQTCSP
        .WORD   MRK
        .WORD   RLS
        .WORD   IOR
        .WORD   UBUSY
        .WORD   POT
        .WORD   UWAIT
        .WORD   UCLEAR
        .WORD   HLT
        .WORD   MEM
        .CSECT  TABLES
        .BLKW   30.
        .WORD   CSP
        .BLKW   14.
        .WORD   RNP
        .WORD   CIP
        .BLKW   18.
        .WORD   RBP
        .WORD   CBP
        .BLKW   10.
        .WORD   CXP
        .WORD   CLP
        .WORD   CGP
        .BLKW   48.

        .END
```

```
; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp ProcOp
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp QX.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "qx.mac")


        .TITLE  QX-11 FLOPPY HANDLER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       SYSTEM TABLE CONTENTS                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=250
        QX$INT          ; QX FLOPPY INTERRUPT HANDLER
        340             ; MAX PRIORITY

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   4
        .BLKW   3
        .ENDR
        .WORD   INBIT!OUTBIT,QXSTRT,QXABRT
        .WORD   INBIT!OUTBIT!400,QXSTRT,QXABRT
        .PAGE
        .CSECT  QXDRVR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;               Q X - 1 1   FLOPPY HANDLER                             ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

QXUNIT: .WORD   0       ; ADDRESS OF UNIT TABLE ENTRY FOR I/O
QXOFST  = 14

DUMCSW: .WORD           ; START OF PHONY IO Q ENTRY FOR TERAK DRIVER
        .WORD   DUMCSW  ; POINTER TO CSW
DUMIOQ: .WORD           ; DISK BLOCK #
        .WORD           ; SPEC FUNC (0) AND UNIT # IN HIGH BYTE
        .WORD           ; BUFFER ADDRESS
        .WORD           ; WORD COUNT
        .WORD   1       ; ASYNCHRONOUS IO REQUEST
```

```
QXSTRT: ; ENTER HERE TO START FLOPPY IO'S.   WE JUST SET UP A PHONY
        ; IO Q FOR THE RT-11 DRIVER FROM TERAK AND LET IT DO ALL THE
        ; DIRTY WORK.
        TST     QXUNIT  ; ANY IO'S ALREADY GOING?
        BNE     QXSTRT  ; IF SO THEN HANG!
        MOV     R1,QXUNIT       ; NOW IO IS GOING ON QX
        BIS     #BSYBIT,@R1
        TST     (R3)+           ; SKIP PAST IO TYPE
        MOV     R3,@SP          ; SAVE ALL REGISTERS...KLUDGE!!
        MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)
        CLR     DUMCSW          ; CLEAR ANY HARD ERROR PROBS
        MOV     #DUMIOQ,R5      ; POINT R5 AT IO Q...READY TO BUILD IT
        MOV     <UBLOCK+QXOFST>(SP),(R5)+
        MOV     @R1,(R5)+       ; UNIT # (SPECIAL FORMAT IN MY UNITABLE)
        MOV     <UBUFFR+QXOFST>(SP),(R5)+
        MOV     <URLENG+QXOFST>(SP),R0
        ROR     R0              ; IN WORD COUNT (CBIT IS CLEAR)
        TST     -(R3)           ; CHECK IF READ OR WRITE
        BNE     1$              ; 1 IS A READ...0 IS A WRITE
        NEG     R0              ; NEGATIVE WORD COUNT IF A WRITE
1$:     MOV     R0,(R5)+        ; FINALLY WORD COUNT SET IN Q EL
        JSR     PC,QENTRY       ; AND START UP DRIVER
        MOV     (SP)+,R5        ; NOW RESTORE THOSE REGS
        MOV     (SP)+,R4
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOVB    DUMCSW,@R1      ; SEND POSSIBLE ERROR BACK TO USER
        JMP     @(SP)+          ; RETURN TO UIO NOW (OLD R3 VAL @SP)

$INTEN: ; FAKE $INTEN FOR DRIVERS BENEFIT
        MOV     R4,-(SP)        ; R5 ALREADY ON STACK
        JSR     PC,@R5
        MOV     (SP)+,R4
        MOV     (SP)+,R5
        JMP     @#INTRTN

QXQUIT: ; THIS IS THE DUMMY QMANAGR ENTERED UPON IO COMPLETE
        MOVB    DUMCSW,@QXUNIT  ; PLACE POSSIBLE HARD ERROR IN RESULT BYTE
        BIC     #BSYBIT,@QXUNIT
        CLR     QXUNIT
        RTS     PC              ; FALLS INTO $INTEN
        .PAGE
;QX CONTROLLER REGISTERS
        QXCS = 177000
        QXDB = 177002
;COMMAND BITS
        QXENBL = 1
        QXRTC  = 2
        QXSTPN = 4
        QXSTPO = 6
        QXRTS  = 10
        QXREAD = 12
        QXWRT  = 14
        QXWRTD = 16
```

```
        QXHEAD = 20
        QXCRNT = 40
        QXNTRP = 100
        WRTFLG = 200
        QXTRK0 = 1000
        QXDDAT = 2000
        QXWRTP = 4000
        QXCRC  = 10000
        QXSYNC = 40000
QXCQE:  .WORD DUMIOQ            ;CURRENT QUEUE ELEMENT
QENTRY: MOV #QXCS, R5           ;SET UP CS REG POINTER
        MOV QXCQE, R1           ;GET CURRENT QUE ELEMENT POINTER
        MOV (R1)+, R3           ;R3 HAS BLOCK #
        INC  R1                 ;SKIP SPEC FUNCTION BYTE
        MOVB (R1)+, R4          ;GET UNIT NMBR
        BIC #^C7, R4            ;CLEAN OFF JOB NUMBER
        CMP R4, #3              ;0 THRU N ONLY
        BHI HERHOP              ;OTHERWISE...HARD ERROR
        CMPB R4, CMDMSK+1       ;IS THIS A DIFFERENT UNIT?
        BEQ BUFCMD              ;NO -> DON'T BOTHER W/TRK PNTR
        SWAB R4                 ;SET NEW UNIT NUMBER
        MOV R4, (R5)            ;INTO HARDWARE...DONE HERE
        SWAB R4                 ;IN CASE OF SPFUN 374
        MOVB R4, CMDMSK+1       ;INSTALL IN MASK
        ADD PC, R4              ;PIC POINTER TO
        ADD #TRKHST-., R4       ;TRACK HISTORY BYTES
        MOV R4, (PC)+           ;REF'D INDIRECTLY
TRACK:  .WORD 0                 ;THRU HERE
BUFCMD: MOV (R1)+, R4           ;R4 HAS BUFFER ADDRESS
        MOV #QXREAD!QXENBL, R0  ;ASSUME A READ COMMAND
        MOV (R1), R1            ;R1 HAS WORD COUNT
        BPL STNDRD              ;TAKE ABS VALUE
        MOV #WRTFLG!QXWRT!QXENBL, R0    ;NEG WRD CNT => WRITE
        ;WRTFLG IS A DON'T CARE BIT...MAKES READ/WRT CHK EASIER
        NEG R1                  ;TAKE ABSLUT VAL
STNDRD: ASL R3                  ;BLOCK NMBR * 4 =
        ASL R3                  ;LOGICAL SECTOR NMBR (LSN)
        MOV R3, LSNREQ          ;PUT LSN AT HELM
        JSR PC, ABCOMP          ;BEAT INTO ABSOLUTE TRK/SEC
        ASL R1                  ;WRDCNT -> BYTE COUNT
        ADD R4, R1              ;PLUS BUF START = BUFFER END
        MOV R1, BUFEND          ;AND THAT'S WHERE WE PUT IT!
        MOV R4, BUFADR          ;SAVE BUFFER POINTER
        MOV PC, R1              ;SET UP MARK
        ADD #QXTOP-.,R1         ;FOR INITL INTRPT
        MOV R1, MRKPTR
        MOV CMDMSK, R1          ;GET OLD MASK(HAS UNIT ALREADY)
        MOV #QXHEAD, R2         ;HEAD BIT MASK!
        BIS R2, R1              ;ASSUME HEAD DOWN
        BIS R1, R0              ;AND INSTALL TO DATA COMMAND
        MOV R0, CMD             ;SAVE NEW DATA COMMAND
        MOV R1, R0              ;NEED ANOTHER COPY
        ADD #QXNTRP!QXRTS!QXENBL, R0    ;NTRPT NBL OFF, RTS IN
        MOV R0, CMDTS           ;INSTALL TO TRKSEC COMMAND
        BIT R2, (R5)            ;WHERE IS REAL HEAD??(UNIT SAME)
        BNE 1$                  ;DOWN...GUESS WAS O.K.
        BIC R2,R1               ;TURN OFF HEAD BIT
1$:     MOV R1, CMDMSK          ;SAVE NEW MASK
        MOV R1, (R5)            ;KICK THE CONTROLLER
        RTS PC                  ;FOR FIRST INTERRUPT
```

```
        ;
HERHOP: CMP -(SP), -(SP)        ;LOAD STACK
HRDHOP: JMP HRDERR              ;WAY STATION FOR HARD ERRORS
TRKHST: .BYTE 200,200,200,200   ;NEG TRK HISTORY BYTE->NEW LOAD

QXTOP:  MOV CMDMSK, (R5)        ;CHECK THAT DOOR!!
        TST (R5)                ;AND THAT IT'S READY
        BMI HRDHOP              ; GIVE HARD ERROR FOR DRIVE NOT CLOSED
ERCOMP: CLR RESTEP              ;SET UP TRACK MISS CNTR
        MOV #8., (PC)+          ;RESET THE RETRY COUNTER
RTRIES: .WORD 0                 ;EACH TIME WE SEEK A NEW SECTOR
RESEEK: MOVB TRKREQ, R2         ;COMPUTE TRACK ERROR
        MOVB @TRACK, R3         ;BETWEEN HISTORY & REQUEST
        BPL 2$                  ;IS HISTORY BYTE VALID??
        MOVB R2, @TRACK         ;NO!...ASSUME HEAD POSITION
        MOV R2, R3              ;IS CORRECT TO CAUSE TRKSEC CHK
2$:     SUB R2, R3              ;SUBTRACT BYTES
        MOV R3, R2              ;NEED SIGNED AND UNSIGNED ERR
        BGE 1$                  ;ABSVAL INTO R
        NEG R2                  ;SIGNED IS IN R3
1$:     CMP R2, #4              ;AT 4 OR LESS STEPS,
        BHI HDUPCK              ;SET HEAD DOWN
        BIT #QXHEAD, (R5)       ;IS HEAD DOWN ALREADY??
        BNE ERRCHK
        BIS #QXHEAD, CMDMSK     ;SET HEAD DOWN BIT IN MSK
        MOV CMDMSK, (R5)        ;AND INTO CONTROLLER
        MOV #^D<50/2>, HDTIME   ;RST HD SETTLING TIME
ERRCHK: TST R2                  ;TRK ERR ZERO ???
        BEQ SETCHK              ;YES>>CHECK HEAD SETTLING
        BR MOVHD                ;NO>>MOVE IT!!!
        ;
HDUPCK: CMP R2, #10             ;IF OVER 8 STEPS REQD,
        BLOS MOVHD              ;WE CAN UNLOAD THE HEAD
        BIC #QXHEAD, CMDMSK     ;HEAD UP WITH NEXT STEP
MOVHD:  TST R3                  ;GET DIRECTIONS..IN OR OUT
        BPL MOVHDO
MOVHDN: JSR PC, STEPN
        BR MOVMRK
MOVHDO: JSR PC, STEPO
MOVMRK: MOV #^D<6/2>, R4        ;SET UP STEP WAIT TIME
        DEC R2                  ;WILL THIS BE LAST STEP?
        BGT 1$
        MOV #^D<26/2>, R4       ;UP TIMEOUT TO 26 MS
1$:     MOV R4, STPTIM          ;SAVE TIMEOUT FIGURE
        SUB R4, HDTIME          ;NUDGE TIMER
        JSR PC, MARK            ;FIRST CATCH STEP INTRPT
        JSR PC, TIME            ;WAIT FOR STEP/HEAD
STPTIM: .WORD 0
        BR RESEEK               ;RE FIGURE ERRORS
        ;
SETCHK: TST HDTIME              ;TIME-OUT REQUIRED??
        BLE HDSTOK              ;IF NEG, IT'S SETTLED
        JSR PC, TIME            ;SAVES SOME TIME WHEN TIMEOUT=0
HDTIME: .WORD 0
HDSTOK: CLR HDTIME              ;IN CASE OF ERRS!
        TSTB CMD                ;WRT OR WD FLAG ON??
        BPL TSREAD              ;NO>>XFR DATA AFTERWARDS
        ;
        ;WRITE DATA TRANSFER
        ;
```

```
WRTMOV: CMPB @TRACK, #53        ;DECIDE ON HEAD CURRENT
        BLOS 2$         ;IF ABOVE 43 (DECML)
        BIS #QXCRNT, CMD        ;SET LOW CURRENT BIT
        BR 1$
2$:     BIC #QXCRNT, CMD        ;CLR LOW CURRENT BIT
1$:     MOV (PC)+, R3           ;R3 INDEXES BUFFER
BUFADR: .WORD 0
        MOV #200, R2            ;64 WORDS
        ADD R3, R2              ;R2 -> END OF THIS CHUNK
        MOV BUFEND, R4          ;R4 -> END OF WHOLE BUFFER
        TST (R5)+       ;R5 PTS AT QXDB (ALSO RESET HDWR PTR)
DATAO:  CMP R3, R4              ;DOES R3 POINT ABOVE
        BHIS SLUFF              ;END OF BUFFER ??
        MOV (R3)+, (R5)         ;NO..MOV WRD TO QX BFR
WRTADV: CMP R2, R3              ;WAIT FOR 200
        BHI DATAO               ;LOOP DE LOOP
        TST -(R5)               ;RESTORE PTR TO QXCS
        BR TSREAD
SLUFF:  MOV #0, (R5)            ;YES..SLUFF ZEROS
        ;NOTE...A CLR DOES NOT WORK!! (CNTRLR IS FINICKY)
        TST (R3)+               ;ADV BUFFER
        BR WRTADV
        ;
TSWAIT: JSR PC, TIME            ;WAIT 4 MS OUT OF 6
        .WORD ^D<4/2>
TSREAD: MOV (PC)+, (R5)         ;READ TRACK/SECTOR ID
CMDTS:  .WORD 0                 ;RTS SANS NTRPT NBL
1$:     TSTB (R5)               ;WAIT ON DONE...STAGED!!
        BPL 1$                  ;SPIN WHEELS &...GREEN LIGHT
        ;WAIT ON DONE BECAUSE CAN'T ABIDE F/B NTRPT LATENCY
        MOV (R5)+, R4           ;ANY ERRORS???
        BPL UPDATE
        TST -(R5)               ;RESTORE CS PTR
        BIT #76000, R4          ;USE ONLY ONE COPY OF ERRS
        BEQ QXTOP               ;READY->DOOR OPENED->WAIT
        ;NOT-READY TOLERATED IF DURING READ I.D.
        JSR R5, RETRY           ;GET CLEARANCE FOR A RETRY
        BR TSWAIT               ;1ST 10 TRIES..RE-READ TS
        BIT #QXSYNC!QXTRK0, R4  ;10 TO 20 RETRIES...
        ;ANY ERR ON TRK 0->STEP IN
        ;NO SYNC ->MAYBE TRK 77->STEP OUT
        BNE ERRSTP              ;NO..THEN JOG HEAD
TSJOG:  JSR PC, STEP            ;STEP IN SAME DX AS LAST TIME
TSX:    JSR PC, MARK            ;WAIT ON STEP DONE
        JSR PC, TIME            ;BECAUSE DX OF STEP REVERSED
        .WORD ^D<26/2>          ;WAIT 26 MS
        BR TSREAD               ;AND RE-READ T/S
        ;
ERRSTP: BIT #QXTRK0, R4         ;TRK 0 ERROR??
        BNE 1$                  ;YS..STEPN / NO..STEPO
        JSR PC, STEPO           ;NO SYNC -> OUT OF TRK 77
        BR TSX                  ;USE COMMON MARK&TIME
1$:     JSR PC, STEPN           ;TRK0 -> STEP N
        BR TSX
        ;
UPDATE: CMP (R5), (PC)+         ;MATCH??
TRKREQ: .BYTE 0
SECREQ: .BYTE 0
        BEQ DATA                ;GOT IT!!
        MOV (R5), R4            ;GET ANOTHER COPY
```

```
        BPL UPDATX              ;DELETED TRACK??
        TST -(R5)               ;ADJ PTR
        BR TSJOG                ;DELTD TRK -> STEP OVER
UPDATX: TST -(R5)               ;ADJ PTR
        CMPB R4, TRKREQ         ;DID WE FIND RIGHT TRACK??
        BEQ TSWAIT              ;TRK O.K.->GO WAIT FOR SECTOR
        COM (PC)+               ;MIS-CALC'D TRACK!!
RESTEP: .WORD 0                 ;COUNT TRK  RECHECKS
        BMI TSWAIT              ;ALLOW ONE
        MOVB R4, @TRACK         ;UPDATE TRACK IMAGE
        JSR R5, RETRY           ;CLEAR FOR RETRY???
        NOP
        JMP RESEEK              ;UP TO 20 SHOTS FOR A DIME
        ;
DATA:   MOV (PC)+, -(R5)                ;ISSUE WHATEVER
CMD:    .WORD 0 ;ALREADY HAS UNIT,ENABLE,HEAD BITS (ALSO WRTFLG)
        INC LSNREQ              ;COMPUTE NEXT ABS SEC NMBR
        JSR PC, ABCOMP
        JSR PC, MARK            ;WAIT FOR COMPLETION
        MOV (R5), R4            ;ANY ERRORS?
        BPL READCK
        BIT #QXWRTP, R4         ;WRITE PROTECT VIOLATN?
        BNE HRDERR              ;YES..WRT CMD IS IMPLICIT
        BIT #QXDDAT, R4         ;DELETED DATA MARK??
        BEQ DATRTY              ;NO..GET SERIOUS THEN!!!
READCK: TSTB CMD                ;WAS THIS A READ COMMAND??
        BMI MOPUP               ;NO..SKIP DATA XFR
REDMOV: MOV (PC)+, R4           ;XFR 64 OR LESS WRDS
BUFEND: .WORD 0
        TST (R5)+       ;R5 -> DATA REG (ALSO RESET HDWR PNTR)
        MOV BUFADR, R3          ;DEPENDING ON WRD CNT
        MOV #100, R2            ;R2 & R4 COUNT XFR
2$:     CMP R3, R4              ;WORD CNT GETS FIRST SHOT
        BHIS 1$                 ;TO STOP XFR
        MOV (R5), (R3)+         ;MOVE A WORD
        SOB R2, 2$              ;OR 64 WORDS MAXIMUM
1$:     TST -(R5)
MOPUP:  ADD #200, BUFADR        ;ADV BFR PTR
        CMP BUFADR, BUFEND      ;JOB DONE???
        BHIS 2$                 ;YES...CHECK FOR WRITE SLUFF
1$:     JMP ERCOMP      ;GET ANOTHER SECTOR + RESET ERRORS
        ;
2$:     TSTB CMD                ;WAS THIS A WRITE/WRITE-D??
        BPL QXDONE              ;READ GETS TO GO HOME
        BIT #3, LSNREQ  ;WRITE MUST SLUFF TO END OF LOGICAL BLOCK
        BNE 1$          ;-> MULT OF 4 IN LOGICAL SECTOR
        BR QXDONE               ;SCHOOL'S OUT!!
        ;
DATRTY: DEC LSNREQ              ;ADJUST TO OLD LOGICAL SECTOR
        JSR PC, ABCOMP          ;CORRECT ABS T&S VALUES
        BIT #76000, R4          ;READY BIT ONLY??
        BNE DATRTZ
        BR HRDERR               ;DOOR OPENED DURING READ/WRITE
DATRTZ: JSR R5, RETRY           ;CLEAR FOR RETRY
        BR TSWAIT               ;RE-READ 10 TIMES, THEN
        BR TSJOG                ;JOG HEAD 10 TIMES
        ;
        ;TIME MARKER
        ;CALLED BY:     JSR PC, TIME
        ;               .WORD X
```

```
        ;X = TIME REQUEST IN 2 MS LUMPS
TIME:   MOV (SP)+, R4           ;POP STACK
        MOV (R4)+, TICKER       ;GET TIME + ADV RETURN
        MOV R4, TIMPTR          ;SAVE RETURN W/O STACK
TIMEX:  JSR PC, TICK            ;ASK FOR A TICK
        JSR PC, MARK            ;WAIT FOR 2 MS
        DEC (PC)+               ;BUMP COUNT
TICKER: .WORD 0
        BGT TIMEX               ;LOOP
        MOV (PC), PC            ;RETURN
TIMPTR: .WORD 0
        ;
RETRY:  DEC RTRIES              ;KNOCK RETRY COUNT
        BLE RETRY1              ;THRU 0 ADV RETURN
        RTS R5                  ;FIRST 10 RETURNS
RETRY1: CMP RTRIES, #-20.       ;OVER 20 YET??
        BLE RETRY2
        TST (R5)+               ;ADVANCE RETURN
        RTS R5                  ;IF TWIXT 10 & 20
RETRY2: MOV (SP)+, R5           ;REPAIR STACK, NO MORE TRYS
HRDERR: MOV QXCQE, R4           ;GET CHAN STAT WRD
        BIS #1, @-(R4)          ;SET HARD ERROR BIT
QXDONE: MOV CMDMSK, R4          ;GET CRNT UNIT & HEAD
        BIC #QXNTRP, R4         ;TURN OFF INTERRUPTS
        MOV R4, (R5)
        MOV (SP)+, R3           ;RESTORE REGS
        MOV (SP)+, R2
        JMP QXQUIT
        ;
MARK:   MOV (SP)+, MRKPTR       ;POP RETURN ADDRESS
        MOV (SP)+, R3           ;RESTORE REGS
        MOV (SP)+, R2
        RTS PC
        ;
QX$INT: JSR R5, $INTEN          ;ENTER SYSTEM STATE
        MOV R2, -(SP)           ;PUSH A FEW REGS
        MOV R3, -(SP)           ;
        MOV #QXCS, R5           ;ALWAYS SET UP QXCS PTR
        MOV (PC), PC            ;RETURN TO MARK CALLER
MRKPTR: .WORD 0
        ;
        ;
QXABRT: MOV     #QXDONE,MRKPTR  ; NEXT INTERRUPT KILLS IO
1$:     TST     QXUNIT          ; NOW HANG UNTIL IO IS DONE
        BNE     1$
        RTS     PC

STEPN:  MOV #105277, STEP       ;GET TRACK BUMPER
                                ;105277 = INCB @(PC)
        MOV #QXSTPN!QXENBL, STPINS      ;GET COMMAND
        BR STEP                 ;GO STEP
        ;
STEPO:  MOV #105377, STEP       ;GET TRACK BUMPER
                                ;105377 = DECB @(PC)
        MOV #QXSTPO!QXENBL, STPINS      ;FALL INTO STEP
        ;
STEP:   INCB @TRACK             ;MAY BE INCB OR DECB
        MOV (PC)+, R4           ;GET COMMAND
STPINS: .WORD QXSTPN!QXENBL     ;MAY BE QXSTPN OR QXSTPO
        BR CMDASY
```

```
         ;
         ;
 TICK:   MOV #QXRTC!QXENBL, R4   ;2 MS TIME COMMAND
         ;
 CMDASY: BIS (PC)+, R4           ;STUFF IN UNIT, INTRPT ENBL,
 CMDMSK: .WORD QXNTRP!QXCRC  ;HEAD UP/DWN W/ INT ENABLED
         ;QXCRC-> UNIT CHANGE WHEN FIRST LOADED
         MOV R4, (R5)            ;STUFF CMD AT QX FLOPPY
         RTS PC                  ;R5 MUST => QXCS
         ;
         ;ABCOMP.........CONVERT LSNREQ TO ABSOLUTE
         ;SECREQ AND TRKREQ USING ONLY R2 & R3
         ;DEC 2:1 INTERLEAVE OF SECTORS IS SUPPORTED
         ;ALSO 6 SECTOR PER TRACK INCREMENT SKEW
         ;PLUS TRACK 0 CANNOT BE REACHED
         ;
 ABCOMP: MOV (PC)+, R3           ;GET LSN REQUEST
 LSNREQ: .WORD 0
         BMI ABCOM4              ;SIGN FLAGS SPFUN -> SKIP
         MOV #10, R2             ;COUNT 8 BITS OF DIVIDE
 ABCOM1: CMP #6400, R3   ;DOES 26 GO INTO DIVIDEND
         BHI ABCOM2              ;BR IF NO, C CLR
         ADD #171400, R3         ;SUB 26 AND C SET
 ABCOM2: ROL R3                  ;SHIFT IN QUOTIENT (C BIT)
         SOB R2, ABCOM1          ;COUNT 8 BITS
         MOVB R3, R2             ;COPY TRK NMBR TO R2
         CLRB R3
         SWAB R3         ;SIGN XTND SECTOR
         CMP #14, R3             ;C=1 IFF 13<R3<26
         ROL R3                  ;2:1 ,C SETS ODD/EVEN GROUP
         ASL R2                  ;ADD IN TRACK SKEW
         ADD R2, R3              ;ADD IN 6 X TRACK
         ADD R2, R3              ; (ALL X 2)
         ADD R2, R3
         ASR R2                  ;REPAIR TRACK NMBR
         INC R2                  ;SKIP TRACK 0
 ABCOM3: SUB #32, R3             ;MODULO 26 THE SECTOR #
         BGE ABCOM3
         ADD #33, R3             ;ADJ TO 1 TO 26
         MOVB R3, SECREQ         ;PUT EM AWAY
         MOVB R2, TRKREQ
 ABCOM4: RTS PC

         .END


; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp QX.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp QXBOOT
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "qxboot.mac")


        .NLIST  TTM
        .TITLE  QX-11 FLOPPY BOOTSTRAP LOADER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;
R0=%0
R1=%1
BK=%2
R2=%2
BASE=%3
R3=%3
IPC=%4
R4=%4
MP=%5
R5=%5
SP=%6
PC=%7
BOOT7G = 10110
RCSR = 177560
XCSR = 177564
XCSR = 177564
XBUF = 177566
DTID = 6
DELENG = 26.
CR = 15
LF = 12
BUZZ = 7
VCR = 177744    ; TEST ADDRESS FOR 8510A
NSEGS = 16.
DATASZ = -10    ; OFFSET TO LOCAL DATA SIZE IN JTAB
SYUNIT = 404
SEG    = 422
MEMTOP = 424
MSCINFO = 472
SEGTBL = 540
        ;
        ;STAGE II RESIDES ON SECTOR 1, TRACK 1 OF UNIT 0
        ;UNIT 0 ONLY IS ALWAYS BOOTSTRAPED
        ;BOOTSTRAP STAGE I IS
        ;A READ ROUTINE TO LOAD
        ;THE FIRST SIX BLOCKS...THIS INCLUDES
        ;THE REMAINING LOADER AND THE DISK DIR.
```

```
        ;
        .ASECT
        . = 0
        MOV R0, R1              ;VALIDATION CODE, AND
                                ;UPDATE BUFFER POINTER
        BR BOOT1                ;JMP AROUND VECTORS
        .WORD NXM
        .WORD 340
        .WORD 173000            ; BAD INSTRUCTION REBOOTS
        .WORD 340
        .WORD QREADS            ;BPT CALLS READ SUB THRU HERE
        .WORD 340
        .WORD QWAIT             ;IOT CALLS WAIT ROUTINE HERE
        .WORD 340
        . = 34
        .WORD   READ            ;CALL DRIVER VIA TRAP
        .WORD 340
BOOT1:  MOVB @(PC)+, R5         ;NXT SECTR + TRAP P.S.W.
1$:     .WORD SECLST            ;LIST OF BYTES
        BEQ BOOT                ;0->GO TO REAL BOOT
        DEC 1$                  ;ADV PNTR...GO THRU HERE ONCE!!
        MOV #BOOT7G, PC         ;GO BACK TO STAGE 1 BOOT
        ;
        . = 100                 ; LINE CLOCK HANDLER
        .WORD   102
        RTI
        ; HANDY DISK ROUTINE HERE OUT OF THE WAY
TMARK:  MOV (R5)+, -(SP)        ;GET # OF 2MS TICKS REQ'D
1$:     MOV #XCLK, (R4)         ;HIT CONTROLLER
        IOT                     ;MARK TIME
        DEC (SP)                ;COUNT TICKS
        BNE 1$
        TST (SP)+               ;ADJUST STACK
        RTS R5                  ;WAKE HIM UP
        .BYTE 0
        .BYTE 26
        .BYTE 24
        .BYTE 22
        .BYTE 20
        .BYTE 16
        .BYTE 14
        .BYTE 12
        .BYTE 10
        .BYTE 6
        .BYTE 4
        .BYTE 2
        .BYTE 31
        .BYTE 27
        .BYTE 25
        .BYTE 23
        .BYTE 21
        .BYTE 17
        .BYTE 15
        .BYTE 13
        .BYTE 11
        .BYTE 7
        .BYTE 5
SECLST: .BYTE 3
        .EVEN
        ;
```

```
        ;
        ;
        ;I/O FOR THE REMAINDER OF THE BOOTSTRAP
        ;IS NOW SUPPORTED BY A DRIVER, WHO
        ;IS SUPPORTED BY A READ-ONLY HANDLER
        ;I/O REQUESTS ARE FOR LOGICAL BLOCK NMBRS
        ;DRIVER USES FOLLOWING CALL PARAMETERS
        ;               R0 = LOGICAL BLOCK NUMBER
        ;               R1 = WORD COUNT
        ;               R2 = BUFFER LOAD POINT
        ;               R3,R4,R5 MAY BE DESTROYED
        ;HNDLR MUST GOTO BIOERR ON FATAL ERROR
READ:   ASL     R0              ;CONVERT BLOCK TO LOGICAL SECTOR
        ASL     R0              ;LSN=BLOCK*4
1$:     MOV     R0,-(SP)        ;SAVE LSN FOR LATER
        MOV     R0,R3           ;WE NEED 2 COPIES OF LSN FOR MAPPER
        MOV     R0,R4
        CLR     R0              ;INIT FOR TRACK QUOTIENT
        BR      3$              ;JUMP INTO DIVIDE LOOP
2$:     SUB     #23.,R3         ;PERFORM MAJIK TRACK DISPLACEMENT
3$:     INC     R0              ;BUMP QUOTIENT, STARTS AT TRACK 1
        SUB     #26.,R4         ;TRACK = INTEGER(LSN/26)
        BPL     2$              ;LOOP - R4=REM(LSN/26)-26
        CMP     #-14.,R4        ;SET C IF SECTOR MAPS TO 1-13
        ROL     R3              ;PERFORM 2:1 INTERLEAVE
4$:     SUB     #26.,R3         ;ADJUST SECTOR INTO RANGE -1,-26
        BPL     4$              ;(DIVIDE FOR REMAINDER ONLY)
        ADD     #27.,R3         ;NOW PUT SECTOR INTO RANGE 1-26
        BPT                     ;CALL READ ONLY HANDLER
        MOV     (SP)+,R0        ;GET THE LSN AGAIN
        INC     R0              ;SET UP FOR NEXT LSN
        TST     R1              ;WHATS LEFT IN THE WORD COUNT
        BGT     1$              ;BRANCH TO TRANSFER ANOTHER SECTOR
NOBOMB: BIS     #1,2(SP)        ; SET CARRY IN RETN PSW
        RTI                     ;RETURN/UNTRAP

        .NLIST  BEX
BIOERR: JSR     R0,BOMB
        .ASCIZ  <CR><LF><BUZZ>'?IO ERROR WHILE BOOTING?'<CR><LF>
        .EVEN

NOCORE: JSR     R0,BOMB
        .ASCIZ  <CR><LF><BUZZ>'?NOT ENOUGH CORE TO BOOT?'<CR><LF>
        .LIST   BEX
        .EVEN

        . = 400
        .ENABL LSB
BOOT:   MOV     #20000,SP       ;SET STACK POINTER
        ; CORE DETERMINATION
        CLR     R2              ;LOOK FOR TOP OF CORE
2$:     ADD     #4000,R2        ;MOVE TO NXT 1K BANK
        CMP     R2, (PC)+       ;REACHED 28K YET ?
$MEMRY: .WORD 160000    ;CHANGE HERE TO LOWER TOP O' MONITR
        BEQ     NXM             ;YES, DO A 28K SYSTEM
        TST     @R2             ;NO, SEE IF THIS LOC EXISTS
        BR      2$              ;KEEP GOING IF WE DIDN'T TRAP
        .DSABL LSB
NXM:    MOV     R2,MEMSIZ       ; STASH MEMORY SIZE FOR LATER
        CMP     R2,#100000      ; DO WE HAVE AT LEAST 16K?
```

```
        BLO     NOCORE
        MOV     #NOBOMB,@#4     ; AVOID BLOWUP FOR BAD MEM NOW
        MOVB    #14,@#177566    ; SEND FORMFEED TO CONSOLE DEVICE
        MOVB    #14,@#177766    ; ALSO TO POSSIBLE TERAK SCREEN
        JSR     R0,DIRSRCH      ; FIND THE CODE FILE FOR THE SYSTEM
        .ASCIZ  <15>'SYSTEM.PASCAL?'<CR><LF>
        MOV     (R1)+,FSTSYS    ; SAVE FIRST BLOCK FOR SYSTEM CODE
        JSR     R0,DIRSRCH      ; NOW LOOK FOR THE INTERPRETER .SAV FILE
        .ASCIZ  <15>'SYSTEM.PDP-11?'<CR><LF>
        MOV     (R1)+,R0        ; BLOCK # OF INTERP
        MOV     @R1,R1          ; LAST BLOCK # IN INTERP
        SUB     R0,R1           ; NOW R1 IS # BLOCKS TO READ
        SWAB    R1              ; MAKE # WORDS FOR READ
        MOV     R1,INTSIZ       ; SAVE INTERPRETER SIZE FOR LATER
        MOV     #20000,R2       ; MEM ADDR TO READ INTERP INTO
        TRAP                    ; PERFORM DISK READ
        MOV     FSTSYS,R0       ; NOW READ SEGTBL FROM PASCAL CODE INTO INTERP
        MOV     #NSEGS*2,R1     ; # WORDS TO READ
        MOV     #DIREC,R2       ; AND THE MEMADDR
        TRAP                    ; PERFORM THE READ
        MOV     #DIREC,R0       ; SOURCE OF SEGDESC...THEN
        MOV     #20000+SEGTBL,R2 ; RELOCATE DISK ADDRS
        MOV     #NSEGS,R1       ; LOOP COUNTER
1$:     MOV     #4,(R2)+        ; UNIT # ALWAYS 4
        MOV     (R0)+,@R2       ; COPY IN REL DISK BLOCK
        ADD     FSTSYS,(R2)+    ; ADD START ADDR TO REL ADDR ALRDY THERE
        MOV     (R0)+,(R2)+
        SOB     R1,1$           ; FOR ALL SEGS...LOOP
        MOV     #20004+SEGTBL,R0 ; POINT R0 AT LENG OF SEG 0
        MOV     MEMSIZ,R2       ; SET UP MEM ADDR TO READ ROUTINE
        SUB     @R0,R2          ; GET HIGH ADDR...SUBTRACT CODE LENGTH
        MOV     R2,SP           ; THIS IS STACK FOR SYSTEM ENTRY..STASH IT
        MOV     @R0,R1          ; NOW # WORDS TO READ
        ASR     R1              ; MAKE # WORDS...ASSUME < 32K BYTES
        MOV     -(R0),R0        ; FINALLY, GET DISK ADDR
        TRAP                    ; AND READ IN SYSTEM CODE
        TST     -(R2)           ; R2 WAS ABOVE HIGH MEM...NOW @ HIGH MEM WORD
        MOV     R2,20000+SEG    ; SET UP SEG STATE IN INTERP
        MOV     R2,20000+MEMTOP ; SET TOP OF MEM PTR...USED IN CXP
        MOV     #4,20000+SYUNIT ; SET UP SYSTEM UNIT #
        SUB     -(R2),R2        ; R2 NOW POINTS @ JTAB OF OUTER BLOCK
        MOV     SP,MP           ; SET UP MP & BASE TO CBP WILL
        SUB     DATASZ(R2),MP   ; TO THEMSELVES
        MOV     #400,@MP        ; FUNNY PARAM TO SYSTEM!!!!!!!!!
        SUB     #14,MP
        MOV     MP,BASE         ; ALL REGS SET UP NOW
        CLR     -(SP)           ; SET UP FOR RTI
        MOV     20040,R0        ; GRAB INTERP ENTRY POINT
        MOV     R0,-(SP)        ; AND PUSH ON STACK FOR RTI
        CLR     @#VCR           ; RESET SCREEN...IMPLICITLY SEE IF THERE
        BCC     10$             ; IF NO ERROR THEN GO ON
        MOV     20010(R0),R0    ; GRAB ADDR OF UNITABLE...GROSS!!!!!
        CLR     20022(R0)       ; ZAP GRAPHICS DEVICE ENTRY
        BIC     #2,20000+MSCINFO        ; ZAP HAS 8510A BIT
10$:    MOV     #FINALE,R0
        MOV     #100000,R1      ; WHERE WE COPY FINALE CODE
        MOV     #<FINEND-FINALE>/2,BK   ; WORD COUNT OF FINALE CODE
FINLOOP:MOV     (R0)+,(R1)+
        SOB     BK,FINLOOP
        MOV     #20000,R0
```

```
        CLR     R1
        MOV     INTSIZ,BK
        JMP     @#100000

FINALE: MOV     (R0)+,(R1)+
        SOB     BK,FINALE
        BIS     #100,@#RCSR
        MOV     PC,IPC
        ADD     #CBP.OP-.,IPC
        NOP
        RTI
CBP.OP: .BYTE   128.+66.,1      ; CALL BASE PROCEDURE #1
FINEND = .

MEMSIZ: .WORD   ; SIZE OF MEMORY IN BYTES
INTSIZ: .WORD   ; SIZE IN WORDS OF INTERPRETER
FSTSYS: .WORD   ; FIRST DISK BLOCK OF PASCAL CODE FILE
.PAGE
.SBTTL  READ ONLY HANDLER, QX
        ;CALLED BY BPT, USES IOT TO CALL WAIT ROUTINE
        ;ALL I/O ERRORS ARE FATAL & WILL REQUIRE RE-BOOT
        ;
        ;ENTRY THRU BPT (LOC 14 & 16)
        ;ON ENTRY:      R0 = DESIRED TRACK
        ;               R1 = RUNNING WORD COUNT
        ;               R2 = BUFFER START ADDRESS
        ;               R3 = DESIRED SECTOR
        ;               R3, R4, R5 ARE DESTROYED
        ;               R2 IS RETURNED PNTING AT EOBUF
        ;
        QXCS = 177000
        QXDB = 177002
        XRTS = 31               ;HEAD DWN/READ TRKSEC/UNIT 0
        XREAD = 33              ;HEAD DWN/READ/UNIT 0
        XSEEKN = 25             ;HEAD DOWN/STEP IN/UNIT 0
        XSEEKO = 27             ;HEAD DOWN/STEP OUT/UNIT 0
        XHEAD = 20              ;HEAD DWN/UNIT 0
        XCLK = 23               ;HEAD DWN/FIRE 2MS CLOCK
        ;
        ;
        .MACRO TIME X           ;MACRO TO MARK TIME
        JSR R5, TMARK
        .RADIX 10
        .IF GE <X-32767>
        .WORD 16384
        .IFF
         .IF LE X
         .WORD 1
         .IFF
        .WORD <X+1>/2           ;IN 2 MS. LUMPS
         .ENDC
        .ENDC
        .RADIX
        .ENDM
        ;
.PAGE
QREADS: MOV #QXCS, R4           ;R4 IS STATUS ADR
        SWAB R3                 ;PUT SECTOR IN UPPER BYTE
        BIT #XHEAD, (R4)        ;IS HEAD DOWN??
        BNE READZ               ;SKIP HED DOWN
```

```
        MOV #XHEAD, (R4)        ;HEAD DOWN
        TIME 50                 ;WAIT 50 MS
READZ:  MOV R4, R5
        MOV #XRTS, (R5)+        ;READ TRACK SECT
        IOT
CKTRK:  MOV (R5), R5            ;GET SECTOR/TRACK
        BPL 1$                  ;SKIP IF NOT DELETED
        CLR R5                  ;SKIP IN SAME DX
        BR SEEKWT               ;AS LAST TIME
1$:     SUB R0, R5              ;CHECK VS. REQUEST
        TSTB R5                 ;TRACK BYTE ONLY
        BEQ CKSCTR              ;FOUND !!!
        BLT SEEKIN
SEEKOT: MOV #XSEEKO, SKIPX      ;MARK DIRECTION
SEEKWT: MOV SKIPX, (R4)         ;STEP WHICHEVER WAY
        IOT                     ;WAIT DONE , CHECK ERRORS
        TIME 6                  ;WAIT 6 MS
        DECB R5                 ;BUMP TRACK ERROR
        BGT SEEKWT              ;STEP FAST TIL ZERO
        TIME 24                 ;WAIT 24 MS MORE(HEAD SETTLE)
        BR READZ                ;GO READ TRKSEC
SEEKIN: MOV #XSEEKN,(PC)+       ;MARK DIRECTION
SKIPX:  .WORD XSEEKN            ;LAST DX STEPPED
        NEGB R5                 ;ABS VAL OF TRK ERR
        BR SEEKWT
CKSCTR: CMP R5, R3              ;SECTOR FOUND ??
        BNE READZ               ;TRY AGAIN IF SECTOR NOT FOUND
MOVBUF: MOV #XREAD, (R4)        ;READ THEM BITS
        IOT
        TST (R4)+               ;GET DB PTR
        MOV #100, R5            ;COUNT 64 WORDS
MOVWRD: DEC R5                  ;BUMP MOD 64 COUNTER
        BLT EXIT                ;64 WORDS XFRED => RETURN
        MOV (R4), (R2)+         ;MOVE ANOTHER TO BUFFER
        SOB R1, MOVWRD          ;LOOP ON WRD CNT ALSO
EXIT:   RTI                     ;THEN RETURN WITH LESS THAN
                                ;64 WORDS TRANSFERED
.SBTTL  HANDLER WAIT ROUTINE
.PAGE
        ;
        ;QWAIT  ROUTINE CALLED BY IOT TO CHK DONE & ERRORS
QWAIT:  TSTB (R4)               ;WAIT ON DRIVE DONE BIT
        BPL QWAIT
        TST (R4)                ;ANY ERRORS???
        BMI 1$                  ;ERROR!!!
        RTI                     ;BACK TO CALLER
1$:     DEC (PC)+               ;BUMP ERROR RETRY COUNT
        .WORD 4                 ;ALLOW 3 RETRIES
        BLE 3$                  ;TOO MANY...HANG UP
        CLR (R4)                ;RETRY O.K. ... RESET ERROR
        JMP BOOT                ;RESTART BOOT
3$:     JMP BIOERR              ;GO TALK ABOUT IT
.PAGE
.SBTTL  MISC ROUTINES FOR BOOTING
DIRSRCH:MOV     #DIREC+DTID,R1  ; R1 POINTS AT DTID OF EACH DIR ENTRY
DIRLOOP:MOV     R1,R4           ; R4 IS USED FOR TITLE COMPARE
        MOV     R0,R3           ; R3 IS TITLE TO LOOK FOR (IN CODE STREAM)
        MOVB    @R0,R2          ; NUMBER OF BYTES IN NAME(STRING VAR)
1$:     CMPB    (R3)+,(R4)+     ; CHECK EACH BYTE FOR EQUAL
        BNE     2$              ; WOOPS, NEQ...CHECK NEXT ENTRY OR BOMB
```

```
         DEC     R2              ; OK SO FAR...DECREMENT LOOP COUNTER
         BPL     1$              ; LOOP FOR LENG+1 CHARS
         ; EUREKA! WE HAVE FOUND IT...RETURN WITH R1 POINTING AT ENTRY
         SUB     #DTID,R1        ; RETURN R1 AT START OF ENTRY
         ADD     #18.,R0         ; POINT R0 PAST STRING IN CODE
         RTS     R0              ; AND RETURN
 2$:     ADD     #DELENG,R1      ; SKIP R1 TO NEXT DIRECTORY ENTRY
         CMP     R1,#ENDDIR      ; CHECK IF WE HAVE GONE OFF END OF DIR
         BLO     DIRLOOP         ; IF NOT, CHECK NEXT ENTRY
         .NLIST  BEX
         JSR     R0,BOMB         ; OH WELL, NO SYSTEM FILE...TIME TO CROAK
         .ASCII  <CR><LF><BUZZ>'?YOU DON'<47>'T HAVE A '<200>
         .LIST   BEX
         .EVEN
         INC     R0              ; SKIP R0 PAST LENGTH BYTE

 BOMB:   MOVB    (R0)+,R1
         BMI     XBOMB           ; IF MINUS IN STRING, RETURN
         BEQ     HALTER
 1$:     TSTB    @#XCSR          ; WAIT UNTIL DL11 DONE BIT
         BPL     1$
         MOVB    R1,@#XBUF
         BR      BOMB
 XBOMB:  ROR     R0              ; RETURN TO USER...WORD BOUND R0
         ADC     R0
         ROL     R0
         RTS     R0

 HALTER: HALT
         BR      HALTER

 BOOTSZ  = . + 777 / 1000
 .       = BOOTSZ * 1000
 DIREC = 2000
 ENDDIR = DIREC + 4000
         .END

; +---------------------------------------------------------------+
; |                                                               |
; |                     F     I     N     I     S                 |
; |                                                               |
; +---------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp QXBOOT
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp RK.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "rk.mac")


        .TITLE  RK-11 (RK05) HANDLER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       SYSTEM TABLE CONTENTS                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=220
        RK$INT
        240

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   9.
        .BLKW   3
        .ENDR
        .WORD   INBIT!OUTBIT,RKSTRT,RKABRT
        .WORD   INBIT!OUTBIT!400,RKSTRT,RKABRT
        .WORD   INBIT!OUTBIT!1000,RKSTRT,RKABRT
        .WORD   INBIT!OUTBIT!1400,RKSTRT,RKABRT
        .PAGE
        .CSECT  RKDRVR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;               R K - 0 5  HARD DISK HANDLER                           ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RKUNIT: .WORD   0       ; ADDRESS OF UNIT TABLE ENTRY FOR I/O
RKOFST  = 14

DUMCSW: .WORD           ; START OF PHONY IO Q ENTRY FOR TERAK DRIVER
        .WORD   DUMCSW  ; POINTER TO CSW
DUMIOQ: .WORD           ; DISK BLOCK #
        .WORD           ; SPEC FUNC (0) AND UNIT # IN HIGH BYTE
        .WORD           ; BUFFER ADDRESS
```

```
        .WORD           ; WORD COUNT
        .WORD   1       ; ASYNCHRONOUS IO REQUEST

RKSTRT: ; ENTER HERE TO START FLOPPY IO'S.   WE JUST SET UP A PHONY
        ; IO Q FOR THE RT-11 DRIVER FROM TERAK AND LET IT DO ALL THE
        ; DIRTY WORK.
        TST     RKUNIT  ; ANY IO'S ALREADY GOING?
        BNE     RKSTRT  ; IF SO THEN HANG!
        MOV     R1,RKUNIT       ; NOW IO IS GOING ON RK
        BIS     #BSYBIT,@R1
        TST     (R3)+           ; SKIP PAST IO TYPE
        MOV     R3,@SP          ; SAVE ALL REGISTERS...KLUDGE!!
        MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)
        CLR     DUMCSW          ; CLEAR ANY HARD ERROR PROBS
        MOV     #DUMIOQ,R5      ; POINT R5 AT IO Q...READY TO BUILD IT
        MOV     <UBLOCK+RKOFST>(SP),(R5)+
        MOV     @R1,(R5)+       ; UNIT # (SPECIAL FORMAT IN MY UNITABLE)
        MOV     <UBUFFR+RKOFST>(SP),(R5)+
        MOV     <URLENG+RKOFST>(SP),R0
        ROR     R0              ; IN WORD COUNT (CBIT IS CLEAR)
        TST     -(R3)           ; CHECK IF READ OR WRITE
        BNE     1$              ; 1 IS A READ...0 IS A WRITE
        NEG     R0              ; NEGATIVE WORD COUNT IF A WRITE
1$:     MOV     R0,(R5)+        ; FINALLY WORD COUNT SET IN Q EL
        JSR     PC,QENTRY       ; AND START UP DRIVER
        MOV     (SP)+,R5        ; NOW RESTORE THOSE REGS
        MOV     (SP)+,R4
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOVB    DUMCSW,@R1      ; SEND POSSIBLE ERROR BACK TO USER
        JMP     @(SP)+          ; RETURN TO UIO NOW (OLD R3 VAL @SP)

$INTEN: ; FAKE $INTEN FOR DRIVERS BENEFIT
        MOV     R4,-(SP)        ; R5 ALREADY ON STACK
        JSR     PC,@R5
        MOV     (SP)+,R4
        MOV     (SP)+,R5
        JMP     @#INTRTN

RKQUIT: ; THIS IS THE DUMMY QMANAGR ENTERED UPON IO COMPLETE
        MOVB    DUMCSW,@RKUNIT  ; PLACE POSSIBLE HARD ERROR IN RESULT BYTE
        BIC     #BSYBIT,@RKUNIT
        CLR     RKUNIT
        RTS     PC              ; FALLS INTO $INTEN
        .PAGE
; RK CONTROL DEFINITIONS:
RKDS    = 177400
RKER    = 177402
RKCS    = 177404
RKWC    = 177406
RKBA    = 177410
RKDA    = 177412

RKCNT   = 10                    ;# ERROR RETRYS
```

```
RKCQE:  .WORD   DUMIOQ

QENTRY: MOV     #RKCNT,(PC)+    ;SET ERROR RETRIES
RETRY:  0                       ;HIGH ORDER BIT USED FOR RESET IN PROGRESS FLAG
        MOV     RKCQE,R5        ;GET Q PARAMETER POINTER
        MOV     @R5,R2          ;R2 = BLOCK NUMBER
        MOV     2(R5),R4        ;R4 = UNIT NUMBER
        ASR     R4              ;ISOLATE UNIT BITS IN HIGH 3 BITS
        ASR     R4
        ASR     R4
        SWAB    R4
        BIC     #^C<160000>,R4
        BR      2$              ;ENTER COMPUTATION LOOP
1$:     ADD     R2,R4           ;ADD 16R TO ADDRESS
        ASR     R2              ;R2 = 8R
        ASR     R2              ;R2 = 4R
        ADD     R3,R2           ;R2 = 4R+S = NEW N
2$:     MOV     R2,R3           ;R3 = N = 16R+S
        BIC     #177760,R3      ;R3 = S
        BIC     R3,R2           ;R2 = 16R
        BNE     1$              ;LOOP IF R <> 0
        CMP     #12.,R3         ;IF S < 12.
        BGT     3$              ;   THEN F(S) = S
        ADD     #4,R3           ;   ELSE F(S)=F(12+S')=16+S'=4+S
3$:     ADD     R3,R4           ;R4 NOW CONTAINS RK ADDRESS
        MOV     R4,DISKAD       ;SAVE DISK ADDRESS
AGAIN:  MOV     RKCQE,R5        ;POINT R5 TO Q ELEMENT
        MOV     #RKDA,R4        ;POINT TO DISK ADDRESS REG
        MOV     (PC)+,@R4       ;PUT IN ADDRESS & UNIT SELECT
DISKAD: 0                       ;SAVED COMPUTED DISK ADDRESS
        CMP     (R5)+,(R5)+     ;ADVANCE TO BUFFER ADDRESS IN Q ELT
        MOV     (R5)+,-(R4)     ;PUT IN BUFFER ADDRESS
        MOV     (R5)+,-(R4)     ;PUT IN WORD COUNT
        BEQ     6$              ;0 COUNT => SEEK
        BMI     5$              ;NEGATIVE => WRITE
        NEG     @R4             ;POSITIVE => READ. FIX FOR CONTROLLER
        MOV     #105,-(R4)      ;START UP A READ
        RTS     PC              ;RETURN TO MONITOR TO AWAIT INTERRUPT
5$:     MOV     #103,-(R4)      ;START UP A WRITE
        RTS     PC              ;AWAIT INTERRUPT
6$:     MOV     #111,-(R4)      ;START UP A SEEK
        RTS     PC              ;AWAIT INTERRUPT

; NOTE THAT THE RTS PC ABOVE SERVES AS THE ABORT ENTRY FOR THE RK

RK$INT: JSR     R5,$INTEN       ;DO IT. INTO MONITOR
        MOV     #RKER,R5        ;POINT TO ERROR STATUS REGISTER
        MOV     (R5)+,R4        ;SAVE ERRORS IN R4,POINT TO RKCS
        TST     RETRY           ;WERE WE DOING A DRIVE RESET?
        BPL     2$              ;NO-NORMAL OPERATION
        TST     @R5             ;YES-ANY ERROR
        BMI     2$              ;YES-HANDLE NORMALLY
        BIT     #20000,@R5      ;RESET COMPLETE?
        BEQ     RTSPC           ;NO-DISMISS INTERRUPT-RK11 WILL INTERRUPT
                                ;AGAIN WHEN RESET COMPLETE
1$:     CLRB    RETRY+1         ;YES-CLEAR RESET FLAG
        BR      AGAIN           ;AND RETRY OPERATION
2$:     CMP     @R5,#310        ;IS THIS FIRST OF TWO INTERRUPTS CAUSED BY SEEK?
        BEQ     RTSPC           ;YES-IGNORE IT.RK WILL INTERRUPT AGAIN
                                ;WHEN SEEK COMPLETE
```

```
         TST     @R5             ;ANY ERRORS?
         BPL     DONE            ;NO-OPERATION COMPLETE
         MOV     #1,@R5          ;YES-RESET CONTROL
 3$:     TSTB    @R5             ;WAIT
         BPL     3$
         DECB    RETRY           ;DECREASE RETRY COUNT
         BEQ     HERROR          ;NONE LEFT-HARD ERROR
         BIT     #110000,R4      ;SEEK INCOMPLETE OR DRIVE ERROR?
                                 ; 100000=DRIVE ERROR
                                 ; 010000=SEEK ERROR
         BEQ     1$              ;NO-RETRY OPERATION
         MOV     DISKAD,@#RKDA   ;YES-RESELECT DRIVE
         MOV     #115,@R5        ;START A DRIVE RESET
         BIS     #100000,RETRY   ;SET FLAG
 RTSPC:  RTS     PC              ;AWAIT INTERRUPT

 HERROR: MOV     RKCQE,R5        ;GET POINTER TO Q ELEMENT
         BIS     #1,@-(R5)       ;GIVE OUR USER AN ERROR IN CHANNEL
 DONE:   CLR     RETRY           ;CLEAR ANY FLAGS
         JMP     RKQUIT

 RKABRT: TST     RKUNIT
         BNE     RKABRT
         RTS     PC
         .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp RK.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp RKBOOT
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "rkboot.mac")


        .NLIST  TTM
        .TITLE  RK05 BOOTSTRAP LOADER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;
R0=%0
R1=%1
BK=%2
R2=%2
BASE=%3
R3=%3
IPC=%4
R4=%4
MP=%5
R5=%5
SP=%6
PC=%7
RCSR = 177560
XCSR = 177564
XCSR = 177564
XBUF = 177566
DTID = 6
DELENG = 26.
CR = 15
LF = 12
BUZZ = 7
NSEGS = 16.
DATASZ = -10    ; OFFSET TO LOCAL DATA SIZE IN JTAB
SYUNIT = 404
SEG    = 422
MEMTOP = 424
CRTCTRL= 476
SEGTBL = 540
        .ASECT
.=0
        NOP
        BR      BOOT1
        .WORD NXM
        .WORD 341
        .WORD 173000            ; BAD INSTRUCTION REBOOTS
        .WORD 340
        . = 34
```

```
        .WORD   READ            ;CALL DRIVER VIA TRAP
        .WORD 340
BOOT1:  MOV     @#RKDA,R0       ; GRAB DISK ADDRESS USED FOR BOOT READ
        ROL     R0
        ROL     R0
        ROL     R0
        ROL     R0
        BIC     #177770,R0      ; ZAP ALL BUT UNIT BITS
        ADD     #9.,R0
        MOV     R0,RKUNIT
        JMP     BOOT

; RK05 DISK HANDLER

RKDA    = 177412                ;RK DISK ADDRESS

RKUNIT: .WORD

READ:   MOV     #14,R3          ;PHYSICAL BLOCK TO RK DISK ADD.
        BR      2$              ;ENTER BLOCK # COMPUTATION
1$:     ADD     #20,R3          ;CONVERT DISK ADDRESS
2$:     SUB     #14,R0
        BPL     1$
        ADD     R3,R0           ;R0 HAS DISK ADDRESS
5$:     MOV     #RKDA,R3        ;POINT TO HARDWARE DISK ADDR REGISTER
        BIC     #17777,@R3      ;LEAVE THE UNIT NUMBER
        BIS     R0,(R3)         ;PUT DISK ADDRESS INTO CONTROLLER
        MOV     R2,-(R3)        ;BUFFER ADD.
        MOV     R1,-(R3)        ;WORD COUNT
        NEG     (R3)            ;(NEGATIVE)
        MOV     #5,-(R3)        ;START DISK READ
3$:     TSTB    (R3)            ;WAIT UNTIL COMPLETE
        BPL     3$
        TST     (R3)            ;ANY ERRORS?
        BMI     BIOERR          ;HARD HALT ON ERROR
        ADD     R1,R2           ; POINT R2 ABOVE LAST IO FOR THE
        ADD     R1,R2           ; BENEFIT OF LOADER (R1 IS WORDS!)
        MOV     (SP)+,@SP
        RTS     PC

BIOERR: JSR     R0,BOMB
        .ASCIZ  <CR><LF><BUZZ>'?IO ERROR WHILE BOOTING?'<CR><LF>
        .EVEN

NOCORE: JSR     R0,BOMB
        .ASCIZ  <CR><LF><BUZZ>'?NOT ENOUGH CORE TO BOOT?'<CR><LF>
        .LIST   BEX
        .EVEN

        .=400
BOOT:   MOV     #20000,SP       ;SET STACK POINTER
        MOV     #1,R0           ; BLOCK #
        MOV     #1280.,R1       ; WORD COUNT
        MOV     #1000,R2        ; BUF ADDR
        TRAP                    ; READ REST OF BOOT AND DIR!
        ; CORE DETERMINATION
        .ENABL  LSB
        CLR     R2              ;LOOK FOR TOP OF CORE
2$:     ADD     #4000,R2        ;MOVE TO NXT 1K BANK
        CMP     R2, (PC)+       ;REACHED 28K YET ?
```

```
$MEMRY: .WORD 160000    ;CHANGE HERE TO LOWER TOP O' MONITR
        BEQ     NXM             ;YES, DO A 28K SYSTEM
        TST     @R2             ;NO, SEE IF THIS LOC EXISTS
        BR      2$              ;KEEP GOING IF WE DIDN'T TRAP
        .DSABL LSB
NXM:    MOV     R2,MEMSIZ       ; STASH MEMORY SIZE FOR LATER
        CMP     R2,#100000      ; DO WE HAVE AT LEAST 16K?
        BLO     NOCORE
        MOV     #NOBOMB,@#4     ; AVOID BLOWUP FOR BAD MEM NOW
        BIS     #100,@#177546
        JSR     R0,DIRSRCH      ; FIND THE CODE FILE FOR THE SYSTEM
        .ASCIZ  <15>'SYSTEM.PASCAL?'<CR><LF>
        MOV     (R1)+,FSTSYS    ; SAVE FIRST BLOCK FOR SYSTEM CODE
        JSR     R0,DIRSRCH      ; NOW LOOK FOR THE INTERPRETER .SAV FILE
        .ASCIZ  <15>'SYSTEM.PDP-11?'<CR><LF>
        MOV     (R1)+,R0        ; BLOCK # OF INTERP
        MOV     @R1,R1          ; LAST BLOCK # IN INTERP
        SUB     R0,R1           ; NOW R1 IS # BLOCKS TO READ
        SWAB    R1              ; MAKE # WORDS FOR READ
        MOV     R1,INTSIZ       ; SAVE INTERPRETER SIZE FOR LATER
        MOV     #20000,R2       ; MEM ADDR TO READ INTERP INTO
        TRAP                    ; PERFORM DISK READ
        MOV     FSTSYS,R0       ; NOW READ SEGTBL FROM PASCAL CODE INTO INTERP
        MOV     #NSEGS*2,R1     ; # WORDS TO READ
        MOV     #DIREC,R2       ; AND THE MEMADDR
        TRAP                    ; PERFORM THE READ
        MOV     #DIREC,R0       ; SOURCE OF SEGDESC...THEN
        MOV     #20000+SEGTBL,R2 ; RELOCATE DISK ADDRS
        MOV     #NSEGS,R1       ; LOOP COUNTER
1$:     MOV     RKUNIT,(R2)+    ; PUT UNIT # INTO SEGTBL
        MOV     (R0)+,@R2       ; COPY IN REL DISK BLOCK
        ADD     FSTSYS,(R2)+    ; ADD START ADDR TO REL ADDR ALRDY THERE
        MOV     (R0)+,(R2)+
        DEC     R1
        BNE     1$
        MOV     #20004+SEGTBL,R0 ; POINT R0 AT LENG OF SEG 0
        MOV     MEMSIZ,R2       ; SET UP MEM ADDR TO READ ROUTINE
        SUB     @R0,R2          ; GET HIGH ADDR...SUBTRACT CODE LENGTH
        MOV     R2,SP           ; THIS IS STACK FOR SYSTEM ENTRY..STASH IT
        MOV     @R0,R1          ; NOW # WORDS TO READ
        ASR     R1              ; MAKE # WORDS...ASSUME < 32K BYTES
        MOV     -(R0),R0        ; FINALLY, GET DISK ADDR
        TRAP                    ; AND READ IN SYSTEM CODE
        TST     -(R2)           ; R2 WAS ABOVE HIGH MEM...NOW @ HIGH MEM WORD
        MOV     R2,20000+SEG    ; SET UP SEG STATE IN INTERP
        MOV     R2,20000+MEMTOP ; SET TOP OF MEM PTR...USED IN CXP
        MOV     RKUNIT,20000+SYUNIT     ; SET UP SYSTEM UNIT #
        SUB     -(R2),R2        ; R2 NOW POINTS @ JTAB OF OUTER BLOCK
        MOV     SP,MP           ; SET UP MP & BASE TO CBP WILL
        SUB     DATASZ(R2),MP   ; TO THEMSELVES
        MOV     #400,@MP        ; FUNNY PARAM TO SYSTEM!!!!!!!!!
        SUB     #14,MP
        MOV     MP,BASE         ; ALL REGS SET UP NOW
        CLR     -(SP)           ; SET UP FOR RTI
        MOV     20040,R0        ; GRAB INTERP ENTRY POINT
        MOV     R0,-(SP)        ; AND PUSH ON STACK FOR RTI
        MOV     #FINALE,R0
        MOV     #100000,R1      ; WHERE WE COPY FINALE CODE
        MOV     #<FINEND-FINALE>/2,BK   ; WORD COUNT OF FINALE CODE
FINLOOP:MOV     (R0)+,(R1)+
```

```
        DEC     BK
        BNE     FINLOOP
        MOV     #20000,R0
        CLR     R1
        MOV     INTSIZ,BK
        JMP     @#100000

FINALE: MOV     (R0)+,(R1)+
        DEC     BK
        BNE     FINALE
        BIS     #100,@#RCSR
        MOV     PC,IPC
        ADD     #CBP.OP-.,IPC
        NOP
        RTI
CBP.OP: .BYTE   128.+66.,1      ; CALL BASE PROCEDURE #1
FINEND = .

MEMSIZ: .WORD   ; SIZE OF MEMORY IN BYTES
INTSIZ: .WORD   ; SIZE IN WORDS OF INTERPRETER
FSTSYS: .WORD   ; FIRST DISK BLOCK OF PASCAL CODE FILE
.PAGE
DIRSRCH:MOV     #DIREC+DTID,R1  ; R1 POINTS AT DTID OF EACH DIR ENTRY
DIRLOOP:MOV     R1,R4           ; R4 IS USED FOR TITLE COMPARE
        MOV     R0,R3           ; R3 IS TITLE TO LOOK FOR (IN CODE STREAM)
        MOVB    @R0,R2          ; NUMBER OF BYTES IN NAME(STRING VAR)
1$:     CMPB    (R3)+,(R4)+     ; CHECK EACH BYTE FOR EQUAL
        BNE     2$              ; WOOPS, NEQ...CHECK NEXT ENTRY OR BOMB
        DEC     R2              ; OK SO FAR...DECREMENT LOOP COUNTER
        BPL     1$              ; LOOP FOR LENG+1 CHARS
        ; EUREKA! WE HAVE FOUND IT...RETURN WITH R1 POINTING AT ENTRY
        SUB     #DTID,R1        ; RETURN R1 AT START OF ENTRY
        ADD     #18.,R0         ; POINT R0 PAST STRING IN CODE
        RTS     R0              ; AND RETURN
2$:     ADD     #DELENG,R1      ; SKIP R1 TO NEXT DIRECTORY ENTRY
        CMP     R1,#ENDDIR      ; CHECK IF WE HAVE GONE OFF END OF DIR
        BLO     DIRLOOP         ; IF NOT, CHECK NEXT ENTRY
        .NLIST  BEX
        JSR     R0,BOMB         ; OH WELL, NO SYSTEM FILE...TIME TO CROAK
        .ASCII  <CR><LF><BUZZ>'?YOU DON'<47>'T HAVE A '<200>
        .LIST   BEX
        .EVEN
        INC     R0              ; SKIP R0 PAST LENGTH BYTE

BOMB:   MOVB    (R0)+,R1
        BMI     XBOMB           ; IF MINUS IN STRING, RETURN
        BEQ     HALTER
1$:     TSTB    @#XCSR          ; WAIT UNTIL DL11 DONE BIT
        BPL     1$
        MOVB    R1,@#XBUF
        BR      BOMB
XBOMB:  ROR     R0              ; RETURN TO USER...WORD BOUND R0
        ADC     R0
        ROL     R0
        RTS     R0

NOBOMB: MOV     (SP)+,@SP
        RTS     PC
HALTER: HALT
        BR      HALTER
```

```
 BOOTSZ  = . + 777 / 1000
 .       = BOOTSZ * 1000
 DIREC = 2000
 ENDDIR = DIREC + 4000
        .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                     F     I     N     I     S                   |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp RKBOOT
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp RX.MAC
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "rx.mac")


        .TITLE  RX-11 FLOPPY HANDLER
        ;
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                       SYSTEM TABLE CONTENTS                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=264
        RX$INT          ; RX FLOPPY INTERRUPT HANDLER
        340             ; MAX PRIORITY

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   4
        .BLKW   3
        .ENDR
        .WORD   INBIT!OUTBIT,RXSTRT,RXBOMB
        .WORD   INBIT!OUTBIT!400,RXSTRT,RXBOMB
        .PAGE
        .CSECT  RXDRVR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;               R X - 1 1   FLOPPY HANDLER                             ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RXUNIT: .WORD   0       ; ADDRESS OF UNIT TABLE ENTRY FOR I/O
RXOFST  = 14

DUMCSW: .WORD           ; START OF PHONY IO Q ENTRY FOR DRIVER
        .WORD   DUMCSW  ; POINTER TO CSW
DUMIOQ: .WORD           ; DISK BLOCK #
        .WORD           ; SPEC FUNC (0) AND UNIT # IN HIGH BYTE
        .WORD           ; BUFFER ADDRESS
        .WORD           ; WORD COUNT
```

```
        .WORD   1       ; ASYNCHRONOUS IO REQUEST

RXSTRT: ; ENTER HERE TO START FLOPPY IO'S.   WE JUST SET UP A PHONY
        ; IO Q FOR THE RT-11 DRIVER FROM TERAK AND LET IT DO ALL THE
        ; DIRTY WORK.

        TST     RXUNIT  ; ANY IO'S ALREADY GOING?
        BNE     RXSTRT  ; IF SO THEN HANG!
        MOV     R1,RXUNIT       ; NOW IO IS GOING ON RX
        BIS     #BSYBIT,@R1
        TST     (R3)+           ; SKIP PAST IO TYPE
        MOV     R3,@SP          ; SAVE ALL REGISTERS...KLUDGE!!
        MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)
        CLR     DUMCSW          ; CLEAR ANY HARD ERROR PROBS
        MOV     #DUMIOQ,R5      ; POINT R5 AT IO Q...READY TO BUILD IT
        MOV     <UBLOCK+RXOFST>(SP),(R5)+
        MOV     @R1,(R5)+       ; UNIT # (SPECIAL FORMAT IN MY UNITABLE)
        MOV     <UBUFFR+RXOFST>(SP),(R5)+
        MOV     <URLENG+RXOFST>(SP),R0
        ROR     R0              ; IN WORD COUNT (CBIT IS CLEAR)
        TST     -(R3)           ; CHECK IF READ OR WRITE
        BNE     1$              ; 1 IS A READ...0 IS A WRITE
        NEG     R0              ; NEGATIVE WORD COUNT IF A WRITE
1$:     MOV     R0,(R5)+        ; FINALLY WORD COUNT SET IN Q EL
        JSR     PC,QENTRY       ; AND START UP DRIVER
        MOV     (SP)+,R5        ; NOW RESTORE THOSE REGS
        MOV     (SP)+,R4
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOVB    DUMCSW,@R1      ; SEND POSSIBLE ERROR BACK TO USER
        JMP     @(SP)+          ; RETURN TO UIO NOW (OLD R3 VAL @SP)

$INTEN: ; FAKE $INTEN FOR DRIVERS BENEFIT
        MOV     R4,-(SP)        ; R5 ALREADY ON STACK
        JSR     PC,@R5
        MOV     (SP)+,R4
        MOV     (SP)+,R5
        JMP     @#INTRTN

RXQUIT: ; THIS IS THE DUMMY QMANAGR ENTERED UPON IO COMPLETE
        MOVB    DUMCSW,@RXUNIT  ; PLACE POSSIBLE HARD ERROR IN RESULT BYTE
        BIC     #BSYBIT,@RXUNIT
        CLR     RXUNIT
        RTS     PC              ; FALLS INTO $INTEN

RXBOMB: TST     RXUNIT
        BNE     RXBOMB
        RTS     PC
        .PAGE
        RXCS=177170
        RXDB=177172
;       CONTROL AND STATUS BITS
        CSGO=1                  ;INITIATE FUNCTION
;       FUNCTIONS (BITS 1-3)
        CSFBUF= 0*2             ;FILL SILO (PRE-WRITE)
```

```
        CSEBUF= 1*2             ;EMPTY SILO (POST-READ)
        CSWRT=  2*2             ;WRITE SECTOR
        CSRD=   3*2             ;READ SECTOR
        CSRDST= 5*2             ;READ STATUS
        CSWRTD= 6*2             ;WRITE SECTOR WITH DELETED DATA
        CSMAIN= 7*2             ;MAINTENANCE

        CSRDWR= CSRD&CSWRT      ;READ OR WRITE BIT
        CSUNIT= 20              ;UNIT BIT
        CSDONE= 40              ;DONE BIT
        CSINT=  100             ;INTERUPT ENABLE
        CSTR=   200             ;TRANSFER REQUEST
        CSINIT= 40000           ;RX11 INITIALIZE
        CSERR=  100000          ;ERROR

        DBDD=   100             ;DELETED DATA MARK
        DBIN=   4               ;RX INIT DONE INDICATOR

        RETRY=8.                ;RETRY COUNT

        SPFUNC= 100000          ;SPECIAL FUNCTIONS FLAG

RXCQE:  .WORD   DUMIOQ
;       REQUEST ENTRY POINT
QENTRY:
        CMP     -(SP),-(SP)     ;RESERVE 2 WORDS ON TO STACK TO USE
                                ;COMMON EXIT SEQUENCE
        MOV     RXCQE,R3        ;GET POINTER TO I/O ELEMENT
        MOV     (R3)+,R5        ;GET BLOCK NUMBER
        MOV     #CSGO+CSRD,R4   ;FORM A GUESS AT RXCS FUNCTION
        MOVB    (R3)+,R1        ;SAVE FUNCTION FOR LATER
        BITB    #6,@R3          ;ANY UNITS BUT 0 OR 1?
        BNE     RXERR           ;BRANCH IF YES, ERROR
        BITB    #1,(R3)+        ;CHECK FOR UNIT 0 OR 1?
        BEQ     1$              ;BRANCH IF 0
        BIS     #CSUNIT,R4      ;UPDATE GUESS AT RXCS
1$:     MOV     (R3)+,R0        ;SAVE THE BUFFER ADDRESS
        MOV     @R3,R2          ;GET WORD COUNT
        BPL     2$              ;TAKE ABSOLUTE VALUE
;       BIC     #CSEBUF,R4      ;WRITE FUNCTIONS ARE 0,2 SO CLEAR 1 BIT
        CMPB    -(R4),-(R4)     ;EQUIVALENT TO ABOVE BIC
        NEG     R2              ;MAKES IT POSITIVE
2$:     ADD     PC,R1           ;FORM PIC REFERENCE TO CHGTBL
        MOVB    CHGTBL-.(R1),R3 ;GET FUNCTION MODIFIER
        ROR     R1              ;SET C TO FLAG IF SPECIAL FUNCTION
        ROR     R3              ;SET SIGN TO SPFUNC FLAG (NON-OBVIOUS!)
        ADD     R3,R4           ;MODIFY FUNCTION BASED ON READ/WRITE
        BMI     3$              ;BRANCH IF SPECIAL FUNCTION REQUEST
        ASL     R5              ;MAKE A LOGICAL SECTOR NUMBER
        ASL     R5              ;BLOCK*4
        ASL     R2              ;MAKE WORD COUNT UNSIGNED BYTE COUNT
        BR      4$              ;SKIP SPECIAL FUNCTION INITING
3$:     CLR     (R0)+           ;CLEAR DELETED DATA FLAG WORD
        MOV     R2,PHYTRK       ;SAVE TRACK FOR LATER
        MOV     #128.,R2        ;SET THE BYTE COUNT TO 128
4$:     MOV     R0,BUFRAD       ;SAVE FOR LATER
        MOV     R5,RXLSN        ;SAVE FOR LATER
        MOV     R4,RXFUN2       ;SAVE READ OR WRITE RXCS COMMAND
        MOV     R2,BYTCNT       ;SAVE FOR LATER
        MOV     #RETRY,(PC)+    ;SET RETRY COUNT
```

```
RXTRY:  .WORD   0               ;RETRY COUNT
RXINIT: CLR     R3              ;SET THIS IS INITIAL INTERRUPT FLAG
        BR      RXWAIT          ;PERFORM A RETURN TO WAIT FOR INTERUPT

RXERR:  MOV     RXCQE,R4        ;R4 -> CURRENT QUEUE ELEMENT
        BIS     #1,@-(R4)       ;SET HARD ERROR IN CSW
RXDONE: MOV     (SP)+,R3        ;RESTORE SOME REGS
        MOV     (SP)+,R2
        CLR     @RXCSA          ;DISABLE FLOPPY INTERRUPTS
        BR      RXEXIT          ;SKIP RX INITIALIZE FUNCTION
;       RX ABORT ENTRY
        BR      RXABRT          ;ABORT OPERATION

;       INTERUPT ENTRY
RX$INT: JSR     R5,$INTEN       ;FIX MY PRIORITY
        MOV     R2,-(SP)        ;SAVE SOME REGS
        MOV     R3,-(SP)
        MOV     (PC)+,R4        ;GET ADDRESS OF RX STATUS
RXCSA:  .WORD   RXCS            ;ONLY POINTER TO I/O PAGE
        MOV     R4,R5           ;POINT R5 TO RX DATA BUFFER
        TST     (R5)+           ;CHECK FOR ERROR
        BMI     RXERR1          ;BRANCH IF ERROR

;  CODE FOR FLOPPY POWER FAIL
;       BIT     #DBIN,@R5       ;DID INITIALIZE APPEAR IN RXES?
;       BNE     RXERR1          ;BRANCH IF SO TO RETRY

        NEG     (PC)+           ;IS THIS INITIAL INTERRUPT? C=0 IF YES
INTINT: .WORD   0               ;INTERNAL INITIAL INTERUPT FLAG
        MOV     #128.,R3        ;INIT R3 FOR 128 BYTES, USED LATER
        BIT     #CSEBUF,RXFUN2  ;READ OR WRITE INTERUPT?
        BNE     2$              ;BRANCH IF READ
        BCC     1$              ;BRANCH TO AVOID UPDATING POINTERS
        JSR     PC,NXTSEC       ;SET UP FOR NEXT WRITE
1$:     JSR     R0,SILOFE       ;LOAD THE SILO
                                ;SILOFE ARG LIST
        MOVB    (R2)+,@R5       ;MOVB TO BE PLACED IN-LINE IN SILOFE
        .WORD   CSGO+CSFBUF     ;FILL BUFFER COMMAND
        CLRB    @R5             ;ZERO FILL SECTOR INSTRUCTION WHICH
                                ;WOULD BE USED FOR SHORT WRITES
;       BR      3$              ;SKIP READ FINISHING, (C BIT IS 0)
2$:     BCC     3$              ;BRANCH TO AVOID EMPTYING SILO
        TST     RXFUN2          ;IS THIS SPECIAL FUNCTION REQUEST?
        BPL     4$              ;BRANCH IF NOT SPECIAL FUNCTION CALL
        BIT     #DBDD,@R5       ;IS DELETED DATA FLAG PRESENT?
        BEQ     4$              ;BRANCH IF IT IS
        MOV     BUFRAD,R2       ;GET ADDRESS OF USER BUFFER AREA
        INC     -(R2)           ;SET FLAG WORD TO 1 INDICATING DEL DATA
4$:     JSR     R0,SILOFE       ;MOVE THE DATA INTO MEMORY FROM SILO
                                ;SILOFE ARG LIST
        MOVB    @R5,(R2)+       ;MOVB TO BE PLACED IN LINE IN SILOFE
        .WORD   CSGO+CSEBUF     ;EMPTY BUFFER COMMAND
        MOVB    @R5,R2          ;DATA SLUFFER TO BE USED FOR SHORT READ
        JSR     PC,NXTSEC       ;SET UP TO READ NEXT SECTOR
3$:     MOV     (PC)+,R3        ;GET THE LOGICAL SECTOR NUMBER
RXLSN:  .WORD   0               ;LOGICAL SECTOR NUMBER KEPT HERE
        MOV     (PC)+,R2        ;IF SPECIAL FUNCTION R3 GETS SECTOR
PHYTRK: .WORD   0               ;ABSOLUTE TRACK FOR SPECIAL FUNCS
        TST     RXFUN2          ;IS THIS SPECIAL FUNCTION?
        BMI     DOFUNC          ;BRANCH IF SPECIAL FUNCTIONS
```

```
;       FLOPPY INTERLEAVE ALGORITHM

        MOV     #8.,R2          ;LOOP COUNT
1$:     CMP     #6400,R3        ;DOES 26 GO INTO DIVIDEND?
        BHI     2$              ;BRANCH IF NOT, C CLEAR
        ADD     #171400,R3      ;SUBTRACT 26 FROM DIVIDEND, SETS C
2$:     ROL     R3              ;SHIFT DIVIDEND AND QUOTIENT
        DEC     R2              ;DEC LOOP COUNT
        BGT     1$              ;BRANCH TILL DIVIDE DONE
        MOVB    R3,R2           ;COPY TRACK NUMBER
        CLRB    R3              ;REMOVE TRACK NUMBER FROM REMAINDER
        SWAB    R3              ;GET REMAINDER
        CMP     #12.,R3         ;C=1 IF 13<=R3<=25, ELSE C=0
        ROL     R3              ;DOUBLE FOR 2 TO 1 INTERLEAVE
                                ;C-BIT COMES IN FOR SECTOR GROUP
        ASL     R2              ;ADD TRACK TO TRACK SKEW TO SECTOR
        ADD     R2,R3           ;SKEW BY 2* TRACK
        ADD     R2,R3           ;SKEW BY 4* TRACK
        ADD     R2,R3           ;SKEW BY 6* TRACK
        ASR     R2              ;REFIX TRACK NUMBER
        INC     R2              ;PUT TRACK # IN RANGE 1-76 TO HANDLE
                                ;ANSI FLOPPY, TRACK 0 IS LEFT ALONE
3$:     SUB     #26.,R3         ;MODULO SECTOR INTO RANGE -26,-1
        BGE     3$              ;LOOP TILL REMAINDER GOES NEGATIVE
        ADD     #27.,R3         ;PUT SECTOR IN RANGE 1,26
DOFUNC: MOV     (PC)+,@R4       ;SET THE FUNCTION
RXFUN2: .WORD   0               ;READ OR WRITE COMMAND ON CORRECT UNIT
1$:     TSTB    @R4             ;WAIT FOR TRANSFER READY
        BEQ     1$              ;WAIT
        BPL     RXERR1          ;TR IS NOT UP, THATS AN ERROR
        MOV     R3,@R5          ;SET SECTOR FOR FLOPPY
2$:     TSTB    @R4             ;WAIT FOR TRANSFER READY
        BEQ     2$              ;WAIT
        BMI     SECOK           ;BRANCH IF TR UP, ELSE ERROR

RXERR1: DEC     RXTRY           ;SHOULD WE TRY AGAIN?
        BLT     RXERR           ;BRANCH IF NO, RETRY COUNT EXPIRED
        MOV     #CSINIT,@R4     ;START A RECALIBRATE
        BR      RXINIT          ;EXIT THROUGH START OPERATION CODE

SECOK:  MOV     R2,@R5          ;SET TRACK FOR FLOPPY
RXWAIT: MOV     R3,INTINT       ;INTINT >0 FOR PROCESS INTERRUPT ENTRY
        MOV     (SP)+,R3        ;RESTORE THE REGS
        MOV     (SP)+,R2
        BIS     #CSINT,@RXCSA   ;ENABLE FLOPPY INTERRUPTS, THIS SHOULD
                                ;CAUSE AN INTERRUPT WHEN DONE IS UP
RETURN: RTS     PC              ;RETURN, WE'LL BE BACK WITH AN INTERUPT

NXTSEC: ADD     R3,BUFRAD       ;UPDATE BUFFER ADDRESS
        INC     RXLSN           ;BUMP LOGICAL SECTOR
        SUB     R3,BYTCNT       ;REDUCE THE AMOUNT LEFT TO TRANSFER
        BHI     RETURN          ;BRANCH IF WE ARE NOT DONE
        CLR     BYTCNT          ;INIT FOR POSSIBLE SHORT WRITE
        BIT     #CSEBUF+SPFUNC,RXFUN2   ;IS THIS A READ OR SPECIAL
                                ;FUNCTION OPERATION?
        BNE     1$              ;BRANCH IF EITHER, FOR .WRITE WE HAVE
                                ;TO 0 TO THE END OF A BLOCK
        BIT     #3,RXLSN        ;ARE WE AT 1ST SECTOR IN BLOCK?
        BNE     RETURN          ;BRANCH IF NOT TO CONTINUE
```

```
1$:     TST     (SP)+           ;POP RETURN ADDRESS FROM STACK
        BR      RXDONE          ;REQUEST DONE
RXABRT: MOV     #CSINIT,@RXCSA  ;PERFORM AN RX11 INITIALIZE
RXEXIT: JMP     RXQUIT

;  R3 IS 128 ON ENTRY
;  THIS ROUTINE ASSUMES ERROR CAN NOT COME UP DURING A FILL OR EMPTY!!

SILOFE: MOV     (R0)+,EFBUF     ;PUT CORRECT MOV INSTRUCTION IN FOR
                                ;EITHER FILLING OR EMPTYING RX BUFFER
        MOV     (R0)+,@R4       ;INITIATE FILL OR EMPTY BUFFER COMMAND
        MOV     (PC)+,-(SP)     ;ASSUME MAXIMUM OF BYTCNT TO MOVE FROM
                                ;BUFFER
BYTCNT: .WORD   0               ;THE BYTE COUNT IS KEPT HERE
        BEQ     ZFILL           ;BRANCH IF SEEK OR SHORT WRITE
                                ;NOTE SEEK DOES THE EMPTY (TIME WASTER)
        CMP     @SP,R3          ;NOW MAKE SURE COUNT IS 128 OR LESS
        BLOS    1$              ;BRANCH IF @SP IS 128 OR LESS
        MOV     R3,@SP          ;RESET COUNT TO 128
1$:     MOV     (PC)+,R2        ;PUT THE BUFFER ADDRESS IN R2
BUFRAD: .WORD   0               ;THE BUFFER ADDRESS IS KEPT HERE
TRBYT:  TSTB    @R4             ;WAIT FOR TRANSFER READY
        BPL     TRBYT           ;BRANCH IF TR NOT UP
EFBUF:  HALT                    ;INSTRUCTION TO MOV OR SLUFF DATA FROM
                                ;BUFFER GETS PLACED HERE
                                ;  MOVE DATA        SLUFF DATA
                                ;MOVB (R2)+,@R5     CLRB @R5       FILL
                                ;MOVB @R5,(R2)+     MOVB @R5,R2   EMPTY
        DEC     @SP             ;CHECK FOR COUNT DONE
        BGT     TRBYT           ;STILL MORE TO TRANSFER
ZFILL:  MOV     @R0,EFBUF       ;CHANGE MOV INSTRUCTION TO CORRECT
                                ;INSTR FOR SLUFFING DATA TO/FROM BUFFER
1$:     TSTB    @R4             ;WAIT LOOP
        BEQ     1$              ;WAIT FOR SOME INDICATION (TR,DONE)
        BMI     EFBUF           ;BRANCH IF TR CAME UP TO SLUFF DATA
        BIT     (R0)+,(SP)+     ;BUMP R0 TO RETURN ADDR AND REMOVE JUNK
                                ;WORD FROM STACK LEAVING C BIT=0
        RTS     R0              ;RETURN


        .BYTE   6*2             ;READ+GO -> WRITE DELETED+GO
        .BYTE   -2*2            ;READ+GO -> WRITE+GO
        .BYTE   0*2             ;READ+GO -> READ+GO
CHGTBL: .BYTE   0*2             ;READ/WRITE STAY THE SAME

        .EVEN

$INPTR: .WORD   $INTEN          ;INTERUPT ENTRY COMMMUNICATIONS WORD

        SYSIZE=.-RXSTRT         ;TOTAL SIZE OF HANDLER

        .END


; +-----------------------------------------------------------------+
; |                                                                 |
; |                    F     I     N     I     S                    |
; |                                                                 |
; +-----------------------------------------------------------------+
```

```
### END OF FILE UCSD Pascal 1.5 Interp RX.MAC
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Interp RXBOOT
########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "rxboot.mac")


        .NLIST  TTM
        .TITLE  FLOPPY BOOTSTRAP
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;    PDP-11 PASCAL SYSTEM BOOTSTRAP LOADER
;  THIS CODE SITS ON BLOCKS 0 & 1 OF THE
;  FLOPPY AND IS READ INTO LOW CORE BY THE
;  HARDWARE LOADER.  CONTROL THEN PASSES TO
;  ADDRESS 0 WHERE WE TAKE OVER.  NOTE THAT
;  R0 = DRIVE # WE WERE LOADED FROM (0 OR 1)
;  ALSO ONLY THE FIRST 128 BYTES ARE LOADED BY
;  HARDWARE.  THE REST HAS TO BE DONE OURSELVES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RELOC   = 32000         ; ADDRESS WHERE THE CODE COPIES ITSELF TO
                        ; AND ACTUALLY RUNS...HENCE THE ASECT GROSSNESS
BK = %2
BASE = %3
IPC= %4
MP = %5
R0 = %0
R1 = %1
R2 = %2
R3 = %3
R4 = %4
R5 = %5
SP = %6
PC = %7
DATASZ  = -10   ; INDEX TO LOCAL DATA SIZE IN JTAB
SYSUNT  = 404   ; ADDRESS OF SYSUNT WORD IN SYSCOM
SEG     = 422   ; SEG ADDR IN SYSCOM
SEGTBL  = 540
MEMTOP = 424
FSTBLK  = 2     ; DIR ENTRY INDEX FOR DFIRSTBLK
DTID    = 6     ;  "    "     "  FOR TITLE FIELD
DELENG  = 26.   ; LENGTH IN BYTES OF EACH DIR ENTRY
KW11    = 177546
RXCS    = 177170
RCSR    = 177560
```

```
RBUF    = 177562
XCSR    = 177564
XBUF    = 177566
CR = 15
LF = 12
BUZZ = 07

        .ASECT
. = RELOC

;  THESE FIRST 128 BYTES ARE REALLY LOADED AT WORD 0
;  AND MUST READ IN THE REST OF THE SYSTEM
;  WHAT IS DONE HERE IS TO READ THE WHOLE BOOTSTRAP
;  INTO THE RELOCATE POSITION FOR ACTUAL EXECUTION.

ENTRY:  NOP                     ; BOOT LOADER WANTS THIS HERE
        MOV     #1000,SP        ; SET UP STACK...NO 4-TRAP HANDLING YET
        MOV     R0,-(SP)        ; SAVE R0 FOR LATER CHECK AFTER START
        BEQ     2$              ; SKIP BIS...NO 10-TRAP EITHER
        BR      1$              ; ELSE SET PROPER UNIT #
        .WORD   TRWAIT-RELOC    ; WAIT ON TR BIT FOR RX (BPT OP)
        .WORD   340
        .WORD   DNWAIT-RELOC    ; WAIT ON DONE BIT (IOT OP)
        .WORD   340
1$:     BIS     #20,RXMASK      ; UNIT WORD, SO SET UNIT 1 BIT IF R0 <> 0
2$:     MOV     #RELOC,R0       ; BUFFER ADDRESS FOR RXREAD
        MOV     #8.,R1          ; NUMBER OF SECTORS TO READ
        CLR     R2              ; LOGICAL SECTOR NUMBER
        MOV     #START,-(SP)    ; TRICKY JSR TO SAVE A WORD (HARD UP)
RXREAD: ; DUMB IO ROUTINE FOR SYSTEM LOADER
        MOV     R2,-(SP)        ; SAVE LSN FOR LATER
RXLOOP: MOV     #RXCS,R4        ; R4 POINTS AT RXCS
        IOT     ; WAIT UNTIL DONE
        MOV     R4,R5           ; SET UP TO POINT R5 AT RXCS+2...RXDB
        MOV     RXMASK,(R5)+    ; SEND READ SECTOR COMMAND
        MOV     (SP),R3         ; SET UP FOR SECTOR AND TRACK CALC
        JSR     PC,@RXCALC      ; HERE IT IS...THE GROSSEST THING EVER
        BPT     ; WAIT UNTIL TR
        MOV     R3,(R5)         ; SEND SECTOR ADDRESS
        BPT     ; WAIT UNTIL TR
        MOV     R2,(R5)         ; SEND TRACK ADDRESS
        IOT     ; WAIT UNTIL DONE
        MOV     #128.,R3        ; LOOP COUNTER FOR MOVB
        MOV     (PC),(R4)       ; SEND EMPTY BUFFER COMMAND
1$:     BPT     ; WAIT UNTIL TR (BPT IS 03, EMPTY BUFF COMMAND, YUK)
        MOVB    (R5),(R0)+
        DEC     R3              ; DEC LOOP COUNTER...128 MOVB DONE
        BNE     1$              ; IF = 0
        INC     (SP)            ; BUMP LOGICAL SECTOR #
        DEC     R1              ; NUMBER OF SECTORS TO READ
        BNE     RXLOOP
        MOV     (SP)+,R2        ; RESTORE UPDATED SECTOR ADDRESS
        RTS     PC              ; SO R0 (ADDRESS) AND R2 (SECTOR) ARE UPDATED

RXMASK: .WORD   7               ; READ SECTOR COMMAND
RXCALC: .WORD   RXDUMB-RELOC    ; ADDR OF TRACK SECT CALC ROUTINE

RXDUMB: ; THIS ROUTINE FIGURES AN ACTUAL SECTOR FROM LSN IN R3
        ; IT IS USED ONLY TO LOAD REMAINDER OF BOOTSTRAP. AFTER
        ; THAT RXCALC IS MADE INTELLIGENT AND WE GO INTO THE OZONE
```

```
        MOV     #1,R2           ; RETURN TRACK #1 (FLOPPY CONVENTION)
        ASL     R3              ; DOUBLE IT FOR 2 INTERLEAVE
        INC     R3              ; AND STAGGER BY 1 (FLOPPY CONVENTION)
        RTS     PC              ; BACK TO RXREAD

TRWAIT: TSTB    (R4)            ; CHECK TR BIT
        BEQ     TRWAIT          ; AS LONG AS ZERO, NOT DONE OR TR
        BPL     HALTER          ; DONE BIT RATHER THAN TR...BOMBOUT
DORTI:  MOV     (SP)+,@SP
        RTS     PC


DNWAIT: TST     (R4)            ; WAIT UNTIL DONE BIT (NOT TR)
        BEQ     DNWAIT          ; SO LOOP AS LONG AS RXCS = 0
        BPL     DORTI           ; DO AN RTI BACK TO RXREAD
HALTER: HALT
        BR      HALTER

;  PAST THIS POINT IS THE CODE LOADED BY THE FIRST 128
;  BYTES.  THIS INCLUDES MISC ROUTINES AND SYSTEM STARTUP

RXSMRT: MOV     #8.,R2          ; LOOP COUNT
1$:     CMP     #6400,R3        ; DOES 26 GO INTO DIVIDEND?
        BHI     2$              ; BRANCH IF NOT, C CLEAR
        ADD     #171400,R3      ; SUBTRACT 26 FROM DIVIDEND, SETS C
2$:     ROL     R3              ; SHIFT DIVIDEND AND QUOTIENT
        DEC     R2              ; DEC LOOP COUNT
        BGT     1$              ; BRANCH TILL DIVIDE DONE
        MOVB    R3,R2           ; COPY TRACK NUMBER
        CLRB    R3              ; REMOVE TRACK NUMBER FROM REMAINDER
        SWAB    R3              ; GET REMAINDER
        CMP     #12.,R3         ; C=1 IF 13<=R3<=25, ELSE C=0
        ROL     R3              ; DOUBLE FOR 2 TO 1 INTERLEAVE
                                ; C-BIT COMES IN FOR SECTOR GROUP
        ASL     R2              ; ADD TRACK TO TRACK SKEW TO SECTOR
        ADD     R2,R3           ; SKEW BY 2* TRACK
        ADD     R2,R3           ; SKEW BY 4* TRACK
        ADD     R2,R3           ; SKEW BY 6* TRACK
        ASR     R2              ; REFIX TRACK NUMBER
        INC     R2              ; PUT TRACK # IN RANGE 1-76 TO HANDLE
                                ; ANSI FLOPPY, TRACK 0 IS LEFT ALONE
3$:     SUB     #26.,R3         ; MODULO SECTOR INTO RANGE -26,-1
        BGE     3$              ; LOOP TILL REMAINDER GOES NEGATIVE
        ADD     #27.,R3         ; PUT SECTOR IN RANGE 1,26
        RTS     PC              ; AND RETURN TO RXREAD

BOMB:   MOVB    (R0)+,R1
        BMI     XBOMB           ; IF MINUS IN STRING, RETURN
        BEQ     HALTER
1$:     TSTB    @#XCSR          ; WAIT UNTIL DL11 DONE BIT
        BPL     1$
        MOVB    R1,@#XBUF
        BR      BOMB
XBOMB:  ROR     R0              ; RETURN TO USER...WORD BOUND R0
        ADC     R0
        ROL     R0
        RTS     R0

NOCORE: JSR     R0,BOMB
        .ASCIZ  <CR><LF><BUZZ>'?NOT ENOUGH CORE TO BOOT?'<CR><LF>
```

```
        .EVEN

DIRSRCH:MOV     #DIREC+DTID,R1  ; R1 POINTS AT DTID OF EACH DIR ENTRY
DIRLOOP:MOV     R1,R4           ; R4 IS USED FOR TITLE COMPARE
        MOV     R0,R3           ; R3 IS TITLE TO LOOK FOR (IN CODE STREAM)
        MOVB    @R0,R2          ; NUMBER OF BYTES IN NAME (STRING VAR)
1$:     CMPB    (R3)+,(R4)+     ; CHECK EACH BYTE FOR EQUAL
        BNE     2$              ; WOOPS, NEQ...CHECK NEXT ENTRY OR BOMB
        DEC     R2              ; OK SO FAR...DECREMENT LOOP COUNTER
        BPL     1$              ; LOOP FOR LENG+1 CHARS
        ; EUREKA! WE HAVE FOUND IT...RETURN WITH R1 POINTING AT ENTRY
        SUB     #DTID,R1        ; RETURN R1 AT START OF ENTRY
        ADD     #18.,R0         ; POINT R0 PAST STRING IN CODE
        RTS     R0              ; AND RETURN
2$:     ADD     #DELENG,R1      ; SKIP R1 TO NEXT DIRECTORY ENTRY
        CMP     R1,#ENDDIR      ; CHECK IF WE HAVE GONE OFF END OF DIR
        BLO     DIRLOOP         ; IF NOT, CHECK NEXT ENTRY
        JSR     R0,BOMB         ; OH WELL, NO SYSTEM FILE...TIME TO CROAK
        .ASCII  <CR><LF><BUZZ>'?YOU DON'<47>'T HAVE A '<200>
        .EVEN
        INC     R0              ; SKIP R0 PAST LENGTH BYTE
        BR      BOMB            ; USE R0 AS IS FOR REST OF MESSAGE

START:  TST     (SP)+           ; GRAB OLD UNIT # FROM ENTRY
        BEQ     1$              ; SO SKIP OVER UNIT 1 STUFF
        BIS     #20,RXMASK      ; SET THE UNIT 1 BIT IN IO MASK
1$:     MOV     #DIREC,SP       ; OUR NEW STACK LOCATION
        MOV     #NOCORE,@#4     ; BOMB SYSTEM IF NO CORE
        MOV     #RXSMRT,RXCALC  ; NOW WE CAN USE THE DISK OK
        MOV     #TRWAIT,@#14    ; RELOCATE THIS STUFF
        MOV     #DNWAIT,@#20    ; PROPERLY RELOCATED
        MOV     #16.,R1         ; READ DIRECTORY...NUMBER OF SECTORS
        JSR     PC,RXREAD       ; R0 AND R2 ARE STILL OK FROM LAST TIME
        JSR     R0,DIRSRCH      ; FIND THE CODE FILE FOR THE SYSTEM
        .ASCIZ  <15>'SYSTEM.PASCAL?'<CR><LF>
        MOV     (R1)+,FSTSYS    ; SAVE FIRST BLOCK FOR SYSTEM CODE
        JSR     R0,DIRSRCH      ; NOW LOOK FOR THE INTERPRETER .SAV FILE
        .ASCIZ  <15>'SYSTEM.PDP-11?'<CR><LF>
        MOV     (R1)+,R2        ; FIRST DISK ADDR OF INTERPRETER
        ASL     R2              ; MULTIPLY BY FOUR TO GET LSN
        ASL     R2
        MOV     @R1,R1          ; GET LAST BLOCK # OF INTERP
        INC     R1              ; MAKE IT POINT AT ALL BLOCKS
        ASL     R1              ; MULTIPLY BY 4
        ASL     R1
        SUB     R2,R1           ; NOW R1 IS TOTAL SECTORS IN INTERP
        MOV     R1,-(SP)        ; SAVE THIS SECTOR COUNT FOR LATER
        MOV     #DIREC,R0       ; READ FIRST SECTOR INTO HIGH CORE
        MOV     #1,R1           ; TO SAVE TRAP VECTOR
        JSR     PC,RXREAD       ; NOW READ IN FIRST SECTOR
        MOV     #200,R0         ; NEXT READ IN REMAINDER OF INTERP
        MOV     (SP)+,R1        ; AGAIN TOTAL LENGTH IN SECTORS
        DEC     R1              ; BUT SUBTRACT THE ONE WE HAVE

        JSR     PC,RXREAD
        MOV     #TSTTRP,@#4     ; NOW FIND HIGHEST CORE FOR SP
TSTLOOP:TST     (R0)+           ; START SKIPPING THROUGH MEMORY
        BR      TSTLOOP         ; UNTIL 4 TRAP TAKES US AWAY
BADSYS: JSR     R0,BOMB         ; GO HERE TO CROAK FOR BAD CODE FILE
        .ASCIZ  <CR><LF><BUZZ>'?YOUR SEGTBL IS RIDICULOUS?'<CR><LF>
```

```
        .EVEN
TSTTRP: SUB     #4,R0           ; BRING R0 BACK DOWN TO EARTH
        MOV     R0,SEG          ; HIGHEST MEM ADDR
        ; NOW READ SYSTEM CODE STUFF
        MOV     #LSEGTBL,R0     ; MEM ADDR TO RXREAD INTO
        MOV     #1,R1           ; NUMBER OF 128 SECTOR TO READ
        MOV     FSTSYS,R2       ; BLOCK NUMBER OF SYSCODE
        ASL     R2              ; MULTIPLY BY 4
        ASL     R2              ; TO END UP WITH RX LOGICAL SECTOR #
        JSR     PC,RXREAD       ; DO IO...READ IN SEGTBL FROM SYSCODE
        MOV     #4,SYSUNT       ; INFORM SYSTEM WHICH UNIT WAS LOADED FROM
        CMP     RXMASK,#7       ; NOW CHECK IF UNIT WAS RIGHT HAND DRIVE
        BEQ     3$              ; SKIP IF OUR IO WORD IS FOR LEFT UNIT
        MOV     #5,SYSUNT       ; IF RIGHT DRIVE, CHANGE UNIT TO # 5
3$:     MOV     #LSEGTBL,R0     ; NOW COPY LOCAL TBL INTO SYSTEM TABLE
        MOV     #SEGTBL,R1      ; AND RELOCATE DISK ADDRS
        MOV     #16.,R2         ; FOR ALL 16 SEGS...LOOP COUNT
TBLCPY: MOV     SYSUNT,(R1)+    ; COPY OVER BOOTED UNIT
        MOV     FSTSYS,@R1      ; PUT BASE FILE ADDR OF SYSTEM INTO TABLE
        ADD     (R0)+,(R1)+     ; ADD IN RELATIVE...NOW ABSOLUTE DK ADDR
        MOV     (R0)+,(R1)+     ; COPY BYTE COUNT TOO
        DEC     R2              ; DONE LOOP 16 TIMES?
        BNE     TBLCPY          ; GO LOOP IF NOT
        ; NOW READ IN SEG #0 OF SYSTEM CODE INTO HIGH CORE
        MOV     SEG,R0          ; GET HIGHEST MEM ADDR AGAIN!
        SUB     SEGTBL+4,R0     ; OPEN GAP @R0 BIG ENOUGH FOR SYSCODE
        TST     (R0)+           ; SEG POINTS @ LAST WORD...NOT ABOVE
        MOV     R0,NEWSP        ; STASH THE NEW STACK TOP
        MOV     #77777,R1       ; NUMBER OF SECTORS TO READ
        MOV     SEGTBL+2,R2     ; FINALLY GET DISK BLOCK AND
        ASL     R2              ; MAKE A LSN
        ASL     R2
        MOV     #AHEAD,@#4      ; TRAP TO STOP DISK READ
        JSR     PC,RXREAD       ; READ IN SYSTEM CODE FILE
AHEAD:  BIT     #40,@#RXCS      ; IS DONE BIT ON?
        BNE     1$
        TSTB    @#RXCS+2        ; GRAB THE LEFT OVER BYTE
        BR      AHEAD
1$:     MOV     NEWSP,SP        ; NEW STACK POINTER JUST BELOW SYSCODE
        MOV     #64.,R2         ; NOW FINAL INITIALIZE...COPY VECTOR STUFF
        CLR     R1              ; COPY 128 BYTES FROM HIGH CORE TO LOW
        MOV     #DIREC,R0
2$:     MOV     (R0)+,(R1)+     ; COPY EACH WORD OF FIRST SECTOR
        DEC     R2
        BNE     2$
        MOV     @#4,-(SP)       ; SAVE 4-TRAP THING FOR KW11 TEST
        MOV     #NOKW11,@#4     ; SET UP TO ENABLE INTERRUPTS ON CLOCK
        BIS     #100,@#KW11     ; ENABLE INTERRUPT...IF NO CLOCK THEN TRAP
        SUB     #4,SP           ; KW11 RUNNING...MAKE STACK LOOK LIKE TRAPPED
NOKW11: ADD     #4,SP           ; CHUCK GARBAGE ON STACK
        MOV     (SP)+,@#4       ; RESTORE INTERP 4 TRAP HANDLER
        ; NOW SET UP REGISTERS FOR ENTRY VIA CBP 1 .  WE WANT OUTER
        ; BLOCK OF SYSTEM TO HAVE STAT&DYN LINKS POINT TO ITSELF.
        ; THIS ENABLES TESTING FOR STACK UNDERFLOWS.
        MOV     SEG,R0          ; POINT R0 AT JTAB FOR OUTER BLOCK
        MOV     R0,MEMTOP
        SUB     -(R0),R0        ; SUBTRACT SELF RELATIVE
        MOV     SP,MP           ; SET MP TO WHERE IT WILL BE AFTER CBP
        SUB     DATASZ(R0),MP   ; OPEN GAP FOR UNIVERSAL LEVEL VARS
        MOV     #400,@MP        ; FUNNY PARAM FOR SYSTEM OUTER BLOCK
```

```
         SUB     #14,MP          ; LEAVE MP AT ITS FINAL LOCATION
         MOV     MP,BASE         ; BASE SHOULD POINT HERE TOO.
         MOV     #CBP.1,IPC      ; SET UP IPC FOR CBP CALL OF OUTER BLOCK
         BIS     #100,@#RCSR     ; ENABLE KEYBOARD INTERRUPTS
         MOV     @#40,BK         ; WORD 40 HAS BACK ADDRESS
         CLR     -(SP)           ; SO PR = 0 UPON ENTRY
         MOV     BK,-(SP)        ; PC FOR RTI...ENTR SYSTEM AT BACK
         RTI                     ; NOW START...FETCH CBP 1 OPCODE

 FSTSYS: .WORD   ; DISK ADDR (BLOCK) OF PASCAL SYSTEM CODE
 NEWSP:  .WORD   ; STACK POINTER WE COMPUTE
 CBP.1:  .BYTE   128.+66.,1      ; OUTER BLOCK CALLING SEQUENCE

 DIREC   = RELOC+1024.
 ENDDIR  = DIREC+2048.   ; ROOM FOR 4 BLOCK DIRECTORY
 LSEGTBL = DIREC+128.    ; DONT DESTROY LOW CORE IMAGE

         .END    ENTRY

; +-------------------------------------------------------------------+
; |                                                                   |
; |                     F     I     N     I     S                     |
; |                                                                   |
; +-------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp RXBOOT
```

```
##############################################################################
### FILE: UCSD Pascal 1.5 Interp TERAK
##############################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "terak.mac")


 TERAK=1


; +--------------------------------------------------------------------+
; |                                                                    |
; |                     F     I     N     I     S                      |
; |                                                                    |
; +--------------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp TERAK
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 Interp TK.MAC
##########################################################################


; UCSD PASCAL I.5 INTERPRETER (FILE "tk.mac")


        .TITLE  TERAK SCREEN HANDLER
        ;
        ; COPYRIGHT (C) 1978 REGENTS OF THE UNIVERSTIY OF CALIFORNIA.
        ; PERMISSION TO COPY OR DISTRIBUTE THIS SOFTWARE OR DOCUMEN-
        ; TATION IN HARD COPY OR SOFT COPY GRANTED ONLY BY WRITTEN LICENSE
        ; OBTAINED FROM THE INSTITUTE OF INFORMATION SYSTEMS.  ALL RIGHTS
        ; RESERVED.  NO PART OF THIS PUBLICATION MAY BE REPRODUCED, STORED
        ; IN A RETRIEVAL SYSTEM ( E.G., IN MEMORY, DISK, OR CORE) OR BE
        ; TRANSMITTED BY ANY MEANS, ELECTRONIC, MECHANICAL, PHOTOCOPY,
        ; RECORDING, OR OTHERWISE, WITHOUT PRIOR WRITTEN PERMISSION FROM THE
        ; PUBLISHER.
        ;
        ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                       SYSTEM TABLE CONTENTS                                  ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .ASECT          ; INTERRUPT HANDLER LOCATION IN VECTORS
.=100
        KW$INT          ; KW-11 (MAYBE REFRESH HARDWARE!) CLOCK HANDLER
        300             ; PR-6
.=164
        EM$INT          ; DATA MEDIA EMULATOR TRAP
        341             ; PR-7...SEE TERAK DOCUMENTATION
                        ; C-BIT FLAGS FIRST CALL FROM CONSOLE DEV
.=174
        EM$INT          ; SAME THING...ALTERNATE IV
                        ; NO C-BIT...LEAVE EMPTR AS AUXILIARY
.=472
        .WORD   ^B0001111
        .WORD   0
        .BYTE   0,EM,VT,GS,FS,US,BS,0,0,14
.=512
        .WORD   24.,80.
        .BYTE   032,014,027,013
.=526
        .BYTE   BS
.=532
        .BYTE   0,03,010

        .CSECT  TABLES
        .BLKW   128.    ; OPERATOR XFER TABLE
        .REPT   3
        .BLKW   3
        .ENDR
        .WORD   OUTBIT,KWSTRT,KWABRT
        .PAGE
        .CSECT  TKDRVR
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;               KW-11 IO HANDLER (GRAPHICS REFRESH)                             ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

KWUNIT: .WORD   0
BELREG: .WORD   0       ; SHIFT REG...<> 0 IF BELL IS ON
VCRVAL: .WORD   0       ; VIDEO CONTROL REG...SET IN KWDRVR
GARVAL: .WORD   20000   ; REFRESH ADDRESS FOR GRAPHICS

GAR = 177740    ; GRAPHICS ADDRESS REGISTER
VIR = 177742    ; VIDEO INDEX...SCAN LINE, BUFFLINE
VCR = 177744    ; VIDEO CONTROL REGISTER

KW$INT: CMP     QHEAD,QTAIL     ; ANY CHARS IN EMULATOR QUEUE?
        BEQ     1$              ; SKIP EMULATOR CALL IF NONE
        JSR     PC,EMULAT       ; DATAMEDIA EMULATOR
1$:     ADD     #1,@#LOTIME     ; BUMP LINE CLOCK COUNTER
        ADC     @#HITIME
        ASR     BELREG          ; UPDATE BELL COUNTER
        BNE     2$              ; IF BELL STILL ON THEN SKIP TURNOFF
        BCC     2$              ; IF BELL WAS OFF THEN SKIP TOO
        BIC     #4000,@#VCR     ; ELSE WAS ON...ZAP 780 HZ TONE BIT IN VCR
2$:     TST     KWUNIT
        BEQ     3$
        MOV     GARVAL,@#GAR
        MOV     VCRVAL,@#VCR
        BIC     #BSYBIT,@KWUNIT
        CLR     KWUNIT
3$:     JMP     @#INTRTN

KWSTRT: TST     KWUNIT
        BNE     KWSTRT
        MTPS    #300
        MOV     R1,KWUNIT
        BIS     #BSYBIT,@R1
        TST     (SP)+
        TST     (R3)+
        MOV     UBUFFR(SP),GARVAL
        MOV     URLENG(SP),VCRVAL
        MTPS    #0
        JMP     @R3

KWABRT: MTPS    #300
        TST     KWUNIT
        BEQ     1$
        BIC     #BSYBIT,@KWUNIT
        CLR     KWUNIT
1$:     MOV     #7,VCRVAL
        MOV     #7,@#VCR
        MTPS    #0
        RTS     PC
        .PAGE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;               CRT EMULATOR INPUT-OUTPUT HANDLER                               ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
EM$BUF: .WORD   177766  ; EMULATOR DATA BUF ADDRESS

EM$INT: ; THIS IS THE INTERRUPT HANDLER FOR THE TERAK DATA MEDIA
        ; EMULATOR.  WE TAKE CHARS FROM THE OTHER SIDE OF THE "DL"
        ; DEVICE AND QUEUE THEM FOR OUTPUT INSIDE OF EMULAT
        BCC     1$              ; CARRY SET ON FIRST CALL FROM CONSOLE
        MOV     #177566,EM$BUF  ; RELOCATE EMULATOR ADDR
        DEC     @#166           ; AND CLEAR CARRY FOR NXT INT
1$:     MOVB    @EM$BUF,@QTAIL  ; GRAB CHAR FROM OTHER SIDE OF DL...QU IT
        INC     QTAIL           ; BUMP QUEUE POINTER
        CMP     QTAIL,#QSTART+QSIZE     ; QUEUE WRAPAROUND?
        BNE     2$
        MOV     #QSTART,QTAIL   ; IF SO THEN POINT AT Q START
2$:     RTT

QSIZE = 100     ; ALLOW 64 BYTES IN QUEUE
QSTART: .BLKB   QSIZE
QHEAD:  .WORD   QSTART  ; CHARS TAKEN FROM HERE IN EMULAT (CALLED IN KW$INT)
QTAIL:  .WORD   QSTART  ; BYTES PUT IN HERE IN EM$INT

HOMADR: .WORD   160000  ; CURRENT SCREEN HOME CHAR IN CHAR BUF
CURADR: .WORD   160000  ;    "       "   CURSOR LOCATION IN CHAR BUF
CURCHR: .WORD   BLANK   ; CHAR UNDER CURSOR

.MACRO  DEQUR1  ?L      ; REMOVES CHARS FROM QUEUE (JUST A CONVENIENCE)
        MOVB    (R1)+,R0
        CMP     R1,#QSTART+QSIZE
        BNE     L
        MOV     #QSTART,R1
L:
.ENDM   DEQUR1

EMULAT: ; THIS IS THE ACTUAL SCREEN EMULATOR...WE GRAB CHARS FROM
        ; THE CIRCULAR QUEUE QSTART AND DO WHAT IS NEEDED FOR THOSE
        ; CHARS.  R1 ALWAYS POINTS AT THE NEXT CHAR TO FETCH (STARTING
        ; AT QHEAD).  R2 IS CURADR IN THE UPDATED STATE.  THE
        ; CORE VERIONS OF THESE VALUES ARE SET UPON EMEXIT.
        MOV     R0,-(SP)        ; STASH REGS FOR TEMP USAGE
        MOV     R1,-(SP)        ; R1 SET FROM QHEAD...GRAB CHARS FROM HERE
        MOV     R2,-(SP)        ; R2 IS CURSOR ADDRESS (NEWEST VERSION)
        MOV     @#VCR,-(SP)     ; EVEN SAVE THIS SO WE CAN TURN
        BIC     #7,@#VCR        ; OFF CHAR MODE TO PREVENT POPCORN
        MOV     CURADR,R2       ; R2 IS ALWAYS NEW CURSOR ADDRESS
        MOVB    CURCHR,@R2      ; RESTORE CHAR UNDER CURSOR
        MOV     QHEAD,R1        ; ALL POINTERS SET NOW BEGIN
EMLOOP: DEQUR1                  ; GRAB CHAR FROM CIRCULAR QUEUE INTO R0
        CMPB    R0,#BLANK       ; CHECK IF IN CONTROL RANGE
        BHIS    1$              ; IF IT IS THEN USING CHAR,
        ASL     R0              ; DO CASE JUMP INTO CONTROL TABLE
        JMP     @CTLTBL(R0)     ; FOR EACH SPECIAL CHARACTER
1$:     MOVB    R0,(R2)+        ; ELSE MOVE CHAR ONTO SCREEN
$NUL:
$BAD:
EMNEXT: CMP     R1,QTAIL        ; HAVE WE EATEN UP CHARS IN QUEUE?
        BNE     EMLOOP          ; IF NOT THEN GO FOR MORE
EMEXIT: MOV     R1,QHEAD        ; ELSE RETURN... UPDATE PERM POINTERS
        MOV     R2,CURADR       ; FOR NEXT INTERRUPT ENTRY
        MOVB    @R2,CURCHR      ; SAVE FOR RESTORE NEXT TIME
        CMPB    @R2,#'          ; IS THE CHAR UNDER CURS A BLANK?
```

```
        BNE     1$              ; IF NOT THEN MESSY INVERSING ELSE
        MOVB    #177,@R2        ; WE CAN USE DEL CHAR OF STD SET
        BR      3$              ; (COURTESY TERRY MILLER)
1$:     MOV     #7000,R0        ; MESS...SET UP CHAR GEN ADDR FOR SCANLINES
        BISB    @R2,R0          ; PUT CHAR IN LOW BYTE AND THEN
        ASH     #4,R0           ; SHIFT INTO FINAL ADDR FORMAT
        MOVB    #377,@R2        ; SET CHAR IN BUFFER TO INV DEL
        MOV     #167760,R1      ; ADDRESS OF INVERSE DEL CHAR
        MOV     #12,R2          ; AND LOOP COUNTER
        MOV     #200,@#VCR      ; SET VCR INTO CHAR GEN MODE
2$:     MOVB    (R0)+,@R1       ; CHANGE INV DEL TO INV CHAR
        COMB    (R1)+           ; PATTERN
        SOB     R2,2$           ; DO ALL 10 SCAN LINES
3$:     MOV     (SP)+,@#VCR     ; RESTORE VIDEO CONTROL REG
        MOV     (SP)+,R2        ; RESTORE REGS NOW
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RTS     PC              ; RETURN TO KW$INT FOR PROPER RETURN

; HERE ARE THE ROUTINES FOR SPECIAL CONTROL FUNCTIONS
; THEY HANDLE THE VARIABLES CURADR (IN R2) AND HOMADR
; IN SOME CASES.  MESSY THINGS ARE THE 25 LINE CHAR BUF
; WRAPAROUND (HANDLED IN LF ETC).

$CR:    ; CARRIAGE RETURN...MOVE TO COLUMN 0
        BIC     #177,R2         ; SET COLUMN FIELD TO 0
        BR      EMNEXT
$HT:    ; HORIZONTAL TAB...GOTO NEXT 8TH COLUMN
        BIS     #7,R2
        INC     R2
        BR      EMNEXT
$BS:    ; BACK SPACE
        BIT     #177,R2         ; ARE WE ALREADY IN COLUMN 0?
        BEQ     EMNEXT          ; IF SO THEN SPLIT
        DEC     R2
        BR      EMNEXT
$EM:    ; HOME CURSOR
        MOV     HOMADR,R2       ; POINT CURADR AT HOM ADR
        BR      EMNEXT
$FF:    ; HOME AND CLEAR
        CLR     @#VIR           ; SET SCANNING TO TOP OF BUFFER
        MOV     #160000,R2      ; RESET SCREEN ADDRESSING
        MOV     R2,HOMADR       ; HOME CURSOR AND CLEAR SCREEN
        BR      $VT             ; ERASE TO END-OF-SCREEN
$GS:    ; ERASE-END-OF-LINE
        MOV     R2,-(SP)        ; SAVE CURSOR LOCATION
        JSR     PC,CLRROW       ; BLANK OUT REMAINDER OF LINE
        MOV     (SP)+,R2        ; RESTORE
        BR      EMNEXT
$FS:    ; FORWARD SHIFT...NON-DESTRUCTIVE FORWARD SPACE
        INC     R2
        BR      EMNEXT
$BEL:   ; BELL...SET BEEP BIT IN VCR
        BIS     #4000,@SP       ; START BEEP AT EXIT...(OLD VCR ON TOS)
        BIS     #1000,BELREG    ; SET UP SHIFT REG FOR 8 TICKS
        BR      EMNEXT
$VT:    ; VERTICAL TAB...ERASE-END-OF-SCREEN
        MOV     R2,-(SP)        ; SAVE CURRENT PLACE ON SCREEN
VTLOOP: JSR     PC,CLRROW       ; CLEAR ROW FROM R2 POINTER
        ADD     #60,R2          ; POINT R2 AT NEXT LINE IN BUFF
```

```
        CMP     R2,#166200      ; ARE WE POINTING AT LINE 25?
        BNE     1$
        MOV     #160000,R2      ; POINT BACK AT LINE 0
1$:     CMP     R2,HOMADR       ; HAVE WE CLEARED THRU HOME?
        BNE     VTLOOP          ; GO ZAP NEXT LINE (DOES INVISIBLE TOO)
        MOV     (SP)+,R2        ; RESTORE OLD CURSOR LOC
..NEXT: BR      EMNEXT
BACKUP: ; GO HERE TO BACKUP R1 AND EXIT IN CASE OF
        ; DLE OR RS WHICH DOESNT HAVE ENOUGH CHARS
        ; TO COMPLETE ITS WORK
        DEC     R1              ; REPOINT R1 AT CHAR IN QUESTION
        CMP     R1,#QSTART      ; CHECK FOR REVERSE WRAPAROUND
        BHIS    EMEXIT
        MOV     #QSTART+QSIZE-1,R1
        BR      EMEXIT          ; GO AWAY UNTIL NEXT TICK
$RS:    ; X-Y ADDRESS... RS (X+32,Y+32)
        MOV     QTAIL,R0        ; GRAB TAIL PTR...SEE IF 2 CHARS QUEUED
        SUB     R1,R0           ; R0 IS # CHARS IN QUEUE
        BPL     1$              ; WITH POSSIBLE WRAP
        ADD     #QSIZE,R0
1$:     CMP     R0,#2           ; AT LEAST 2 CHARS LEFT (X Y PARAMS)?
        BLT     BACKUP          ; IF NOT THEN BACKUP AND EXIT
        BIC     #177,R2         ; OK...START OUT IN COL 0
        DEQUR1                  ; GRAB X VALUE FROM STREAM
        SUB     #32.,R0         ; NORMALIZE IT TO REAL COLUMN #

        BIC     #177200,R0      ; ZAP ALL BUT LOW 7 BITS
        MOV     R0,-(SP)        ; AND SAVE FOR FINAL MASKING IN
        DEQUR1                  ; NOW GET ROW #
        BIC     #177740,R0      ; ZAP ALL BUT LOW 5 BITS
        CMP     R0,#24.         ; ROW 24 MEANS NO CHANGE...SKIP STUFF
        BEQ     2$
        MOV     HOMADR,R2       ; CHANGE ROW...START AT HOME
        ASH     #7,R0           ; SHIFT ROW # INTO ROW ADDRESS FIELD
        ADD     R0,R2           ; MOVE TO NEW ROW # (COL IN R2 ALRDY 0)
        CMP     R2,#166200      ; WRAPPED AROUND SCREEN BUF?
        BLO     2$
        SUB     #6200,R2        ; RELOCATE IT TO OTHER END OF BUF
2$:     BIS     (SP)+,R2        ; MASK IN COLUMN NUMBER
        BR      ..NEXT
$LF:    ; LINE FEED...MAYBE SCROLLS
        ADD     #200,R2         ; BUMP R2 TO NEXT SCAN LINE
        CMP     R2,#166200      ; IS IT PAST THE LEGIT CHAR BUFFER?
        BLO     1$              ; IF SO FINE
        BIC     #7600,R2        ; ELSE WRAP TO LINE 0
1$:     MOV     R2,R0           ; NOW FIND CURR ROW #
        MOV     R3,-(SP)
        MOV     HOMADR,R3       ; GRAB ROW #S OF HOME AND CURRENT
        ASH     #-7,R0
        ASH     #-7,R3
        SUB     R3,R0           ; SUBTRACT HOME ROW FROM CUR ROW
        BPL     2$              ; CHECK WRAP AGAIN
        ADD     #25.,R0
2$:     MOV     (SP)+,R3
        CMP     R0,#24.         ; LINE DIF 24 LINES?
        BNE     ..NEXT          ; IF NOT THEN EASY...NO SCROLL
        MOV     R2,-(SP)        ; ELSE STASH R2 FOR CLEAR
        BIC     #177,R2         ; CLEAR LINE...SET "COL" TO ZERO
        JSR     PC,CLRROW
        MOV     (SP)+,R2
```

```
         ADD     #400,@#VIR      ; BUMP SCAN ROW #
         CMP     @#VIR,#25.*256. ; WRAPED AROUND ON HARD SCAN ROW?
         BLO     3$
         CLR     @#VIR
 3$:     ADD     #200,HOMADR     ; FINALLY BUMP HOME ADDRESS
         CMP     HOMADR,#166200  ; WRAP FOR THIS TOO
         BLO     ..NEXT
         BIC     #7600,HOMADR
         BR      ..NEXT
 $US:    ; UP SHIFT- REVERSE LINE FEED
         MOV     R2,R0           ; GRAB CURRENT CURSOR LOC
         BIC     #177,R0         ; ZAP COLUMN # TO ZERO
         CMP     R0,HOMADR       ; ARE WE AT TOP LINE?
         BEQ     ..NEXT          ; IF SO THEN DO NOTHIN
         SUB     #200,R2         ; ELSE GO BACKWARDS 1 LINE
         CMP     R2,#160000      ; CHECK REVERSE WRAPAROUND
         BHIS    ..NEXT
         ADD     #6200,R2        ; BUMP BACK INTO REALITY
         BR      ..NEXT

 CTLTBL: ; TRANSFER TABLE INDEXED BY CONTROL CHAR
         ; DISPATCHES TO PROPER ROUTINE.

         .WORD   $NUL,$BAD,$BAD,$BAD,$BAD,$BAD,$BAD,$BEL
         .WORD   $BS ,$HT ,$LF ,$VT ,$FF ,$CR ,$BAD,$BAD
         .WORD   $BAD,$BAD,$BAD,$BAD,$BAD,$BAD,$BAD,$BAD
         .WORD   $BAD,$EM ,$BAD,$NUL,$FS ,$GS ,$RS ,$US

 CLRROW: ; HANDY ROUTINE TO CLEAR FROM CURRENT POINTER IN
         ; R2 TO END OF THE CURRENT LINE.  R2 RETURNED POINTING
         ; AND 80TH (ILLEGAL) COLUMN.  R0 DESTROYED. R1 UNTOUCHED.
         MOV     R3,-(SP)        ; NEED A REG
         MOV     #80.,R0         ; CALCULATE # CHARS LEFT IN LINE
         MOV     R2,R3           ; GRAB CURRENT BUFFER POINTER
         BIC     #177600,R3      ; CLEAR ALL BUT COLUMN BITS
         SUB     R3,R0           ; R0 NOW HAS CHARS REMAINING
         BLE     2$              ; JUST IN CASE OF LAME CALL
         MOV     #BLANK,R3       ; USE A REG AS SOURCE...MUCH FASTER
 1$:     MOVB    R3,(R2)+        ; PUT BLANK IN BUFFER
         SOB     R0,1$
 2$:     MOV     (SP)+,R3        ; RESTORE
         RTS     PC              ; AND RETURN

         .END

; +-----------------------------------------------------------------+
; |                                                                 |
; |                    F     I     N     I     S                  |
; |                                                                 |
; +-----------------------------------------------------------------+

### END OF FILE UCSD Pascal 1.5 Interp TK.MAC
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 LibMap
##########################################################################

 {$S+}

     (*****************************************************************)
     (*                                                               *)
     (*  Copyright (c) 1978 Regents of the University of California.  *)
     (*  Permission to copy or distribute this software or documen-   *)
     (*  tation in hard or soft copy granted only by written license  *)
     (*  obtained from the Institute for Information Systems.         *)
     (*                                                               *)
     (*****************************************************************)

 {$G+,I-


       LIBRARY MAPPER UTILITY

    written in a great hurry using ....

         UCSD  PASCAL  SYSTEM
           PROGRAM  LINKER

         Written September 78
         by Robert Hofkin
         Major portions stolen from
         Roger T. Sumner


 }

 program LIBMAP;

 const
     MAXSEG = 15;        { max code seg # in code files }
     MAXSEG1 = 16;       { MAXSEG+1, useful for loop vars }
     MAXLC = MAXINT;     { max compiler assigned address }
     MAXIC = 2400;       { max number bytes of code per proc }
     MAXPROC = 160;      { max legal procedure number }

 type

     { subranges }
     { --------- }

     segrange = 0..MAXSEG;       { seg table subscript type }
     segindex = 0..MAXSEG1;      { wish we had const expressions! }
     lcrange = 1..MAXLC;         { base offsets a la P-code }
     icrange = 0..MAXIC;         { legal length for proc/func code }
     procrange = 1..MAXPROC;     { legit procedure numbers }

     { miscellaneous }
     { ------------- }

     alpha = packed array [0..7] of char;

     { link info structures }
```

```
{ ---- ---- ---------- }

placep = ^placerec;         { position in source seg }
placerec = record
             srcbase, destbase: integer;
             length: icrange
           end { placerec } ;

refp = ^refnode;            { in-core version of ref lists }
refnode = record
            next: refp;
            refs: array [0..7] of integer;
          end { refnode } ;

litypes = (EOFMARK,         { end-of-link-info marker }
               { ext ref types, designates      }
               { fields to be updated by linker }
           UNITREF,         { refs to invisibly used units (archaic?) }
           GLOBREF,         { refs to external global addrs }
           PUBLREF,         { refs to BASE lev vars in host }
           PRIVREF,         { refs to BASE vars, allocated by linker }
           CONSTREF,        { refs to host BASE lev constant }
               { defining types, gives      }
               {  linker values to fix refs }
           GLOBDEF,         { global addr location }
           PUBLDEF,         { BASE var location }
           CONSTDEF,        { BASE const definition }
               { proc/func info, assem }
               { to PASCAL and PASCAL  }
               { to PASCAL interface   }
           EXTPROC,         { EXTERNAL proc to be linked into PASCAL }
           EXTFUNC,         {    "     func "  "    "    "      "    }
           SEPPROC,         { Separate proc definition record }
           SEPFUNC,         {   "      func     "        "    }
           SEPPREF,         { PASCAL ref to a sep proc }
           SEPFREF);        {   "    ref to a sep func }

liset = set of litypes;
opformat = (WORD, BYTE, BIG);       { instruction operand field formats }

lientry = record    { format of link info records }
            name: alpha;
            case litype: litypes of
              SEPPREF,
              SEPFREF,
              UNITREF,
              GLOBREF,
              PUBLREF,
              PRIVREF,
              CONSTREF:
                    (format: opformat;      { how to deal with the refs }
                     nrefs: integer;        { words following with refs }
                     nwords: lcrange;       { size of privates in words }
                     reflist: refp);        { list of refs after read in }
              EXTPROC,
              EXTFUNC,
              SEPPROC,
              SEPFUNC:
                    (srcproc: procrange;    { the procnum in source seg }
                     nparams: integer;      { words passed/expected }
```

```
                        place: placep);        { position in source/dest seg }
                  GLOBDEF:
                        (homeproc: procrange;  { which proc it occurs in }
                         icoffset: icrange);   { its byte offset in pcode }
                  PUBLDEF:
                        (baseoffset: lcrange); { compiler assign word offset }
                  CONSTDEF:
                        (constval: integer);   { users defined value }
                  EOFMARK:
                        (nextlc: lcrange)      { private var alloc info }
                end { lientry } ;


    { segment information }
    { ------- ----------- }

    segkinds =(LINKED,          { no work needed, executable as is }
               HOSTSEG,         { PASCAL host program outer block  }
               SEGPROC,         { PASCAL segment procedure, not host }
               UNITSEG,         { library unit occurance/reference }
               SEPRTSEG);       { library separate proc/func TLA segment }



    { host/lib file access info }
    { ---- --- ---- ------ ---- }

    I5segtbl = record   { first full block of all code files }
                 diskinfo: array [segrange] of
                             record
                               codeleng, codeaddr: integer
                             end { diskinfo } ;
                 segname: array [segrange] of alpha;
                 segkind: array [segrange] of segkinds;
                 textstart: array [segrange] of integer;
                 filler: array [0..87] of integer;
                 notice: string [79];
               end { I5segtbl } ;




var
    segtbl: I5segtbl;   { disk seg table w/ source info }
    fp: file;
    mapfile: interactive;
    listmap, listrefs, firsttime: boolean;


{
*  Alphabetic returns TRUE if name contains all legal
*  characters for PASCAL identifiers.  Used to validate
*  segnames and link info entries.
}

function alphabetic (var name: alpha): boolean;
  label 1;
  var i: integer;
begin
  alphabetic := FALSE;
```

```
  for i := 0 to 7 do
    if not (name[i] in ['A'..'Z', '0'..'9', ' ', '_']) then
      goto 1;
  alphabetic := TRUE;
1:
end { alphabetic } ;



procedure phase2;
  var s: segindex;

    {
    *  Readlinkinfo reads in the link info for segment s
    *  and builds its symtab.  Some simple disk io routines
    *  do unblocking.
    }

    procedure readlinkinfo (s: segrange);
      var
          nextblk, recsleft: integer;
          entry: lientry;
          nointerface: boolean;
          buf: array [0..31] of
                 array [0..7] of integer;

      function copyinterface (start: integer): boolean;
        const IMPLMTSY = 52;
        var j: integer; { FIXED DECLARATION ORDER }
            s: integer;
            d: integer;
            n: alpha;
            last: integer;
            done: boolean;
            buf: packed array [0..1023] of char;
      begin
        copyinterface := true;
        if (start <= 0) or (start > 200) then
          begin
            copyinterface := false;
            exit (copyinterface)
          end;
        done := false;
        repeat
          if blockread (fp, buf, 2, start) <> 2 then
            begin
              writeln (mapfile, 'Interface read error');
              copyinterface := false;
              done := true
            end
          else
              begin
                start := start + 2;
                j := 0;
                repeat
                  if buf [j] IN ['A'..'Z', 'a'..'z'] then
                    begin
                      last := j;
                      IDSEARCH (j, buf);
                      done := s = IMPLMTSY;
```

```
                    end;
                  if buf [j] = chr (13) THEN
                    if buf [j+1] = chr (0) THEN
                      begin
                        last := j-1;
                        j := 1023;
                      end;
                  j := j+1
                until done or (j > 1023);
                writeln (mapfile, buf:last)
              end
          until done;
          writeln (mapfile)
        end { copyinterface } ;

{
        *  Getentry reads an 8 word record from disk buf
        *  sequentially.  No validity checking is done here,
        *  only disk read errors.
}

        procedure getentry (var entry: lientry);
          var err: boolean;
        begin
          err := FALSE;
          if recsleft = 0 then
            begin
              recsleft := 32;
              err := blockread (fp, buf, 1, nextblk) <> 1;
              if err then
                  writeln (mapfile, 'library read error!')
              else
                nextblk := nextblk+1
            end;
          moveleft(buf[32-recsleft], entry, 16);
          if err then
            entry.litype := EOFMARK;
          recsleft := recsleft-1
        end { getentry } ;

        procedure ref (what: string);
        var
          nrecs: integer;
          temp: lientry;
        begin
          with entry do
            begin
              if listrefs then
                begin
                  write (mapfile, name:12, what);
                  case format of
                    WORD: write (mapfile, ' word reference');
                    BYTE: write (mapfile, ' byte reference');
                    BIG:  write (mapfile, ' big reference');
                  end;
                  if nrefs > 1 then
                    write (mapfile, ' (', nrefs, ' times)')
                  else
                    write (mapfile, ' (once)');
                  writeln (mapfile);
```

```
            end;
          for nrecs := 1 to (nrefs+7) div 8 do
            getentry (temp);  { skip reference list }
        end { with };
    end { ref };

begin { readlinkinfo }

  with segtbl do
    begin
      write (mapfile, segname[s]);
      nointerface := true;
      case segkind[s] of
        LINKED:  begin
                   writeln (mapfile, '  completely linked segment');
                   exit (readlinkinfo); { rein a faire }
                 end;
        HOSTSEG: writeln (mapfile, '  Pascal host outer block');
        SEGPROC: begin
                   writeln (mapfile, '  Pascal segment');
                   nointerface := not copyinterface (textstart[s])
                 end;
        UNITSEG: begin
                   writeln (mapfile,'  library unit');
                   nointerface := not copyinterface (textstart[s])
                 end;
        SEPRTSEG: writeln (mapfile, '  separate procedure segment');
      end;

      recsleft := 0;      { 8 wd recs left in buf }
      with diskinfo[s] do
        nextblk := codeaddr + (codeleng+511) div 512;  { seek to linkinfo }

      if listmap or nointerface then
        repeat
         getentry(entry);
         with entry do
           if litype <> EOFMARK then
             begin { list the entry }
               if alphabetic (name) then
                 begin
                   case litype of
                     GLOBDEF:  if listmap then
                                 writeln (mapfile, name:12,
                          ' global addr    P #',homeproc,',  I #',icoffset);
                     PUBLDEF:  if listmap then
                                 writeln (mapfile, name:12,
                                        ' public var  base = ', baseoffset);
                     CONSTDEF: if listmap then
                                 writeln (mapfile, name:12,
                                          ' constant value of ', constval);
                     EXTPROC,
                     EXTFUNC:  if listrefs then
                                 writeln (mapfile, name:12,
                                           ' external proc  P #', srcproc);
                     SEPPROC,
                     SEPFUNC:  writeln (mapfile, name:12,
                                           ' separate proc  P #', srcproc);
                     GLOBREF: ref (' global');
                     PUBLREF: ref (' public');
```

```
                        CONSTREF: ref (' constant');
                        SEPFREF,
                        SEPPREF: ref (' separate');
                        UNITREF: ref (' unit');
                        PRIVREF:  ref (' private');
                      end { case };
                    end { if alphabetic };
                end { with entry };
            until entry.litype = EOFMARK
        end { with segtbl }
    end { readlinkinfo } ;


begin { phase2 }

  for s := 0 to MAXSEG do
      with segtbl, diskinfo[s] do
        if codeleng > 0 then
          begin
            write (mapfile, 'Segment #', s:2, ':  ');
            readlinkinfo (s);
            writeln (mapfile);
            writeln (mapfile,
   '------------------------------------------------------------------------':75);
            writeln (mapfile);
          end;

end { phase2 } ;


  procedure getfile;
  label 1;
  var
    s: segindex;
    CH: char;
    libtitle, maptitle: string;
  begin
1:
    close (fp);
    write ('enter library name: ');
    readln (libtitle);
    if libtitle = '' then
      begin
        close (mapfile, lock);
        exit (program);
      end;

    if libtitle = '*' then
      libtitle := '*SYSTEM.LIBRARY';

    reset (fp, libtitle);

    if ioresult <> 0 then
      begin
        insert ('.CODE', libtitle, length(libtitle)+1);
        reset (fp, libtitle)
      end;

    if blockread (fp, segtbl, 1, 0) <> 1 then
      begin
```

```
        writeln ('bad file');
        goto 1;
      end;

    with segtbl do
      for s := 0 to MAXSEG do
        if (diskinfo[s].codeleng < 0) or (diskinfo[s].codeaddr < 0) or
           (diskinfo[s].codeaddr > 300) or not alphabetic (segname[s]) then
          begin
            writeln ('not a code file');
            goto 1;
          end;

     write ('list linker info table (Y/N)? ');
     repeat
       read (keyboard, CH)
     until CH in ['Y', 'N', 'y', 'n', ' '];
     if not eoln (keyboard) then writeln (CH);
     listmap := (CH in ['Y', 'y']);

     if listmap then
       begin
         write ('list referenced items (Y/N)? ');
         repeat
           read (keyboard, CH)
         until CH in ['Y', 'N', 'y', 'n', ' '];
         if not eoln (keyboard) then writeln (CH);
         listrefs := (CH in ['Y', 'y'])
       end
     else
       listrefs := false;

     if firsttime then
       repeat
         write ('map output file name: ');
         readln (maptitle);
         if maptitle = '' then
           exit (program);

         if maptitle[length(maptitle)] = '.' then
           delete (maptitle, length(maptitle), 1)
         else
           if maptitle[length(maptitle)] <> ':' then
             insert ('.TEXT', maptitle, length(maptitle)+1);

         rewrite (mapfile, maptitle)
       until ioresult = 0;

    page (mapfile);
    writeln (mapfile, '  LIBRARY MAP FOR ', libtitle);
    writeln (mapfile);

    with segtbl do
      if length (notice) > 0 then
        begin
          writeln (mapfile, ' ':5, notice);
          writeln (mapfile);
        end;
    writeln (mapfile);
  end { gettitle } ;
```

```
 begin  { main }

   writeln ('Library map utility  [I.5:4]');

   firsttime := true;

   repeat
     getfile;
     firsttime := false;
     phase2;
   until false;

 END.

{ +-----------------------------------------------------------------+
  |                                                                 |
  |                     F     I     N     I     S                   |
  |                                                                 |
  +-----------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 LibMap
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Librarian
########################################################################


 (* UCSD PASCAL I.5 P-SYSTEM "LIBRARIAN" *)

 PROGRAM PLIBRARIAN;

 (*$U-*)

 CONST
      MAXSEG = 15;       (*MAX CODE SEGMENT NUMBER*)

 TYPE
                                        (*CODE SEGMENT LAYOUTS*)

      SEGRANGE = 0..MAXSEG;
      SEGDESC = RECORD
                  DISKADDR: INTEGER;     (*REL BLK IN CODE...ABS IN SYSCOM^*)
                  CODELENG: INTEGER      (*# BYTES TO READ IN*)
                END (*SEGDESC*) ;
                                        (*SYSTEM COMMUNICATION AREA*)
                                        (*SEE INTERPRETERS...NOTE  *)
                                        (*THAT WE ASSUME BACKWARD  *)
                                        (*FIELD ALLOCATION IS DONE *)
      SYSCOMREC = RECORD
                    IORSLT: INTEGER ;    (*RESULT OF LAST IO CALL*)
                    XEQERR: INTEGER;     (*REASON FOR EXECERROR CALL*)
                    SYSUNIT: INTEGER;    (*PHYSICAL UNIT OF BOOTLOAD*)
                    BUGSTATE: INTEGER;   (*DEBUGGER INFO*)
                    GDIRP: INTEGER;
                    LASTMP,STKBASE,BOMBP: INTEGER;
                    MEMTOP,SEG,JTAB: INTEGER;
                    BOMBIPC: INTEGER;    (*WHERE XEQERR BLOWUP WAS*)
                    HLTLINE: INTEGER;    (*MORE DEBUGGER STUFF*)
                    BRKPTS: ARRAY [0..3] OF INTEGER;
                    RETRIES: INTEGER;    (*DRIVERS PUT RETRY COUNTS*)
                    EXPANSION: ARRAY [0..8] OF INTEGER;
                    HIGHTIME,LOWTIME: INTEGER;
                    MISCINFO: PACKED RECORD
                                NOBREAK,STUPID,SLOWTERM,
                                HASXYCRT,HASLCCRT,HAS8510A,HASCLOCK: BOOLEAN
                              END;
                    CRTTYPE: INTEGER;
                    CRTCTRL: PACKED RECORD
                               RLF,NDFS,ERASEEOL,ERASEEOS,HOME,ESCAPE: CHAR;
                               BACKSPACE: CHAR;
                               FILLCOUNT: 0..255;
                               EXPANSION: PACKED ARRAY [0..3] OF CHAR
                             END;
                    CRTINFO: PACKED RECORD
                               WIDTH,HEIGHT: INTEGER;
                               RIGHT,LEFT,DOWN,UP: CHAR;
                               BADCH,CHARDEL,STOP,BREAK,FLUSH,EOF: CHAR;
                               ALTMODE,LINEDEL: CHAR;
                               EXPANSION: PACKED ARRAY [0..5] OF CHAR
                             END;
                    SEGTABLE: ARRAY [SEGRANGE] OF
```

```
                              RECORD
                                CODEUNIT: INTEGER;
                                CODEDESC: SEGDESC
                              END
                END (*SYSCOM*);

VAR
    SYSCOM: ^SYSCOMREC;                 (*MAGIC PARAM...SET UP IN BOOT*)

SEGMENT PROCEDURE LIBRARIAN(III,JJJ: INTEGER);

CONST
  WINDOW = 2;
  ERROR = 23;
  MARKCODE = 15;
  MARKIN = 5;


TYPE
     BLOCK0P = ^BLOCK0;
     BLOCK0 = RECORD
                SEGDSC: ARRAY [SEGRANGE] OF SEGDESC;
                SEGNAME: ARRAY [SEGRANGE] OF
                           PACKED ARRAY [0..7] OF CHAR;
                SEGKIND: ARRAY [SEGRANGE] OF INTEGER;
                EXTRA: ARRAY [SEGRANGE] OF INTEGER;
                FILLER: ARRAY [1..88] OF INTEGER;
                NOTICE: STRING[79]
              END;


VAR NBLOCKS,RSLT,OUTBLOCK: INTEGER;
    BUF: BLOCK0P;
    DSEG,SSEG: SEGRANGE;
    PL,TITLE: STRING;
    CODETBL: BLOCK0;
    CODE,INFILE: FILE;

PROCEDURE NEWLINKER;

VAR CCH: CHAR;
    INTBL: BLOCK0P;
    NTITLE: STRING;
    CODETABLE: BLOCK0P;
    PL: STRING;

PROCEDURE PROMPT(AT: INTEGER);
BEGIN
  GOTOXY(0,AT);
  IF AT = ERROR THEN WRITE(CHR(7));
  WRITE(PL);
  WITH SYSCOM^.CRTCTRL DO WRITE(ESCAPE,ERASEEOL);
END;

FUNCTION CHECKIO:BOOLEAN;
VAR RSLT:INTEGER;
BEGIN
  CHECKIO:=IORESULT=0;
  IF IORESULT <> 0 THEN
    BEGIN
```

```
      RSLT:=IORESULT;
      PL := 'I/O error # ';
      PROMPT(ERROR);
      WRITE(OUTPUT,RSLT);
    END;
END; (* CHECKIO *)

PROCEDURE OPENFILE;
BEGIN
  REPEAT
    PL := 'Link Code File -> ';
    PROMPT(4);
    READLN(INPUT,NTITLE);
    IF LENGTH(NTITLE) > 0 THEN
      BEGIN
        TITLE := NTITLE;
        RESET(INFILE,NTITLE);
      END;
  UNTIL (CHECKIO) OR (LENGTH(NTITLE) = 0);
END (*OPENFILE*) ;

PROCEDURE DISPLAY(AT: INTEGER; WHAT: BLOCK0P);
VAR
  T: INTEGER;
BEGIN
  GOTOXY(0,AT);
  WITH WHAT^ DO
    FOR T := 0 TO 3 DO
      BEGIN
        WRITE(T:3,'-',SEGNAME[T],SEGDSC[T].CODELENG:6);
        WRITE(T+4:5,'-',SEGNAME[T+4],SEGDSC[T+4].CODELENG:6);
        WRITE(T+8:5,'-',SEGNAME[T+8],SEGDSC[T+8].CODELENG:6);
        WRITELN(T+12:5,'-',SEGNAME[T+12],SEGDSC[T+12].CODELENG:6)
      END;
  PL := 'Code file length - ';
  PROMPT(12);
  WRITE(OUTPUT,OUTBLOCK);
END;


PROCEDURE LINKCODE;
  VAR NBLOCKS: INTEGER;

  PROCEDURE LINKIT;

    PROCEDURE COPYLINKINFO(INFOBLK: INTEGER);
      VAR N, NRECS: INTEGER;
          DONE: BOOLEAN;
          REC: ARRAY [0..7] OF INTEGER;
          BUF: ARRAY [0..31, 0..7] OF INTEGER;

        PROCEDURE GETREC;
        BEGIN
          IF NRECS = 0 THEN
            IF BLOCKREAD(INFILE, BUF, 1, INFOBLK) <> 1 THEN
              BEGIN
                PL := 'Link info read err';
                PROMPT(ERROR);
                DONE := TRUE
              END
```

```
        ELSE
          IF BLOCKWRITE(CODE, BUF, 1, OUTBLOCK) <> 1 THEN
            BEGIN
              PL := 'Code file overflow';
              PROMPT(ERROR);
              DONE := TRUE
            END
          ELSE
            BEGIN
              OUTBLOCK := OUTBLOCK+1;
              INFOBLK := INFOBLK+1;
              NRECS := 32
            END;
      IF NOT DONE THEN
        REC := BUF[32-NRECS];
      NRECS := NRECS-1
    END { GETREC } ;

BEGIN { COPYLINKINFO }
  NRECS := 0; DONE := FALSE;
  REPEAT
    GETREC;
    IF NOT (REC[4] IN [0..14]) THEN
      BEGIN
        PL := 'Bad link info';
        PROMPT(ERROR);
        REC[4] := 0
      END;
    DONE := REC[4] = 0;
    IF NOT DONE THEN
      IF REC[4] IN [1..5,13,14] THEN
        BEGIN { COPY REF LIST }
          N := (REC[6]+7) DIV 8;
          WHILE N > 0 DO
            BEGIN GETREC; N := N-1 END
        END
  UNTIL DONE
END { COPYLINKINFO } ;

PROCEDURE COPYINTERFACE(START: INTEGER);
  CONST IMPLMTSY = 52;
  VAR J: INTEGER; { FIXED DECLARATION ORDER }
      S: INTEGER;
      O: INTEGER;
      N: PACKED ARRAY [0..7] OF CHAR;
      DONE: BOOLEAN;
      BUF: PACKED ARRAY [0..1023] OF CHAR;
BEGIN
  IF (START <= 0) OR (START > 200) THEN
    EXIT(COPYINTERFACE);
  CODETABLE^.EXTRA[DSEG] := OUTBLOCK;
  DONE := FALSE;
  REPEAT
    IF BLOCKREAD(INFILE, BUF, 2, START) <> 2 THEN
      BEGIN
        PL := 'Interface read err';
        PROMPT(ERROR);
        DONE := TRUE
      END
    ELSE
```

```
      IF BLOCKWRITE(CODE, BUF, 2, OUTBLOCK) <> 2 THEN
        BEGIN
          PL := 'Interface write err';
          PROMPT(ERROR);
          DONE := TRUE
        END
      ELSE
        BEGIN
          START := START+2;
          OUTBLOCK := OUTBLOCK+2;
          J := 0;
          REPEAT
            IF BUF[J] IN ['A'..'Z', 'a'..'z'] THEN
              BEGIN
                IDSEARCH(J,BUF);
                DONE := S = IMPLMTSY;
                IF DONE THEN
                  IF J < 510 THEN
                    OUTBLOCK := OUTBLOCK-1
              END;
            IF BUF[J] = CHR(13) THEN
              IF BUF[J+1] = CHR(0) THEN
                J := 1023;
            J := J+1
          UNTIL DONE OR (J > 1023)
        END
  UNTIL DONE
END { COPYINTERFACE } ;

BEGIN
  WITH INTBL^,SEGDSC[SSEG] DO
    BEGIN
      NBLOCKS := (CODELENG+511) DIV 512;
      IF BLOCKREAD(INFILE,BUF^,NBLOCKS,DISKADDR) <> NBLOCKS THEN
        BEGIN
          PL := 'Error reading seg ';
          PROMPT(ERROR);
          WRITE(OUTPUT,SSEG)
        END
      ELSE
        IF BLOCKWRITE(CODE,BUF^,NBLOCKS,OUTBLOCK) <> NBLOCKS THEN
          BEGIN
            PL := 'I/O error - no room on disk';
            PROMPT(ERROR);
          END
        ELSE
          BEGIN
            CODETABLE^.SEGNAME[DSEG] := SEGNAME[SSEG];
            CODETABLE^.SEGDSC[DSEG].CODELENG := CODELENG;
            CODETABLE^.SEGDSC[DSEG].DISKADDR := OUTBLOCK;
            OUTBLOCK := OUTBLOCK+NBLOCKS;
            IF (SEGKIND[SSEG] < 0) OR (SEGKIND[SSEG] > 4) THEN
              SEGKIND[SSEG] := 0;
            CODETABLE^.SEGKIND[DSEG] := SEGKIND[SSEG];
            CODETABLE^.EXTRA[DSEG] := 0;
            IF SEGKIND[SSEG] <> 0 THEN
              COPYLINKINFO(DISKADDR+NBLOCKS);
            IF (SEGKIND[SSEG] IN [3,4]) THEN
              COPYINTERFACE(EXTRA[SSEG])
          END
```

```
        END;
      DISPLAY(MARKCODE,CODETABLE);
    END;

FUNCTION CONFIRM: BOOLEAN;
  VAR
    N: INTEGER;
  BEGIN
    CONFIRM:=FALSE;
    (*get segment*)
    N:= 0;
    PL := '';
    PROMPT(WINDOW);
    REPEAT
      READ(CCH);
      IF CCH = CHR(8) THEN
        N := N DIV 10;
      IF CCH IN ['0'..'9'] THEN
        N := N*10 + ORD(CCH)-ORD('0')
    UNTIL NOT (CCH IN [CHR(8),'0'..'9']);
    IF CCH <> ' ' THEN (*probably N or Q*)
      EXIT(CONFIRM);
    IF N IN [0..MAXSEG] THEN (*good segment number*)
      WITH INTBL^ DO
        IF SEGDSC[N].CODELENG > 0 THEN (*any chunk of code*)
          BEGIN
            SSEG := N;
            REPEAT
              PL := 'Seg to link into? ';
              PROMPT(WINDOW);
              READ(DSEG)
            UNTIL DSEG IN [0..MAXSEG];
            READ(CCH); { EAT XTRA CHAR }
            CCH := 'Y'; (* TRICK THE REPLACEMENT BELOW *)
            IF (CODETABLE^.SEGDSC[DSEG].CODELENG <> 0) THEN (*linking again*)
              BEGIN
                PL :=
'WARNING - Segment already linked.  Please Reconfirm (y/n) - ';
                PROMPT(WINDOW);
                READ(INPUT,CCH);
                WRITELN(OUTPUT);
              END;
            CONFIRM := CCH IN ['Y','y']
          END;
  END; (* CONFIRM *)

BEGIN
  IF LENGTH(NTITLE)>0 THEN
    IF BLOCKREAD(INFILE,INTBL^,1,0) = 1 THEN
      DISPLAY(MARKIN,INTBL)
    ELSE
      BEGIN
        RSLT:=IORESULT;
        PL := 'Read error # ';
        PROMPT(ERROR);
        WRITE(OUTPUT,RSLT);
      END;
  PL :=
'Segment # to link and <space>, N(ew file, Q(uit, A(bort';
  PROMPT(0);
```

```
  REPEAT
    IF CONFIRM THEN LINKIT;
  UNTIL CCH IN ['N','Q','A','n','q','a'];
  CLOSE(INFILE)
END (*LINKCODE*) ;

BEGIN
  PAGE(OUTPUT);
  PL := 'Pascal System Librarian';
  PROMPT(0);
  NEW(CODETABLE);
  NEW(INTBL);
  PL := 'Output code file -> ';
  REPEAT
    PROMPT(11);
    READLN(INPUT,TITLE);
    IF LENGTH(TITLE) = 0 THEN EXIT(LIBRARIAN)
    ELSE REWRITE(CODE,TITLE)
  UNTIL (LENGTH(TITLE) = 0) OR (CHECKIO);
  OUTBLOCK := 1; NEW(BUF);
  IF SIZEOF(BLOCK0) <> 512 THEN
    HALT;
  FILLCHAR(CODETABLE^, SIZEOF(BLOCK0), 0);
  WITH CODETABLE^ DO
    FOR DSEG := 0 TO MAXSEG DO
      SEGNAME[DSEG] := '        ';
  REPEAT
    OPENFILE;
    LINKCODE;
  UNTIL CCH IN ['Q','q','A','a'];
  IF CCH IN ['A','a'] THEN EXIT(LIBRARIAN);
  PL := 'Notice? ';
  PROMPT(23);
  READLN(CODETABLE^.NOTICE);
  IF BLOCKWRITE(CODE,CODETABLE^,1,0) = 1 THEN
    CLOSE(CODE,LOCK)
  ELSE
    WRITELN(OUTPUT,'Code write error ')
END { NEWLINKER } ;

{
FUNCTION CHECKIO:BOOLEAN;
VAR RSLT:INTEGER;

BEGIN
  CHECKIO:=IORESULT=0;
  IF IORESULT <> 0 THEN
    BEGIN
    RSLT:=IORESULT;
    WRITELN(OUTPUT,'I/O error # ',RSLT);
    END;
END; (* CHECKIO *)

FUNCTION OPENFILE: BOOLEAN;
BEGIN
  REPEAT
    WRITE(OUTPUT,'Link Code File? '); READLN(INPUT,TITLE);
    IF LENGTH(TITLE) > 0 THEN RESET(INFILE,TITLE);
  UNTIL (CHECKIO) OR (LENGTH(TITLE) = 0);
  OPENFILE := LENGTH(TITLE) > 0
```

```
END (*OPENFILE*) ;

PROCEDURE LINKCODE;
  VAR NBLOCKS: INTEGER;
      INTBL: BLOCK0;

  FUNCTION CONFIRM:BOOLEAN;
  VAR CH:CHAR;
  BEGIN
    CONFIRM:=FALSE;
    WITH INTBL DO
      BEGIN
      IF SEGDSC[DSEG].CODELENG > 24 THEN
        BEGIN
        WRITE(OUTPUT,'Linking ',SEGNAME[DSEG],'.  Please Confirm (y/n)');
        READ(INPUT,CH);
        WRITELN(OUTPUT);
        IF (CODETBL.SEGDSC[DSEG].CODELENG <> 0) AND (CH IN ['Y','y']) THEN
          BEGIN
          WRITE(OUTPUT,
'WARNING - segment already linked.  Please Reconfirm');
          READ(INPUT,CH);
          WRITELN(OUTPUT);
          END;
        CONFIRM := CH IN ['y','Y'];
        END;
      END;
  END; (* CONFIRM *)

BEGIN
  IF BLOCKREAD(INFILE,INTBL,1,0) = 1 THEN
    BEGIN
      WITH INTBL DO
        FOR DSEG := 0 TO MAXSEG DO
            WITH SEGDSC[DSEG] DO
              IF CONFIRM THEN
                BEGIN NBLOCKS := (CODELENG+511) DIV 512;
                  IF BLOCKREAD(INFILE,BUF^,NBLOCKS,DISKADDR) <> NBLOCKS THEN
                    WRITELN(OUTPUT,'Error reading seg ',DSEG)
                  ELSE
                    IF BLOCKWRITE(CODE,BUF^,NBLOCKS,OUTBLOCK) <> NBLOCKS THEN
                      WRITELN(OUTPUT,'I/O error - no room on disk')
                    ELSE
                      BEGIN
                        WRITELN(OUTPUT,SEGNAME[DSEG],' Seg # ',DSEG,', Block ',
                                OUTBLOCK,', ',CODELENG,' Bytes');
                        CODETBL.SEGNAME[DSEG] := SEGNAME[DSEG];
                        CODETBL.SEGDSC[DSEG].CODELENG := CODELENG;
                        CODETBL.SEGDSC[DSEG].DISKADDR := OUTBLOCK;
                        OUTBLOCK := OUTBLOCK + NBLOCKS
                      END
                END
    END
  ELSE
    BEGIN
    RSLT:=IORESULT;
    WRITELN(OUTPUT,'Input file read error # ',RSLT);
    END;
  CLOSE(INFILE)
END (*LINKCODE*) ;
```

```
 BEGIN
   IF NOT SYSCOM^.MISCINFO.SLOWTERM THEN NEWLINKER
   ELSE
     BEGIN
       REPEAT
         WRITE(OUTPUT,'Output code file? '); READLN(INPUT,TITLE);
         IF LENGTH(TITLE) > 0 THEN REWRITE(CODE,TITLE)
       UNTIL (LENGTH(TITLE) = 0) OR (CHECKIO);
       IF LENGTH(TITLE) > 0 THEN
         BEGIN OUTBLOCK := 1; NEW(BUF);
           WITH CODETBL DO
             FOR DSEG := 0 TO MAXSEG DO
               BEGIN  SEGNAME[DSEG] := '        ';
                 SEGDSC[DSEG].CODELENG := 0;
                 SEGDSC[DSEG].DISKADDR := 0
               END;
           WHILE OPENFILE DO LINKCODE;
           WRITE('Notice:');
           READLN(CODETBL.NOTICE);
           IF BLOCKWRITE(CODE,CODETBL,1,0) = 1 THEN CLOSE(CODE,LOCK)
           ELSE
             WRITELN(OUTPUT,'Code file write error ')
         END
     END
 }
 BEGIN
   NEWLINKER
 END { LIBRARIAN } ;

 BEGIN END.

### END OF FILE UCSD Pascal 1.5 Librarian
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 Linker
##########################################################################


{$S+}
{ $I link0 }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

{$U-,R+


        UCSD  PASCAL  SYSTEM
          PROGRAM  LINKER

        Written summer '78 by
        Roger T. Sumner, IIS

        Copyright (c) 1978, Regents of
        the University of California

    All hope abandon ye who enter here
                        -Dante

}

program systemlevel;

const
    SYSPROG = 4;

var
    syscom: ^integer;
    gfiles: array [0..5] of integer;
    userinfo: record
                filler: array [0..4] of integer;
                slowterm, stupid: boolean;
                altmode: char;
                gotsym, gotcode: boolean;
                workvid, symvid, codevid: string[7];
                worktid, symtid, codetid: string[15]
              end;
    filler: array [0..4] of integer;
    syvid, dkvid: string[7];
    junk1, junk2: integer;
    cmdstate: integer;

{
*  The linker is made up of three phases:
*     Phase1 which open all input files, reads up seg tables
*            from them and decides which segments are to be
```

```
*            linked into the final code file.
*     Phase2 reads the linker info for each segment that is
*            going to be used, either to select sep procs from
*            or copy with modifications into output code.
*            The main symbol tree are built here, one for each
*            code segment.
*     Phase3 does the crunching of code segments into their
*            final form by figuring out the procs that need to
*            be linked in, resolves all references (PUBLREF,
*            GLOBREF, etc), patches the code pointed to by their
*            reflists, and writes the final code seg(s).
}

segment procedure linker(iii, jjj: integer);

const
    MAXSEG = 15;        { max code seg # in code files }
    MAXSEG1 = 16;       { MAXSEG+1, useful for loop vars }
    MASTERSEG = 1;      { USERHOST segment number # }
    FIRSTSEG =  7;      { first linker assignable seg # }
    MAXFILE = 7;        { number of lib files we can use }
    MAXLC = MAXINT;     { max compiler assigned address }
    MAXIC = 14000;      { max number bytes of code per proc }
    MAXPROC = 160;      { max legal procedure number }
    MSDELTA = 12;       { mark stack size for pub/priv fixup }

type

    { subranges }
    { --------- }

    segrange = 0..MAXSEG;       { seg table subscript type }
    segindex = 0..MAXSEG1;      { wish we had const expressions! }
    lcrange = 1..MAXLC;         { base offsets a la P-code }
    icrange = 0..MAXIC;         { legal length for proc/func code }
    procrange = 1..MAXPROC;     { legit procedure numbers }

    { miscellaneous }
    { ------------- }

    alpha = packed array [0..7] of char;
    diskblock = packed array [0..511] of 0..255;
    codefile = file;            { trick compiler to get ^file }
    filep = ^codefile;
    codep = ^diskblock;         { space management...non-PASCAL kludge }

    { link info structures }
    { ---- ---- ---------- }

    placep = ^placerec;         { position in source seg }
    placerec = record
                 srcbase, destbase: integer;
                 length: icrange
               end { placerec } ;

    refp = ^refnode;            { in-core version of ref lists }
    refnode = record
                next: refp;
                refs: array [0..7] of integer;
              end { refnode } ;
```

```
litypes = (EOFMARK,         { end-of-link-info marker }
               { ext ref types, designates      }
               { fields to be updated by linker }
           UNITREF,         { refs to invisibly used units (archaic?) }
           GLOBREF,         { refs to external global addrs }
           PUBLREF,         { refs to BASE lev vars in host }
           PRIVREF,         { refs to BASE vars, allocated by linker }
           CONSTREF,        { refs to host BASE lev constant }
               { defining types, gives      }
               {  linker values to fix refs }
           GLOBDEF,         { global addr location }
           PUBLDEF,         { BASE var location }
           CONSTDEF,        { BASE const definition }
               { proc/func info, assem }
               { to PASCAL and PASCAL  }
               { to PASCAL interface   }
           EXTPROC,         { EXTERNAL proc to be linked into PASCAL }
           EXTFUNC,         {    "     func "  "    "    "      "    }
           SEPPROC,         { Separate proc definition record }
           SEPFUNC,         {   "      func     "        "    }
           SEPPREF,         { PASCAL ref to a sep proc }
           SEPFREF);        {   "    ref to a sep func }

liset = set of litypes;
opformat = (WORD, BYTE, BIG);       { instruction operand field formats }

lientry = record    { format of link info records }
            name: alpha;
            case litype: litypes of
              SEPPREF,
              SEPFREF,
              UNITREF,
              GLOBREF,
              PUBLREF,
              PRIVREF,
              CONSTREF:
                    (format: opformat;      { how to deal with the refs }
                     nrefs: integer;        { words following with refs }
                     nwords: lcrange;       { size of private or nparams }
                     reflist: refp);        { list of refs after read in }
              EXTPROC,
              EXTFUNC,
              SEPPROC,
              SEPFUNC:
                    (srcproc: procrange;    { the procnum in source seg }
                     nparams: integer;      { words passed/expected }
                     place: placep);        { position in source/dest seg }
              GLOBDEF:
                    (homeproc: procrange;   { which proc it occurs in }
                     icoffset: icrange);    { its byte offset in pcode }
              PUBLDEF:
                    (baseoffset: lcrange);  { compiler assign word offset }
              CONSTDEF:
                    (constval: integer);    { users defined value }
              EOFMARK:
                    (nextlc: lcrange)       { private var alloc info }
            end { lientry } ;

{ symbol table items }
```

```
    { ------ ----- ----- }

    symp = ^symbol;
    symbol = record
               llink, rlink,            { binary subtrees for diff names }
               slink: symp;             { same name, diff litypes }
               entry: lientry           { actual id information }
             end { symbol } ;

    { segment information }
    { ------- ----------- }

    segkinds =(LINKED,          { no work needed, executable as is }
               HOSTSEG,         { PASCAL host program outer block  }
               SEGPROC,         { PASCAL segment procedure, not host }
               UNITSEG,         { library unit occurance/reference }
               SEPRTSEG);       { library separate proc/func TLA segment }

    finfop = ^fileinforec;      { forward type dec }

    segp = ^segrec;             { this structure provides access to all }
    segrec = record             { info for segs to be linked to/from    }
               srcfile: finfop;         { source file of segment }
               srcseg: segrange;        { source file seg # }
               symtab: symp;            { symbol table tree }
               case segkind: segkinds of
                 SEPRTSEG:
                        (next: segp)    { used for library sep seg list }
             end { segrec } ;

    { host/lib file access info }
    { ---- --- ---- ------ ---- }

    I5segtbl = record   { first full block of all code files }
                 diskinfo: array [segrange] of
                             record
                               codeleng, codeaddr: integer
                             end { diskinfo } ;
                 segname: array [segrange] of alpha;
                 segkind: array [segrange] of segkinds;
                 filler: array [0..143] of integer
               end { I5segtbl } ;

    filekind = (USERHOST, USERLIB, SYSTEMLIB);

    fileinforec = record
                    next: finfop;       { link to next file thats open }
                    code: filep;        { pointer to PASCAL file...sneaky! }
                    fkind: filekind;    { used to validate the segkinds }
                    segtbl: I5segtbl    { disk seg table w/ source info }
                  end { fileinforec } ;


var
    hostfile,           { host file info ptr, its next = libfiles }
    libfiles: finfop;   { list of lib files, user and system }

    seplist: segp;      { list of sep segs to search through }
    reflitypes: liset;  { those litypes with ref lists }
```

```
    talkative,
    useworkfile: boolean;

    errcount: integer;
    heapbase: ^integer;

    hostsp: segp;                        { ptr to host prog outer block }
    nextbaselc: lcrange;                 { next base offset for private alloc }
    seginfo: array [segrange] of segp;   { seg is available if NIL }
    nextseg: segindex;                   { next slot in seginfo available }

    mapname: string[40];

    f0, f1, f2, f3,
    f4, f5, f6, f7,                      { input files with lurking pntrs }
    code: codefile;                      { output code file, *system.wrk.code }


{
*  Print an error message and bump
*  the error counter.
}

procedure error(msg: string);
  var ch: char;
begin
  writeln(msg);
  repeat
    write('Type <sp>(continue), <esc>(terminate)');
    read(keyboard, ch);
    if ch = userinfo.altmode then
      exit(linker)
  until ch = ' ';
  errcount := errcount+1
end { error } ;

{
*  Routines to access object code segments.  There
*  is subtle business involving byte flipping with
*  the 16-bit operations.  This needs more research
*  when the time comes.
}
{$R-}

function fetchbyte(cp: codep; offset: integer): integer;
begin
  fetchbyte := cp^[offset]
end { fetchbyte } ;

function fetchword(cp: codep; offset: integer): integer;
  var i: integer;
begin
  moveleft(cp^[offset], i, 2);
  { byte swap i }
  fetchword := i
end { fetchword } ;

procedure storebyte(val: integer; cp: codep; offset: integer);
begin
  cp^[offset] := val
```

```
end { storebyte } ;

procedure storeword(val: integer; cp: codep; offset: integer);
begin
  { byte swap val }
  moveleft(val, cp^[offset], 2)
end { storeword } ;

{$R+}

{
*  Enter newsym in symtab tree.  The tree is binary for
*  different names and entries with the same name are entered
*  onto sideways links (slink).  No check is made for dup
*  entry types, caller must do that.  Nodes on slink will
*  always have NIL rlink and llink.
}

procedure entersym(newsym: symp; var symtab: symp);
  var syp, lastsyp: symp;
      useleft: boolean;
begin
  newsym^.llink := NIL;
  newsym^.rlink := NIL;
  newsym^.slink := NIL;
  if symtab = NIL then
    symtab := newsym
  else
    begin { search symtab and add newsym }
      syp := symtab;
      repeat
        lastsyp := syp;
        if syp^.entry.name > newsym^.entry.name then
          begin syp := syp^.llink; useleft := TRUE end
        else
          if syp^.entry.name < newsym^.entry.name then
            begin syp := syp^.rlink; useleft := FALSE end
          else { equal }
            begin { add into sideways list }
              newsym^.slink := syp^.slink;
              syp^.slink := newsym;
              lastsyp := NIL;     { already added flag }
              syp := NIL          { stop repeat loop }
            end
      until syp = NIL;
      if lastsyp <> NIL then
        begin { add to bottom of tree }
          if useleft then
            lastsyp^.llink := newsym
          else
            lastsyp^.rlink := newsym
        end
    end { symtab <> NIL }
end { entersym } ;

{
*  Look up name in symtab tree and return pointer
*  to it.  Oktype restricts what litype is
*  acceptable.  NIL is returned if name not found.
}
```

```
function symsrch(var name: alpha; oktype: litypes; symtab: symp): symp;
  var syp: symp;
begin
  symsrch := NIL;
  syp := symtab;
  while syp <> NIL do
    if syp^.entry.name > name then
      syp := syp^.llink
    else
      if syp^.entry.name < name then
        syp := syp^.rlink
      else { equal name }
        if syp^.entry.litype <> oktype then
          syp := syp^.slink
        else { found! }
          begin symsrch := syp; syp := NIL end
end { symsrch } ;

{
*  Search for the occurance of the unit segment
*  given by name in the list of files in fp.
*  Return the file and segment number in seg.
*  NIL is returned for non-existant units and
*  an error is given.
}

function unitsrch(fp: finfop; var name: alpha; var seg: segrange): finfop;
  label 1;
  var s: segindex;
begin seg := 0;
  while fp <> NIL do
    begin
      with fp^.segtbl do
        for s := 0 to MAXSEG do
          if segname[s] = name then
            if segkind[s] = UNITSEG then
              goto 1;
      fp := fp^.next
    end;
  write('Unit ', name);
  error(' not found');
  s := 0;
1:
  seg := s;
  unitsrch := fp
end { unitsrch } ;

{
*  Alphabetic returns TRUE if name contains all legal
*  characters for PASCAL identifiers.  Used to validate
*  segnames and link info entries.
}

function alphabetic(var name: alpha): boolean;
  label 1;
  var i: integer;
begin
  alphabetic := FALSE;
  for i := 0 to 7 do
```

```
    if not (name[i] in ['A'..'Z', '0'..'9', ' ', '_']) then
      goto 1;
  alphabetic := TRUE;
1:
end { alphabetic } ;

{
*  Getcodep is a sneaky routine to point codep's anywhere
*  in memory.  It violates Robot's Rules of Order, but is
*  very useful for dealing with the variable size segments
}

function getcodep(memaddr: integer): codep;
  var r: record
           case boolean of
             TRUE:  (i: integer);
             FALSE: (p: codep)
           end;
begin
  r.i := memaddr;
  getcodep := r.p
end { getcodep } ;

{ $I link1 }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

{
*  Phase 1 opens host and library files and
*  reads in seg tables.  All fields are verified
*  and the hostfile/libfiles file list is built.
*  The prototype final seg table is set up in
*  seginfo[*] from the host file and the sep seg
*  list is set up for searching in later phases.
}

procedure phase1;

    {
    *  Build file list opens input code files and reads segtbls.
    *  The var hostfile is set up as head of linked list of file
    *  info recs.  The order of these files determines how id's
    *  will be searched for.  Note that libfiles points at the
    *  list just past the host file front entry.
    }

    procedure buildfilelist;
      label 1;
      var f: 0..MAXFILE;
          i: integer;
          p, q: finfop;
          fname: string[40];
```

```
{
*  Setupfile opens file and enters new finfo rec in
*  hostfile list.  Segtbl is read in and validated.
}

procedure setupfile(num: integer; kind: filekind; title: string);
  var errs: integer;
      s: segindex;
      cp: filep;
      fp: finfop;
      alllinked: boolean;
      goodkinds: set of segkinds;

    {
    *  Getfilep returns a pointer to a file using unspeakable
    *  methods, but the ends justify the means.
    }

    function getfilep(var f: codefile): filep;
      var a: array [0..0] of filep;
    begin
      {$R-}
      getfilep := a[-1];
      {$R+}
    end { getfilep } ;

begin { setupfile }
  case num of
    0:  cp := getfilep(f0);
    1:  cp := getfilep(f1);
    2:  cp := getfilep(f2);
    3:  cp := getfilep(f3);
    4:  cp := getfilep(f4);
    5:  cp := getfilep(f5);
    6:  cp := getfilep(f6);
    7:  cp := getfilep(f7)
  end { cases } ;
  reset(cp^, title);
  if IORESULT <> 0 then
    if title <> 'in workspace' then
      begin
        insert('.CODE', title, length(title)+1);
        reset(cp^, title)
      end;
  if IORESULT <> 0 then
    begin
      insert('No file ', title, 1);
      error(title);
      if kind <> USERHOST then
        errcount := errcount-1
    end
  else
    begin { file open ok }
      if talkative then
        writeln('Opening ', title);
      new(fp);
      fp^.next := hostfile;
      fp^.code := cp;
      fp^.fkind := kind;
      if blockread(cp^, fp^.segtbl, 1, 0) <> 1 then
```

```
            error('segtbl read err')
          else
            begin { now check segtbl values }
              s := 0; alllinked := TRUE;
              errs := errcount;
              if kind = USERHOST then
                goodkinds := [LINKED,SEGPROC,SEPRTSEG,HOSTSEG,UNITSEG]
              else
                goodkinds := [LINKED,UNITSEG,SEPRTSEG];
              with fp^.segtbl do
                repeat
                  alllinked := alllinked and (segkind[s] = LINKED);
                  if (diskinfo[s].codeleng = 0)
                  and (segkind[s] <> LINKED) then
                    if (kind <> USERHOST)
                    or (segkind[s] <> UNITSEG) then
                      error('funny code seg');
                  if (diskinfo[s].codeleng < 0)
                  or (diskinfo[s].codeaddr < 0)
                  or (diskinfo[s].codeaddr > 300) then
                    error('bad diskinfo');
                  if not (segkind[s] in goodkinds) then
                    error('bad seg kind');
                  if not alphabetic(segname[s]) then
                    error('bad seg name');
                  if errcount > errs then
                    s := MAXSEG;
                  s := s+1
                until s > MAXSEG;
              if alllinked and (kind = USERHOST) then
                begin
                  write('All segs linked');
                  exit(linker)
                end;
              if errcount = errs then
                hostfile := fp            { ok file...link in }
            end
        end
    end { setupfile } ;

begin { buildfilelist }
  if talkative then
    begin
      for i := 1 to 7 do
        writeln;
      writeln('Linker [I.5]')
    end;
  useworkfile := cmdstate <> SYSPROG;
  with userinfo do
    if useworkfile then
      begin
        if gotcode then
          fname := concat(codevid, ':', codetid)
        else
          fname := 'in workspace';
        setupfile(0, USERHOST, fname);
        setupfile(1, SYSTEMLIB, '*SYSTEM.LIBRARY')
      end
    else
      begin
```

```
          write('Host file? ');
          readln(fname);
          if fname = '' then
            if gotcode then
              fname := concat(codevid, ':', codetid)
            else
              fname := 'in workspace';
          setupfile(0, USERHOST, fname);
          if errcount > 0 then
            exit(linker); { no host! }
          for f := 1 to MAXFILE do
            begin
              write('Lib file? ');
              readln(fname);
              if fname = '' then
                goto 1;
              if fname = '*' then
                setupfile(f, SYSTEMLIB, '*SYSTEM.LIBRARY')
              else
                setupfile(f, USERLIB, fname)
            end;
      1:
          write('Map name? ');
          readln(mapname);
          if mapname <> '' then
            if mapname[length(mapname)] = '.' then
              delete(mapname, length(mapname), 1)
            else
              insert('.TEXT', mapname, length(mapname)+1)
        end;

    { now reverse list so host is }
    { first and syslib is last    }

    p := hostfile; hostfile := NIL;
    repeat
      q := p^.next;
      p^.next := hostfile;
      hostfile := p;
      p := q
    until p = NIL;
    libfiles := hostfile^.next;
  end { buildfilelist } ;

  {
  *  Buildseginfo initializes the seginfo table from
  *  the host prototype seg table.  All legal states
  *  are checked, and imported units found.  This
  *  leaves a list of all segs to finally appear in
  *  the output code file.
  }

  procedure buildseginfo;
    label 1;
    var s: segindex;
        errs: integer;
        sp: segp;
  begin
    with hostfile^.segtbl do
      for s := 0 to MAXSEG do
```

```
        if (segkind[s] = LINKED)
        and (diskinfo[s].codeleng = 0) then
          seginfo[s] := NIL   { not in use }
        else
          begin { do something with seg }
            errs := errcount;
            new(sp);
            sp^.srcfile := hostfile;
            sp^.srcseg := s;
            sp^.symtab := NIL;
            sp^.segkind := segkind[s];
            case sp^.segkind of
              SEGPROC,
              LINKED:    ;  { nothing to check! }

              HOSTSEG:   if s <> MASTERSEG then
                           error('bad host seg')
                         else
                           if hostsp <> NIL then
                             error('dup host seg')
                           else
                             hostsp := sp;

              SEPRTSEG:  if s = MASTERSEG then
                           sp^.next := NIL
                         else
                           begin { put into seplist }
                             sp^.next := seplist;
                             seplist := sp;
                             sp := NIL
                           end;

              UNITSEG:   if diskinfo[s].codeleng = 0 then
                           sp^.srcfile := unitsrch(libfiles,
                                                   segname[s],
                                                   sp^.srcseg)
            end { cases } ;
            if errs = errcount then
              seginfo[s] := sp
            else
              seginfo[s] := NIL
          end;

      { now find first assignable seg }

      for s := FIRSTSEG to MAXSEG do
        if seginfo[s] = NIL then
          goto 1;
      s := MAXSEG1;
    1:
      nextseg := s;
      if seginfo[MASTERSEG] = NIL then
        error('wierd host')
    end { buildseginfo } ;

    {
    *  Buildseplist searches through libraries and adds onto
    *  a global list of sep segs that are to be searched
    *  for procs and globals.  They are initially build in
    *  the reverse order, then reversed again so searches
```

```
    *  will go in the order the files were specified.
    }

    procedure buildseplist;
      var sp, p, q: segp;
          fp: finfop;
          s: segindex;
    begin
      fp := libfiles;
      while fp <> NIL do
        begin
          for s := 0 to MAXSEG do
            if fp^.segtbl.segkind[s] = SEPRTSEG then
              begin
                new(sp);
                sp^.next := seplist;
                sp^.srcfile := fp;
                sp^.srcseg := s;
                sp^.symtab := NIL;
                sp^.segkind := SEPRTSEG;
                sp^.next := seplist;
                seplist := sp
              end;
          fp := fp^.next
        end;

      { now reverse the list to maintain original order }

      p := seplist; seplist := NIL;
      while p <> NIL do
        begin
          q := p^.next;
          p^.next := seplist;
          seplist := p;
          p := q
        end
    end { buildseplist } ;

begin { phase1 }

  { initialize globals }

  hostfile := NIL;
  libfiles := NIL;
  hostsp := NIL;
  seplist := NIL;
  reflitypes := [UNITREF, GLOBREF, PUBLREF,
                 PRIVREF, CONSTREF,
                 SEPPREF, SEPFREF];
  errcount := 0;
  nextbaselc := 3;
  mapname := '';
  talkative := not userinfo.slowterm;
  mark(heapbase);
  unitwrite(3, heapbase^, 35);

  { build list of input files }

  buildfilelist;
  if errcount > 0 then
```

```
    exit(linker);

  { init basic seg info table }

  buildseginfo;
  if errcount > 0 then
    exit(linker);

  { finally build sep seg list }

  buildseplist;
  if errcount > 0 then
    exit(linker)
end { phase1 } ;

{ $I link2 }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

{
*  Phase2 reads in all linker info associated with
*  the segs in seginfo and sep seg list.  Again all
*  fields are checked carefully.  As a help to phase3,
*  ref lists are collected and place records for sep
*  proc/func are computed.  Some small optimization is
*  done to eliminate the sep seg list if it is not
*  going to be needed, saving a few disk IO's.
}

procedure phase2;
  var s: segindex;
      sp: segp;
      dumpseps: boolean;

    {
    *  Readlinkinfo reads in the link info for segment sp
    *  and builds its symtab.  Some simple disk io routines
    *  do unblocking, and all fields are again verified.
    *  The only legal litypes are in oktypes. Assume that
    *  sp <> NIL
    }

    procedure readlinkinfo(sp: segp; oktypes: liset);
      var rp, rq: refp;
          syp: symp;
          errs, nrecs, nextblk, recsleft: integer;
          entry, temp: lientry;
          buf: array [0..31] of
                 array [0..7] of integer;

        {
        *  Getentry reads an 8 word record from disk buf
        *  sequentially.  No validity checking is done here,
```

```
*  only disk read errors.
}

procedure getentry(var entry: lientry);
  var err: boolean;
begin
  err := FALSE;
  if recsleft = 0 then
    begin
      recsleft := 32;
      err := blockread(sp^.srcfile^.code^, buf, 1, nextblk) <> 1;
      if err then
        error('li read err')
      else
        nextblk := nextblk+1
    end;
  moveleft(buf[32-recsleft], entry, 16);
  if err then
    entry.litype := EOFMARK;
  recsleft := recsleft-1
end { getentry } ;

{
*  Addunit is called to find or allocate a library unit
*  that is found in link info as an external ref.  This
*  occurs in lib units which use other units.  If
*  the unit can't be found or no room, error is called.
}

procedure addunit(var name: alpha);
  var fp: finfop; seg: integer;
begin
  fp := unitsrch(hostfile, name, seg);
  if fp <> NIL then
    if fp <> hostfile then
      if fp^.segtbl.diskinfo[seg].codeleng <> 0 then
        if nextseg = MAXSEG1 then
          error('no room in seginfo')
        else
          begin { allocate new seginfo el }
            new(seginfo[nextseg]);
            with seginfo[nextseg]^ do
              begin
                srcfile := fp;
                srcseg := seg;
                segkind := UNITSEG;
                symtab := NIL
              end;
            nextseg := nextseg+1
          end
end { addunit } ;

{
*  Validate verifies lientry format.
*  If the entry is SEPPROC or FUNC
*  then a place rec is allocated for buildplace.  If
*  a UNITREF is found, it searched for and possibly
*  allocated.  If the unit must be added to seginfo,
*  it is placed after current position so it will have
*  its link info read as well.
```

```
    }

    procedure validate(var entry: lientry);
    begin
      with entry do
        if not alphabetic(name) then
          error('non-alpha name')
        else
          case litype of
            SEPPREF,
            SEPFREF,
            UNITREF,
            GLOBREF,
            PUBLREF,
            PRIVREF,
            CONSTREF:  begin
                         reflist := NIL;
                         if (nrefs < 0)
                         or (nrefs > 500) then
                           error('too many refs');
                         if not (format in [WORD, BYTE, BIG]) then
                           error('bad format');
                         if litype = PRIVREF then
                           if (nwords <= 0)
                           or (nwords > MAXLC) then
                             error('bad private');
                         if (litype = UNITREF) and (nrefs > 0) then
                           addunit(name)
                       end;
            GLOBDEF:   if (homeproc <= 0)
                       or (homeproc > MAXPROC)
                       or (icoffset < 0)
                       or (icoffset > MAXIC) then
                         error('bad globdef');
            PUBLDEF:   if (baseoffset <= 0)
                       or (baseoffset > MAXLC) then
                         error('bad publicdef');
            EXTPROC,
            EXTFUNC,
            SEPPROC,
            SEPFUNC:   begin
                         if litype in [SEPPROC,SEPFUNC] then
                           new(place) { for use in buildplaces }
                         else
                           place := NIL;
                         if (srcproc <= 0)
                         or (srcproc > MAXPROC)
                         or (nparams < 0)
                         or (nparams > 100) then
                           error('bad proc/func')
                       end
          end { case litype }
    end { validate } ;

  begin { readlinkinfo }
    recsleft := 0;      { 8 wd recs left in buf }
    with sp^.srcfile^.segtbl, diskinfo[sp^.srcseg] do
      begin { seek to linkinfo }
        nextblk := codeaddr + (codeleng+511) div 512;
        if talkative then
```

```
        writeln('Reading ', segname[sp^.srcseg])
    end;
  repeat
    getentry(entry);
    errs := errcount;
    if entry.litype <> EOFMARK then
      if entry.litype in oktypes then
        validate(entry)
      else
        begin
          error('bad litype');
          entry.litype := EOFMARK
        end;
    if dumpseps then
      if entry.litype in [SEPPREF, SEPFREF,
                          EXTPROC, EXTFUNC,
                          GLOBREF] then
        dumpseps := FALSE;  { we need them! }
    if entry.litype in reflitypes then
      begin { read ref list }
        nrecs := (entry.nrefs+7) div 8;
        while nrecs > 0 do
          begin { read ref rec }
            getentry(temp);
            new(rp);
            moveleft(temp, rp^.refs, 16);
            rp^.next := entry.reflist;
            entry.reflist := rp;
            nrecs := nrecs-1
          end;
        { reverse ref list }
        rp := entry.reflist;
        entry.reflist := NIL;
        while rp <> NIL do
          begin
            rq := rp^.next;
            rp^.next := entry.reflist;
            entry.reflist := rp;
            rp := rq
          end
      end;
    if entry.litype = EOFMARK then
      if sp^.segkind = HOSTSEG then
        if (entry.nextlc > 0)
        and (entry.nextlc <= MAXLC) then
          nextbaselc := entry.nextlc
        else
          error('bad host LC')
      else
    else
      if errs = errcount then
        begin { ok...add to symtab }
          new(syp);
          syp^.entry := entry;
          entersym(syp, sp^.symtab)
        end
  until entry.litype = EOFMARK
end { readlinkinfo } ;

{
```

```
*  Buildplaces reads code of sep segs from disk to generate
*  the placerec entries for use during phase3.  The seg is
*  read into the heap and the grossness begins. Assume that
*  sp <> NIL
}

procedure buildplaces(sp: segp);
  var cp: codep; heap: ^integer;
      nbytes, nblocks, nprocs, n: integer;

    {
    *  procsrch recursivly searches symtab of sp to find
    *  sepproc and sepfunc entries and build the actual
    *  place record for the link info entry by indexing
    *  thru proc dict to jtab and using entric field.
    }

    procedure procsrch(symtab: symp);
      var i, j: integer;
    begin
      if symtab <> NIL then
        begin
          procsrch(symtab^.llink);
          procsrch(symtab^.rlink);
          procsrch(symtab^.slink);
          with symtab^.entry do
            if litype in [SEPPROC, SEPFUNC] then
              if (srcproc <= 0) or (srcproc > nprocs) then
                error('bad proc #')
              else { find byte place in code }
                begin
                  i := nbytes-2-2*srcproc;      { point i at proc dict }
                  i := i-fetchword(cp, i);      { point i at jtab }
                  if (fetchbyte(cp, i) <> srcproc)
                  and (fetchbyte(cp, i) <> 0) then
                    error('disagreeing p #')
                  else
                    begin
                      j := fetchword(cp, i-2)+4;
                      place^.srcbase := i+2-j;
                      if (place^.srcbase < 0)
                      or (j <= 0) or (j > MAXIC) then
                        error('proc place err')
                      else
                        place^.length := j
                    end
                end
        end
    end { procsrch } ;

begin { buildplaces }
  nbytes := sp^.srcfile^.segtbl.diskinfo[sp^.srcseg].codeleng;
  nblocks := (nbytes+511) div 512;
  if memavail-400 < nblocks*256 then
    error('sep seg 2 big')
  else
    begin { alloc space in heap }
      mark(heap);
      n := nblocks;
      repeat
```

```
            new(cp);
            n := n-1
          until n <= 0;
          if blockread(sp^.srcfile^.code^, heap^, nblocks,
               sp^.srcfile^.segtbl.diskinfo[sp^.srcseg].codeaddr) <> nblocks then
            error('sep seg read err')
          else
            begin
              cp := getcodep(ord(heap));
              nprocs := fetchbyte(cp, nbytes-1);
              if (nprocs < 0) or (nprocs > MAXPROC) then
                error('bad proc dict')
              else
                procsrch(sp^.symtab)
            end;
          release(heap)
        end
    end { buildplaces } ;

begin { phase2 }

  mark(heapbase);
  unitwrite(3, heapbase^, 35);

  { read link info for host segs }

  dumpseps := TRUE;     { assume we don't need sep segs }
  for s := 0 to MAXSEG do
    if seginfo[s] <> NIL then
      case seginfo[s]^.segkind of
        LINKED:    ; { nothin }
        UNITSEG:   readlinkinfo(seginfo[s], [PUBLREF, PRIVREF, UNITREF,
                                             CONSTDEF,EXTPROC, EXTFUNC]);
        SEPRTSEG:  readlinkinfo(seginfo[s], [GLOBREF, GLOBDEF,
                                             CONSTDEF,SEPPROC, SEPFUNC]);
        HOSTSEG:   readlinkinfo(seginfo[s], [PUBLDEF, CONSTDEF,
                                             EXTPROC, EXTFUNC]);
        SEGPROC:   readlinkinfo(seginfo[s], [EXTPROC, EXTFUNC])
      end { cases } ;

  { now do sep list elements }

  if dumpseps then
    seplist := NIL;
  sp := seplist;
  while sp <> NIL do
    begin
      readlinkinfo(sp, reflitypes+[CONSTDEF, GLOBDEF, SEPPROC, SEPFUNC]);
      sp := sp^.next
    end;

  { build proc place entries for sep segs }

  if seginfo[MASTERSEG]^.segkind = SEPRTSEG then
    buildplaces(seginfo[MASTERSEG]);

  sp := seplist;
  while sp <> NIL do
    begin
      buildplaces(sp);
```

```
      sp := sp^.next
    end;
  if errcount > 0 then
    exit(linker)
end { phase2 } ;

{ $I link3a }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

{
*  Phase3 of the linker does all the real work of code
*  massaging.  For each segment in seginfo to be placed
*  into the output code file, all referenced procedures
*  and functions are found, globals and other refs are
*  resolved, and finally the final code segment is built.
*  In the case of a SEPRTSEG host (eg an interpreter), then
*  all the procs in it are put in the unresolved list and
*  the host seg is made to appear as just another sep seg.
*  This drags along all the original procedures and maintains
*  their original ordering for possible ASECT integrity.
}

procedure phase3;
  type
      workp = ^workrec;         { all seg work is driven by these lists }
      workrec = record
                  next: workp;          { list link }
                  refsym,               { symtab entry of unresolved name }
                  defsym: symp;         {   "      "   "  resolving entry }
                  refseg,               { seg refls point into, refrange only }
                  defseg: segp;         { seg where defsym was found }
                  case litypes of       { same as litype in refsym^.entry }
                    SEPPREF,
                    SEPFREF,
                    GLOBREF:
                        (defproc: workp);       { work item of homeproc }
                    UNITREF:
                        (defsegnum: segrange);  { resolved seg #, def = ref }
                    PRIVREF:
                        (newoffset: lcrange);   { newly assigned base offset }
                    EXTPROC,
                    EXTFUNC,
                    SEPPROC,
                    SEPFUNC:
                        (needsrch: boolean;     { refs haven't been found }
                         newproc: 0..MAXPROC)   { proc #, comp or link chosen }
                  end { workrec } ;             { 0 implies added proc }

  var s: segindex;
      segbase: codep;   { address of current seg being crunched }
      segleng,          { final code seg length for writeout }
      nextblk: integer; { next available output code block }
```

```
  uprocs,           { unresolved external proc/func work list }
  procs,            { resolved list of above items }
  ulocal,           { unresolved list of updates for seginfo entry }
  local,            { resolved list of fixups that came along with seg }
  uother,           { unresolved work list of things other than procs }
  other: workp;     { resolved list of above }
  sephost: boolean; { flag for interpreter host case (only seg #1) }
  fname: string[39];{ output code file name }
  segtbl: I5segtbl; { output code's seg table }
  map: text;        { map text output file }

{
*  Buildworklists is called for all segments which need to
*  be copied, and maybe need to have sepprocs or others stuff
*  fixed up within them.  The idea here is to get a list
*  of procs and other item needing attention, with
*  all the subtle implications of global defs falling
*  in procs which are not yet selected for linking etc.
*  In fact, three lists are built:
*     The procs list with all procs and func to be grabbed
*  from the various sep segs.
*     The local list of refs in the original segment which must
*  ALL be fixed up such as public or private refs in a unit seg.
*     The other list which has work items which have at least one
*  ref occuring in the procs or funcs in the procs list.
}

procedure buildworklists;
  var sp: segp;
      wp: workp;

    {
    *  Findprocs goes through symtab and builds a list of
    *  procedure and functions which occur in the tree and
    *  whose litype is in the okset.  The resulting list
    *  is not ordered in any particular fashion.  It is
    *  called to build initial uproc list.
    }

    function findprocs(okset: liset; symtab: symp): workp;
      var work: workp;

        {
        *  procsrch recursivly searches subtrees to pick out
        *  those symbols which are in the okset, generates
        *  new work nodes, and puts them into local work list.
        }

        procedure procsrch(sym: symp);
          var wp: workp;
        begin
          if sym <> NIL then
            begin
              procsrch(sym^.llink);
              procsrch(sym^.rlink);
              procsrch(sym^.slink);
              if sym^.entry.litype in okset then
                begin { place new node in list }
                  new(wp);
                  wp^.refsym := sym;
```

```
                wp^.refseg := NIL;
                wp^.defsym := NIL;
                wp^.defseg := NIL;
                wp^.needsrch := TRUE;
                if sephost then
                  wp^.newproc := 0  { see readsrcseg! }
                else
                  wp^.newproc := sym^.entry.srcproc;
                wp^.next := work;
                work := wp
              end
          end
      end { procsrch } ;

begin { findprocs }
  work := NIL;
  procsrch(symtab);
  findprocs := work
end { findprocs } ;

{
*  Findnewprocs is called to place new procedures into the
*  uprocs work list that are needed to resolve GLOBDEFs,
*  SEPPREFs, and SEPFREFs.  The other list is traversed and
*  for each element whose defining proc has not been added
*  into the uprocs list, the defining proc is located and
*  added into uprocs.
}

procedure findnewprocs;
  var wp, wp1: workp;
      pnum: integer;

    {
    *  Findnadd finds the procedure numbered pnum in the
    *  symbol table symtab.  An error is given if the
    *  required proc cannot be found. It returns a work
    *  node for the proc once it has been found.  This
    *  node is also added into the uprocs list.  Any procs
    *  added this way are "invisible", dragged along because
    *  of global refs/defs.
    }

    function findnadd(symtab: symp): workp;

        {
        *  procsrch recursivly searches the sym tree looking
        *  for the actual symbol containing pnum.  This does
        *  most of the work of findnadd.
        }

        procedure procsrch(sym: symp);
          var wp: workp;
        begin
          if sym <> NIL then
            begin
              procsrch(sym^.llink);
              procsrch(sym^.rlink);
              procsrch(sym^.slink);
              if sym^.entry.litype in [SEPPROC, SEPFUNC] then
```

```
                    if sym^.entry.srcproc = pnum then
                      begin
                        new(wp);
                        wp^.refsym := sym;
                        wp^.refseg := NIL;
                        wp^.defsym := NIL;
                        wp^.defseg := NIL;
                        wp^.needsrch := TRUE;
                        wp^.newproc := 0;
                        wp^.next := uprocs;
                        uprocs := wp;
                        findnadd := wp;
                        exit(findnadd)
                      end
                end
            end { procsrch } ;

        begin { findnadd }
          findnadd := NIL;
          procsrch(symtab);
          { if we get here then didnt find it }
          error('missing proc')
        end { findnadd } ;

      begin { findnewprocs }
        wp := other;      { assume only globref, seppref, sepfref in list }
        while wp <> NIL do
          begin
            if wp^.defproc = NIL then
              begin { find proc/func needed }
                if wp^.refsym^.entry.litype = GLOBREF then
                  pnum := wp^.defsym^.entry.homeproc
                else { assume a SEP proc/func }
                  pnum := wp^.defsym^.entry.srcproc;
                wp1 := procs;
                while wp1 <> NIL do
                  if wp^.defseg = wp1^.defseg then
                    if wp1^.defsym^.entry.srcproc = pnum then
                      begin { already gonna be linked }
                        wp^.defproc := wp1;
                        wp1 := NIL
                      end
                    else
                      wp1 := wp1^.next
                  else
                    wp1 := wp1^.next;
                if wp^.defproc = NIL then { forcibly link it }
                  wp^.defproc := findnadd(wp^.defseg^.symtab)
              end;
            wp := wp^.next
          end { while }
      end { findnewprocs } ;

      {
      *  Resolve removes work items from inlist, searches symtabs
      *  for its corresponding definition symbol (error if not found),
      *  and moves the work item into the output list.  Each flavor
      *  of work item needs some special handling to collect extra
      *  info related to specific things.  In general, defsym and
      *  defseg are filled in.  The insert algorithm is special for
```

```
*  procedure types to make life easier on refsrch.
}

procedure resolve(var inlist, outlist: workp);
  var seg: segrange;
      err: boolean;
      wp: workp;

    {
    *  Sepsrch sequentially search the symtabs in the seplist
    *  to resolve the refsym of inlist^.  It basically just
    *  calls symsrch repetively and fixes up defsym and
    *  defseg fields.  If the name of the refsym could
    *  not be found, an error is given.
    }

    procedure sepsrch(oktype: litypes);
      var syp: symp;
          sp: segp;
    begin
      sp := seplist;
      while sp <> NIL do
        begin
          syp := symsrch(inlist^.refsym^.entry.name,
                         oktype, sp^.symtab);
          if syp <> NIL then
            begin
              inlist^.defsym := syp;
              inlist^.defseg := sp;
              sp := NIL
            end
          else
            sp := sp^.next
        end
    end { sepsrch } ;

    {
    *  Procinsert is called to insert work into the procs
    *  list using a special set of sort keys so that copyin-
    *  procs will run reasonably fast and use the disk
    *  efficiently.  The procs list is sorted by segment,
    *  srcbase keys.  The seg ordering is dictated by the
    *  seplist, so user ASECTS etc will retain their original
    *  ordering.
    }

    procedure procinsert(work: workp);
      label 1;
      var crnt, prev: workp;
          sp: segp;
    begin
      prev := NIL;
      sp := seplist;
      while sp <> outlist^.defseg do
        if sp = work^.defseg then
          goto 1
        else
          sp := sp^.next;
      crnt := outlist;
      repeat
```

```
          if crnt^.defseg = work^.defseg then
            repeat
              if work^.defsym^.entry.place^.srcbase <
                 crnt^.defsym^.entry.place^.srcbase then
                goto 1;
              prev := crnt;
              crnt := crnt^.next;
              if crnt = NIL then
                goto 1
            until crnt^.defseg <> work^.defseg
          else
            begin
              prev := crnt;
              crnt := crnt^.next;
              if crnt <> NIL then
                while sp <> crnt^.defseg do
                  if sp = work^.defseg then
                    goto 1
                  else
                    sp := sp^.next
            end
        until crnt = NIL;
    1:
        if prev = NIL then
          begin
            work^.next := outlist;
            outlist := work
          end
        else
          begin
            work^.next := prev^.next;
            prev^.next := work
          end
      end { procinsert } ;

  begin { resolve }
    while inlist <> NIL do
      begin
        with inlist^, refsym^.entry do
          case litype of
            GLOBREF:    begin
                          sepsrch(GLOBDEF);
                          defproc := NIL
                        end;

            CONSTREF:   if hostsp <> NIL then
                          begin
                            defsym := symsrch(name, CONSTDEF,
                                              hostsp^.symtab);
                            defseg := hostsp
                          end;

            PUBLREF:    if hostsp <> NIL then
                          begin
                            defsym := symsrch(name, PUBLDEF,
                                              hostsp^.symtab);
                            defseg := hostsp
                          end;

            PRIVREF:    begin
```

```
                    newoffset := nextbaselc;
                    nextbaselc := nextbaselc+nwords;
                    if hostsp <> NIL then
                      defsym := refsym;
                    defseg := hostsp
                  end;
      EXTPROC,
      SEPPROC,
      SEPPREF:    begin
                    sepsrch(SEPPROC);
                    if litype = SEPPREF then
                      defproc := NIL;
                    err := FALSE;
                    if defsym <> NIL then
                      if litype = SEPPREF then
                        err := defsym^.entry.nparams <> nwords
                      else
                        err := defsym^.entry.nparams <> nparams;

                    if err then
                      begin
                        write('Proc ', name);
                        error(' param mismatch')
                      end
                  end;
      EXTFUNC,
      SEPFUNC,
      SEPFREF:    begin
                    sepsrch(SEPFUNC);
                    if litype = SEPFREF then
                      defproc := NIL;
                    err := FALSE;
                    if defsym <> NIL then
                      if litype = SEPFREF then
                        err := defsym^.entry.nparams <> nwords
                      else
                        err := defsym^.entry.nparams <> nparams;
                    if err then
                      begin
                        write('Func ', name);
                        error(' param mismatch')
                      end
                  end;

      UNITREF:    if unitsrch(hostfile, name, seg) = hostfile then
                    begin { will be found in host }
                      defsym := refsym;
                      defsegnum := seg
                    end
                  else { "impossible" }
                    error('unit err')
    end { cases } ;

  wp := inlist;
  inlist := wp^.next;
  if wp^.defsym = NIL then
    with wp^.refsym^.entry do
      begin
        case litype of
          GLOBREF:  write('Global ');
```

```
                PUBLREF:  write('Public ');
                CONSTREF: write('Const ');
                SEPPREF,
                EXTPROC:  write('Proc ');
                SEPFREF,
                EXTFUNC:  write('Func ')
              end { cases } ;
              write(name);
              error(' undefined')
            end
        else
          if  (wp^.defsym^.entry.litype in [SEPPROC, SEPFUNC])
          and (outlist <> NIL) then
            procinsert(wp)
          else
            begin
              wp^.next := outlist;
              outlist := wp
            end
      end { while }
  end { resolve } ;

  {
  *  Refsrch slowly goes through all reference lists in symbols
  *  which are in the okset to see if any "occur" within the
  *  procedures/functions selected to be linked, that is contained
  *  in procs list.  It is assumed that procs is sorted by defseg
  *  so only the procs between ipl and lpl are searched.
  *  Any symbols which have any refs in selected procs are given
  *  work nodes and are placed in the uother list in no certain
  *  order so resolve can be called right away.
  }

  procedure refsrch(okset: liset; sp: segp);
    var lpl, ipl: workp;
        diffseg: boolean;

      {
      *  Checkrefs recursivly searches sym tree to kind names
      *  in the okset.  When one is found, each of its ref pointers
      *  are checked to see if they fall in one of the procs
      *  to-be-linked (between ipl & lpl).  If so, a new work item
      *  is generated and it's put on the uother list.
      }

      procedure checkrefs(sym: symp);
        label 1, 2;
        var pl, wp: workp;
            i, n, ref: integer;
            rp: refp;
      begin
        if sym <> NIL then
          begin
            checkrefs(sym^.llink);
            checkrefs(sym^.rlink);
            checkrefs(sym^.slink);
            with sym^.entry do
              if litype in okset then
                begin
                  n := nrefs;
```

```
                    rp := reflist;
                    while rp <> NIL do
                      begin
                        if n > 8 then
                          begin
                            i := 7;
                            n := n-8
                          end
                        else
                          i := n-1;
                        repeat { for each ref }
                          ref := rp^.refs[i];
                          pl := ipl;
                          repeat { search proc list }
                            if pl^.needsrch then
                              with pl^.defsym^.entry.place^ do
                                if ref < srcbase then
                                  goto 2 { terminate proc search }
                                else
                                  if ref < srcbase+length then
                                    begin { occurs in proc }
                                      new(wp);
                                      wp^.refsym := sym;
                                      wp^.refseg := sp;
                                      wp^.defsym := NIL;
                                      wp^.defseg := NIL;
                                      wp^.next := uother;
                                      uother := wp;
                                      goto 1
                                    end;
                            pl := pl^.next
                          until pl = lpl;
                        2:
                          i := i-1
                        until i < 0;
                        rp := rp^.next
                      end { while }
                  end
            end;
      1:
        end { checkrefs } ;

    begin { refsrch }
      ipl := NIL;
      lpl := procs;
      while lpl <> NIL do
        if (lpl^.defseg = sp)
        and lpl^.needsrch then
          begin
            ipl := lpl;
            lpl := NIL
          end
        else
          lpl := lpl^.next;
      if ipl <> NIL then
        begin
          lpl := ipl;
          repeat
            diffseg := lpl^.defseg <> ipl^.defseg;
            if not diffseg then
```

```
            lpl := lpl^.next
          until diffseg or (lpl = NIL);
          checkrefs(sp^.symtab);
          repeat
            ipl^.needsrch := FALSE;
            ipl := ipl^.next
          until ipl = lpl
        end
    end { refsrch } ;

    {
    *  findlocals recursivly searches the main segs symtab to
    *  place any unresolved things like public refs in unit
    *  segs into the ulocal list so they can be fixed up in
    *  fixuprefs in addition to the sep proc things.
    }

    procedure findlocals(sym: symp);
      var wp: workp;
    begin
      if sym <> NIL then
        begin
          findlocals(sym^.llink);
          findlocals(sym^.rlink);
          findlocals(sym^.slink);
          if sym^.entry.litype in [UNITREF, PUBLREF, PRIVREF] then
            begin
              new(wp);
              wp^.refsym := sym;
              wp^.refseg := NIL;
              wp^.defsym := NIL;
              wp^.defseg := NIL;
              wp^.next := ulocal;
              ulocal := wp
            end
        end
    end { findlocals } ;

begin { buildworklists }
  procs := NIL;
  local := NIL;
  other := NIL;
  uprocs := NIL;
  ulocal := NIL;
  uother := NIL;
  with seginfo[s]^ do
    if segkind <> LINKED then
      begin
        sephost := segkind = SEPRTSEG;
        if sephost then
          begin
            next := seplist;
            seplist := seginfo[s];
            uprocs := findprocs([SEPPROC, SEPFUNC], symtab)
          end
        else
          uprocs := findprocs([EXTPROC, EXTFUNC], symtab);
        while uprocs <> NIL do
          begin
            resolve(uprocs, procs);
```

```
                sp := seplist;
                while sp <> NIL do
                  begin
                    refsrch([GLOBREF, SEPPREF, SEPFREF], sp);
                    sp := sp^.next
                  end;
                resolve(uother, other);
                findnewprocs
              end;
            if not sephost then
              begin
                findlocals(symtab);
                resolve(ulocal, local)
              end;
            wp := procs;
            while wp <> NIL do
              begin
                wp^.needsrch := TRUE;
                wp := wp^.next
              end;
            sp := seplist;
            while sp <> NIL do
              begin
                refsrch([PUBLREF, PRIVREF, CONSTREF], sp);
                sp := sp^.next
              end;
            resolve(uother, other)
          end
      end { buildworklists } ;

  { $I link3b }

      (*****************************************************************)
      (*                                                               *)
      (*  Copyright (c) 1978 Regents of the University of California.  *)
      (*  Permission to copy or distribute this software or documen-   *)
      (*  tation in hard or soft copy granted only by written license  *)
      (*  obtained from the Institute for Information Systems.         *)
      (*                                                               *)
      (*****************************************************************)

      {
      *  Readsrcseg determines the final segment size after adding
      *  in the external procs/funcs, allocates enough area for the
      *  entire output code seg, reads in the original code (or uses
      *  identity segment for sephost special case), and splits the
      *  segdict off from the code.  For all procs to-be-linked, a new
      *  destbase position is assigned in seg and the new proc num is
      *  set up in pdict.  The segment number field of the pdict is
      *  also updated to the value of s.  All is ready to copy in the
      *  sep procs/funcs.  The values for segbase and segleng are set
      *  here too.
      }

      procedure readsrcseg;
        var orgleng, addr,
            addleng, addprocs,
            nextspot: integer;
            last: 0..MAXPROC;
            wp: workp;
```

```
  lheap: ^integer;

{
*  Readnsplit arranges for the source seg to be placed in
*  room allocated for segbase.  This may involve disk read
*  or perhaps only creating an empty segment.  In any case
*  segbase points at lowest addr, and nextspot is pointed
*  at the next place code can be copied into.  This is used
*  for destbase assignment in readsrcseg.
}

procedure readnsplit;
  var nblocks, n, pdleng,
      pddelta, nprocs: integer;
      cp0, cp1: codep;
begin
  nblocks := (segleng+511) div 512;
  if memavail-400 < nblocks*256 then
    begin
      error('no mem room');
      exit(linker)
    end;
  n := nblocks;
  repeat
    { alloc heap space }
    new(cp1);
    n := n-1
  until n <= 0;
  if sephost then
    begin { set up identity seg }
      storeword(0, segbase, segleng-2);
      nextspot := 0
    end
  else
    begin { read from disk }
      nblocks := (orgleng+511) div 512;
      if blockread(seginfo[s]^.srcfile^.code^, segbase^,
                nblocks, addr) <> nblocks then
        begin
          error('seg read err');
          exit(linker)
        end;
      pddelta := segleng-orgleng;
      nprocs := fetchbyte(segbase, orgleng-1);
      pdleng := nprocs*2+2;
      nextspot := orgleng-pdleng;
      cp0 := getcodep(ord(segbase)+orgleng-pdleng);
      cp1 := getcodep(ord(segbase)+segleng-pdleng);
      if cp0 <> cp1 then
        begin { move proc dict }
          n := pdleng;
          while n > 2 do
            begin
              storeword(pddelta+fetchword(segbase, orgleng-n),
                  segbase, orgleng-n);
              n := n-2
            end;
          moveright(cp0^, cp1^, pdleng);
          fillchar(cp0^, pddelta, 0)
        end
```

```
        end
    end { readnsplit } ;

begin { readsrcseg }
  if sephost then
    orgleng := 2
  else
    with seginfo[s]^, srcfile^.segtbl.diskinfo[srcseg] do
      begin
        orgleng := codeleng;
        addr := codeaddr
      end;
  addleng := 0;
  addprocs := 0;
  wp := procs;
  while wp <> NIL do
    begin { add up final seg size }
      addleng := addleng+wp^.defsym^.entry.place^.length;
      if wp^.newproc = 0 then
        addprocs := addprocs+1;
      wp := wp^.next
    end;
  mark(lheap);
  segbase := getcodep(ord(lheap));
  segleng := orgleng+addleng+2*addprocs;
  if segleng <= 0 then
    begin
      error('size oflow');
      exit(linker)
    end;
  readnsplit;
  last := fetchbyte(segbase, segleng-1);
  wp := procs;
  while wp <> NIL do
    begin { assign places in code seg }
      with wp^.defsym^.entry.place^ do
        begin
          destbase := nextspot;
          nextspot := nextspot+length
        end;
      if wp^.newproc = 0 then
        begin { assign new proc # }
          last := last+1;
          if last > MAXPROC then
            begin
              error('proc num oflow');
              last := 1
            end;
          wp^.newproc := last
        end;
      wp := wp^.next
    end;
  storebyte(last, segbase, segleng-1);
  storebyte(s, segbase, segleng-2)
end { readsrcseg } ;

{
*  Copyinprocs goes through procs list and copies procedure
*  bodies from the sep segs into the dest code segment into
*  locations set up in readsrcseg. If all goes right, we should
```

```
*  fill dest seg to the exact byte.  The proc dict is
*  updated to show procedures' position.
}

procedure copyinprocs;
  var cp0, cp1, pdp,
      jtab, sepbase: codep;
      wp: workp;
      cursp: segp;
      lheap: ^integer;

    {
    *  Readsepseg reads the sep seg in sp onto the heap as
    *  done in Phase 2.  We set up sepbase and cursp for
    *  copyinprocs.
    }

    procedure readsepseg(sp: segp);
      var n, nblocks: integer;
    begin
      release(lheap);
      n := sp^.srcfile^.segtbl.diskinfo[sp^.srcseg].codeleng;
      nblocks := (n+511) div 512;
      if memavail-400 < nblocks*256 then
        begin
          error('out of mem');
          exit(linker)
        end;
      n := nblocks;
      repeat
        new(sepbase);
        n := n-1
      until n <= 0;
      sepbase := getcodep(ord(lheap));
      if blockread(sp^.srcfile^.code^, sepbase^, nblocks,
          sp^.srcfile^.segtbl.diskinfo[sp^.srcseg].codeaddr) <> nblocks then
        begin
          error('sep seg read err');
          exit(linker)
        end;
      cursp := sp
    end { readsepseg } ;

begin { copyinprocs }
  sepbase := NIL;
  cursp := NIL;
  mark(lheap);
  wp := procs;
  while wp <> NIL do
    with wp^, defsym^.entry do
      begin { copy in each proc }
        if cursp <> defseg then
          readsepseg(defseg);
        if talkative then
          begin
            write('   Copying ');
            if litype = SEPPROC then
              write('proc ')
            else
              write('func ');
```

```
            writeln(name)
          end;
        cp0 := getcodep(ord(sepbase)+place^.srcbase);
        cp1 := getcodep(ord(segbase)+place^.destbase);
        moveleft(cp0^, cp1^, place^.length);
        jtab := getcodep(ord(segbase)+place^.destbase+place^.length-2);
        if fetchbyte(jtab, 0) <> 0 then
          storebyte(newproc, jtab, 0);
        pdp := getcodep(ord(segbase)+segleng-2*newproc-2);
        storeword(ord(pdp)-ord(jtab), pdp, 0);
        wp := next
      end;
  release(lheap)
end { copyinprocs } ;

{
*  Fixuprefs is called to search through reflists and fix
*  operand fields of P-code and native code to refer to the
*  resolved values.  If fixallrefs is true, then all pointers
*  in the ref lists are used, otherwise the reference pointers
*  are checked to see if they occur in the procs to-be-linked.
}

procedure fixuprefs(work: workp; fixallrefs: boolean);
  var n, i, ref, val: integer;
      wp, wp1: workp;
      rp: refp;
      skipit: boolean;
      r: packed record
           case boolean of
             TRUE:  (integ: integer);
             FALSE: (lowbyte: 0..255;
                     highbyte: 0..255)
         end { r } ;
begin
  while work <> NIL do
    with work^, refsym^.entry do
      begin { for each work item }
            { figure resolve val }
        case litype of
          SEPPREF,
          SEPFREF:  val := defproc^.newproc;
          UNITREF:  val := defsegnum;
          CONSTREF: val := defsym^.entry.constval;
          GLOBREF:  val := defsym^.entry.icoffset+
                           defproc^.defsym^.entry.place^.destbase;
          PUBLREF,
          PRIVREF:  begin
                      if litype = PRIVREF then
                        val := newoffset
                      else
                        val := defsym^.entry.baseoffset;
                      if format = WORD then
                        val := (val-1)*2+MSDELTA
                      else { assume BIG }
                        if val >= 0 then
                          begin
                            r.highbyte := val mod 256;
                            r.lowbyte := val div 256 + 128;
                            val := r.integ
```

```
                    end
                  else
                    error('addr oflow')
                end
          end;
          n := nrefs;
          rp := reflist;
          while rp <> NIL do
            begin
              if n > 8 then
                begin
                  i := 7;
                  n := n-8
                end
              else
                i := n-1;
              repeat
                ref := rp^.refs[i];
                skipit := not fixallrefs;
                if skipit then
                  begin { see if pertinent }
                    wp := NIL;
                    wp1 := procs;
                    while wp1 <> NIL do
                      if wp1^.defseg = refseg then
                        begin { find matching seg }
                          wp := wp1;
                          wp1 := NIL
                        end
                      else
                        wp1 := wp1^.next;
                    while (wp <> NIL) and skipit do
                      if wp^.defseg = refseg then
                        with wp^.defsym^.entry.place^ do
                          if ref >= srcbase then
                            if ref < srcbase+length then
                              begin
                                ref := ref-srcbase+destbase;
                                skipit := FALSE
                              end
                            else
                              wp := wp^.next
                          else
                            wp := NIL
                      else
                        wp := NIL
                  end;
                if not skipit then
                  case format of { fix up this ref }
                    WORD:  storeword(val+fetchword(segbase, ref),
                                                   segbase, ref);
                    BYTE:  storebyte(val, segbase, ref);
                    BIG:   storeword(val, segbase, ref)
                  end;
                i := i-1
              until i < 0;
              rp := rp^.next
            end;
          work := next
        end
```

```
end { fixuprefs } ;

{
*  writetocode takes the finalized destseg and puts it in
*  the output code file.  This also involves setting up values
*  in the final segtable for writeout just before locking it.
}

procedure writetocode;
  var nblocks: integer;
      jtab: codep;
begin
  if hostsp = seginfo[s] then
    begin { fix up baselc }
      jtab := getcodep(ord(segbase)+segleng-4);
      jtab := getcodep(ord(jtab)-fetchword(jtab, 0));
      storeword(nextbaselc*2-6, jtab, -8)
    end;
  with seginfo[s]^, segtbl do
    begin
      nblocks := (segleng+511) div 512;
      if blockwrite(code, segbase^, nblocks, nextblk) <> nblocks then
        begin
          error('code write err');
          exit(linker)
        end;
      diskinfo[s].codeaddr := nextblk;
      diskinfo[s].codeleng := segleng;
      segname[s] := srcfile^.segtbl.segname[srcseg];
      segkind[s] := LINKED;
      nextblk := nextblk+nblocks
    end
end { writetocode } ;

{
*  Linksegment is called for each segment to be placed into
*  the final code file.  The global var s has the seginfo index
*  pertaining to the segment, and all the other procedures of
*  Phase 3 are called from here.  This proc facilitates linking
*  the master seg separatly from the other segs to ensure that
*  the DATASZ of the outer block correctly reflects the number
*  of PRIVREF words allocated by resolve.
}

procedure linksegment;

    {
    *  Writemap is called for each seg to write some
    *  info into map file.
    }

    procedure writemap;
      var wp: workp;
          b: boolean;
    begin
      with seginfo[s]^ do
        writeln(map, 'Seg # ',s,', ', srcfile^.segtbl.segname[srcseg]);
      wp := procs;
      if wp <> NIL then
        writeln(map, '   Sep procs');
```

```
        while wp <> NIL do
          with wp^.defsym^.entry do
            begin
              write(map, '      ', name);
              if litype = SEPPROC then
                write(map, ' proc')
              else
                write(map, ' func');
              write(map, ' # ', wp^.newproc: 3);
              write(map, '    base =', place^.destbase: 6);
              write(map, '    leng =', place^.length: 5);
              writeln(map);
              wp := wp^.next
            end;
        for b := FALSE to TRUE do
          begin
            if b then
              begin
                wp := other;
                if wp <> NIL then
                  writeln(map, '   Sep proc refs')
              end
            else
              begin
                wp := local;
                if wp <> NIL then
                  writeln(map, '   Local seg refs')
              end;
            while wp <> NIL do
              with wp^.defsym^.entry do
                begin
                  write(map, '      ', name);
                  case litype of
                    SEPPROC,
                    SEPFUNC:   ;
                    PUBLDEF:   write(map, ' public LC =', baseoffset: 5);
                    CONSTDEF:  write(map, ' const val =', constval: 6);
                    PRIVREF:   write(map, ' privat LC =', wp^.newoffset: 5);
                    UNITREF:   write(map, ' unit seg# =', wp^.defsegnum: 3);
                    GLOBDEF:   write(map, ' glob def in ',
                                  wp^.defproc^.defsym^.entry.name,
                                  ' @', icoffset: 5)
                  end;
                  writeln(map);
                  wp := wp^.next
                end
          end;
        writeln(map)
      end { writemap } ;

    begin { linksegment }
      sephost := FALSE;
      segbase := NIL;
      segleng := 0;
      if talkative then
        with seginfo[s]^ do
          writeln('Linking ',
                   srcfile^.segtbl.segname[srcseg], ' # ', s);
      buildworklists;
      if errcount = 0 then
```

```
        begin
          readsrcseg;
          if mapname <> '' then
            writemap;
          copyinprocs;
          fixuprefs(local, TRUE);
          fixuprefs(other, FALSE);
          writetocode
        end;
      if sephost then
        seplist := seginfo[s]^.next;
      release(heapbase)
    end { linksegment } ;

begin { phase3 }
  if not useworkfile then
    begin
      write('Output file? ');
      readln(fname);
      useworkfile := fname = ''
    end;
  if useworkfile then
    rewrite(code, '*SYSTEM.WRK.CODE[*]')
  else
    rewrite(code, fname);
  if IORESULT <> 0 then
    begin
      error('Code open err');
      exit(linker)
    end;
  nextblk := 1;
  { clear output seg table }
  fillchar(segtbl, sizeof(segtbl), 0);
  with segtbl do
    for s := 0 to MAXSEG do
      begin
        segname[s] := '        ';
        segkind[s] := LINKED
      end;
  if mapname <> '' then
    begin
      rewrite(map, mapname);
      if IORESULT <> 0 then
        begin
          writeln('Can''t open ', mapname);
          mapname := ''
        end
      else
        begin
          write(map, 'Link map for ');
          if hostsp <> NIL then
            writeln(map, hostsp^.srcfile^.segtbl.segname[hostsp^.srcseg])
          else
            writeln(map, 'assem host');
          writeln(map)
        end
    end;
  mark(heapbase);
  unitwrite(3, heapbase^, 35);
  { link all but host }
```

```
   for s := 0 to MAXSEG do
     if (seginfo[s] <> NIL)
     and (seginfo[s] <> hostsp) then
       linksegment;
   { link host last! }
   if hostsp <> NIL then
     begin
       s := MASTERSEG;
       linksegment
     end;
   if blockwrite(code, segtbl, 1, 0) <> 1 then
     error('Code write err');
   if errcount = 0 then
     begin { final cleanup }
       close(code, LOCK);
       if useworkfile then
         with userinfo do
           begin
             gotcode := TRUE;
             codevid := syvid;
             codetid := 'SYSTEM.WRK.CODE'
           end;
       if mapname <> '' then
         begin
           if hostsp <> NIL then
             writeln(map, 'next base LC = ', nextbaselc);
           close(map, LOCK)
         end
     end

 end { phase3 } ;


 begin { linker }
   phase1;
   phase2;
   phase3;
   unitclear(3)
 end { linker } ;

 begin end.

{ +-------------------------------------------------------------------+
  |                                                                   |
  |                     F     I     N     I     S                     |
  |                                                                   |
  +-------------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 Linker
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 PIO
##########################################################################

 (*$U-*)
 (*$S+*)

 (* $I GLOBALS.TEXT*)

 (*$U-,S+*)

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

 PROGRAM PASCALSYSTEM;

 (*************************************************)
 (*                                               *)
 (*    UCSD PASCAL OPERATING SYSTEM               *)
 (*                                               *)
 (*    RELEASE LEVEL:  I.3   AUGUST, 1977         *)
 (*                      I.4   JANUARY, 1978      *)
 (*                    I.5   SEPTEMBER, 1978      *)
 (*                                               *)
 (*    WRITTEN BY ROGER T. SUMNER                 *)
 (*    WINTER 1977                                *)
 (*                                               *)
 (*    INSTITUTE FOR INFORMATION SYSTEMS          *)
 (*    UC SAN DIEGO, LA JOLLA, CA                 *)
 (*                                               *)
 (*    KENNETH L. BOWLES, DIRECTOR                *)
 (*                                               *)
 (*************************************************)

 CONST
      MMAXINT = 32767;   (*MAXIMUM INTEGER VALUE*)
      MAXUNIT = 12;      (*MAXIMUM PHYSICAL UNIT # FOR UREAD*)
      MAXDIR = 77;       (*MAX NUMBER OF ENTRIES IN A DIRECTORY*)
      VIDLENG = 7;       (*NUMBER OF CHARS IN A VOLUME ID*)
      TIDLENG = 15;      (*NUMBER OF CHARS IN TITLE ID*)
      MAXSEG = 15;       (*MAX CODE SEGMENT NUMBER*)
      FBLKSIZE = 512;    (*STANDARD DISK BLOCK LENGTH*)
      DIRBLK = 2;        (*DISK ADDR OF DIRECTORY*)
      AGELIMIT = 300;    (*MAX AGE FOR GDIRP...IN TICKS*)
      EOL = 13;          (*END-OF-LINE...ASCII CR*)
      DLE = 16;          (*BLANK COMPRESSION CODE*)

 TYPE

      IORSLTWD = (INOERROR,IBADBLOCK,IBADUNIT,IBADMODE,ITIMEOUT,
                  ILOSTUNIT,ILOSTFILE,IBADTITLE,INOROOM,INOUNIT,
                  INOFILE,IDUPFILE,INOTCLOSED,INOTOPEN,IBADFORMAT,
                  ISTRGOVFL);
```

```
                                    (*COMMAND STATES...SEE GETCMD*)

CMDSTATE = (HALTINIT,DEBUGCALL,
            UPROGNOU,UPROGUOK,SYSPROG,
            COMPONLY,COMPANDGO,COMPDEBUG,
            LINKANDGO,LINKDEBUG);

                                    (*CODE FILES USED IN GETCMD*)

SYSFILE = (ASSMBLER,COMPILER,EDITOR,FILER,LINKER);

                                    (*ARCHIVAL INFO...THE DATE*)

DATEREC = PACKED RECORD
            MONTH: 0..12;           (*0 IMPLIES DATE NOT MEANINGFUL*)
            DAY: 0..31;             (*DAY OF MONTH*)
            YEAR: 0..100            (*100 IS TEMP DISK FLAG*)
          END (*DATEREC*) ;

                                    (*VOLUME TABLES*)
UNITNUM = 0..MAXUNIT;
VID = STRING[VIDLENG];

                                    (*DISK DIRECTORIES*)
DIRRANGE = 0..MAXDIR;
TID = STRING[TIDLENG];

FILEKIND = (UNTYPEDFILE,XDSKFILE,CODEFILE,TEXTFILE,
            INFOFILE,DATAFILE,GRAFFILE,FOTOFILE,SECUREDIR);

DIRENTRY = RECORD
             DFIRSTBLK: INTEGER;    (*FIRST PHYSICAL DISK ADDR*)
             DLASTBLK: INTEGER;     (*POINTS AT BLOCK FOLLOWING*)
             CASE DFKIND: FILEKIND OF
               SECUREDIR,
               UNTYPEDFILE: (*ONLY IN DIR[0]...VOLUME INFO*)
                  (DVID: VID;               (*NAME OF DISK VOLUME*)
                   DEOVBLK: INTEGER;        (*LASTBLK OF VOLUME*)
                   DNUMFILES: DIRRANGE;     (*NUM FILES IN DIR*)
                   DLOADTIME: INTEGER;      (*TIME OF LAST ACCESS*)
                   DLASTBOOT: DATEREC);     (*MOST RECENT DATE SETTING*)
               XDSKFILE,CODEFILE,TEXTFILE,INFOFILE,
               DATAFILE,GRAFFILE,FOTOFILE:
                  (DTID: TID;               (*TITLE OF FILE*)
                   DLASTBYTE: 1..FBLKSIZE;  (*NUM BYTES IN LAST BLOCK*)
                   DACCESS: DATEREC)        (*LAST MODIFICATION DATE*)
           END (*DIRENTRY*) ;

DIRP = ^DIRECTORY;

DIRECTORY = ARRAY [DIRRANGE] OF DIRENTRY;

                                    (*FILE INFORMATION*)

CLOSETYPE = (CNORMAL,CLOCK,CPURGE,CCRUNCH);
WINDOWP = ^WINDOW;
WINDOW = PACKED ARRAY [0..0] OF CHAR;

FIBP = ^FIB;
```

```
FIB = RECORD
        FWINDOW: WINDOWP;  (*USER WINDOW...F^, USED BY GET-PUT*)
        FEOF,FEOLN: BOOLEAN;
        FSTATE: (FJANDW,FNEEDCHAR,FGOTCHAR);
        FRECSIZE: INTEGER; (*IN BYTES...0=>BLOCKFILE, 1=>CHARFILE*)
        CASE FISOPEN: BOOLEAN OF
          TRUE: (FISBLKD: BOOLEAN; (*FILE IS ON BLOCK DEVICE*)
                 FUNIT: UNITNUM;   (*PHYSICAL UNIT #*)
                 FVID: VID;        (*VOLUME NAME*)
                 FREPTCNT,         (* # TIMES F^ VALID W/O GET*)
                 FNXTBLK,          (*NEXT REL BLOCK TO IO*)
                 FMAXBLK: INTEGER; (*MAX REL BLOCK ACCESSED*)
                 FMODIFIED:BOOLEAN;(*PLEASE SET NEW DATE IN CLOSE*)
                 FHEADER: DIRENTRY;(*COPY OF DISK DIR ENTRY*)
                 CASE FSOFTBUF: BOOLEAN OF (*DISK GET-PUT STUFF*)
                   TRUE: (FNXTBYTE,FMAXBYTE: INTEGER;
                          FBUFCHNGD: BOOLEAN;
                          FBUFFER: PACKED ARRAY [0..FBLKSIZE] OF CHAR))
      END (*FIB*) ;

                                  (*USER WORKFILE STUFF*)

INFOREC = RECORD
            SYMFIBP,CODEFIBP: FIBP;        (*WORKFILES FOR SCRATCH*)
            ERRSYM,ERRBLK,ERRNUM: INTEGER; (*ERROR STUFF IN EDIT*)
            SLOWTERM,STUPID: BOOLEAN;      (*STUDENT PROGRAMMER ID!!*)
            ALTMODE: CHAR;                 (*WASHOUT CHAR FOR COMPILER*)
            GOTSYM,GOTCODE: BOOLEAN;       (*TITLES ARE MEANINGFUL*)
            WORKVID,SYMVID,CODEVID: VID;   (*PERM&CUR WORKFILE VOLUMES*)
            WORKTID,SYMTID,CODETID: TID    (*PERM&CUR WORKFILES TITLE*)
          END (*INFOREC*) ;

                                  (*CODE SEGMENT LAYOUTS*)

SEGRANGE = 0..MAXSEG;
SEGDESC = RECORD
            DISKADDR: INTEGER;     (*REL BLK IN CODE...ABS IN SYSCOM^*)
            CODELENG: INTEGER      (*# BYTES TO READ IN*)
          END (*SEGDESC*) ;

                                  (*DEBUGGER STUFF*)

BYTERANGE = 0..255;
TRICKARRAY = ARRAY [0..0] OF INTEGER; (* FOR MEMORY DIDDLING*)
MSCWP = ^ MSCW;            (*MARK STACK RECORD POINTER*)
MSCW = RECORD
         STATLINK: MSCWP;  (*POINTER TO PARENT MSCW*)
         DYNLINK: MSCWP;   (*POINTER TO CALLER'S MSCW*)
         MSSEG,MSJTAB: ^TRICKARRAY;
         MSIPC: INTEGER;
         LOCALDATA: TRICKARRAY
       END (*MSCW*) ;

                                  (*SYSTEM COMMUNICATION AREA*)
                                  (*SEE INTERPRETERS...NOTE  *)
                                  (*THAT WE ASSUME BACKWARD  *)
                                  (*FIELD ALLOCATION IS DONE *)
SYSCOMREC = RECORD
              IORSLT: IORSLTWD;    (*RESULT OF LAST IO CALL*)
```

```
                    XEQERR: INTEGER;     (*REASON FOR EXECERROR CALL*)
                    SYSUNIT: UNITNUM;    (*PHYSICAL UNIT OF BOOTLOAD*)
                    BUGSTATE: INTEGER;   (*DEBUGGER INFO*)
                    GDIRP: DIRP;         (*GLOBAL DIR POINTER,SEE VOLSEARCH*)
                    LASTMP,STKBASE,BOMBP: MSCWP;
                    MEMTOP,SEG,JTAB: INTEGER;
                    BOMBIPC: INTEGER;    (*WHERE XEQERR BLOWUP WAS*)
                    HLTLINE: INTEGER;    (*MORE DEBUGGER STUFF*)
                    BRKPTS: ARRAY [0..3] OF INTEGER;
                    RETRIES: INTEGER;    (*DRIVERS PUT RETRY COUNTS*)
                    EXPANSION: ARRAY [0..8] OF INTEGER;
                    HIGHTIME,LOWTIME: INTEGER;
                    MISCINFO: PACKED RECORD
                                NOBREAK,STUPID,SLOWTERM,
                                HASXYCRT,HASLCCRT,HAS8510A,HASCLOCK: BOOLEAN;
                                USERKIND:(NORMAL, AQUIZ, BOOKER, PQUIZ)
                              END;
                    CRTTYPE: INTEGER;
                    CRTCTRL: PACKED RECORD
                                RLF,NDFS,ERASEEOL,ERASEEOS,HOME,ESCAPE: CHAR;
                                BACKSPACE: CHAR;
                                FILLCOUNT: 0..255;
                                CLEARSCREEN, CLEARLINE: CHAR;
                                PREFIXED: PACKED ARRAY [0..8] OF BOOLEAN
                             END;
                    CRTINFO: PACKED RECORD
                                WIDTH,HEIGHT: INTEGER;
                                RIGHT,LEFT,DOWN,UP: CHAR;
                                BADCH,CHARDEL,STOP,BREAK,FLUSH,EOF: CHAR;
                                ALTMODE,LINEDEL: CHAR;
                                BACKSPACE,ETX,PREFIX: CHAR;
                                PREFIXED: PACKED ARRAY [0..13] OF BOOLEAN
                             END;
                    SEGTABLE: ARRAY [SEGRANGE] OF
                                RECORD
                                  CODEUNIT: UNITNUM;
                                  CODEDESC: SEGDESC
                                END
                  END (*SYSCOM*);

      MISCINFOREC = RECORD
                      MSYSCOM: SYSCOMREC
                    END;

VAR
    SYSCOM: ^SYSCOMREC;                 (*MAGIC PARAM...SET UP IN BOOT*)
    GFILES: ARRAY [0..5] OF FIBP;       (*GLOBAL FILES, 0=INPUT, 1=OUTPUT*)
    USERINFO: INFOREC;                  (*WORK STUFF FOR COMPILER ETC*)
    EMPTYHEAP: ^INTEGER;                (*HEAP MARK FOR MEM MANAGING*)
    INPUTFIB,OUTPUTFIB,                 (*CONSOLE FILES...GFILES ARE COPIES*)
    SYSTERM,SWAPFIB: FIBP;              (*CONTROL AND SWAPSPACE FILES*)
    SYVID,DKVID: VID;                   (*SYSUNIT VOLID & DEFAULT VOLID*)
    THEDATE: DATEREC;                   (*TODAY...SET IN FILER OR SIGN ON*)
    DEBUGINFO: ^INTEGER;                (*DEBUGGERS GLOBAL INFO WHILE RUNIN*)
    STATE: CMDSTATE;                    (*FOR GETCOMMAND*)
    PL: STRING;                         (*PROMPTLINE STRING...SEE PROMPT*)
    IPOT: ARRAY [0..4] OF INTEGER;      (*INTEGER POWERS OF TEN*)
    FILLER: STRING[11];                 (*NULLS FOR CARRIAGE DELAY*)
    DIGITS: SET OF '0'..'9';
    UNITABLE: ARRAY [UNITNUM] OF (*0 NOT USED*)
```

```
                RECORD
                  UVID: VID;    (*VOLUME ID FOR UNIT*)
                  CASE UISBLKD: BOOLEAN OF
                    TRUE: (UEOVBLK: INTEGER)
                END (*UNITABLE*) ;
    FILENAME: ARRAY [SYSFILE] OF STRING[23];

(*-------------------------------------------------------------------------*)
(* SYSTEM PROCEDURE FORWARD DECLARATIONS *)
(* THESE ARE ADDRESSED BY OBJECT CODE... *)
(*  DO NOT MOVE WITHOUT CAREFUL THOUGHT  *)

PROCEDURE EXECERROR;
  FORWARD;
PROCEDURE FINIT(VAR F: FIB; WINDOW: WINDOWP; RECWORDS: INTEGER);
  FORWARD;
PROCEDURE FRESET(VAR F: FIB);
  FORWARD;
PROCEDURE FOPEN(VAR F: FIB; VAR FTITLE: STRING;
                FOPENOLD: BOOLEAN; JUNK: FIBP);
  FORWARD;
PROCEDURE FCLOSE(VAR F: FIB; FTYPE: CLOSETYPE);
  FORWARD;
PROCEDURE FGET(VAR F: FIB);
  FORWARD;
PROCEDURE FPUT(VAR F: FIB);
  FORWARD;
PROCEDURE XSEEK;
  FORWARD;
FUNCTION FEOF(VAR F: FIB): BOOLEAN;
  FORWARD;
FUNCTION FEOLN(VAR F: FIB): BOOLEAN;
  FORWARD;
PROCEDURE FREADINT(VAR F: FIB; VAR I: INTEGER);
  FORWARD;
PROCEDURE FWRITEINT(VAR F: FIB; I,RLENG: INTEGER);
  FORWARD;
PROCEDURE XREADREAL;
  FORWARD;
PROCEDURE XWRITEREAL;
  FORWARD;
PROCEDURE FREADCHAR(VAR F: FIB; VAR CH: CHAR);
  FORWARD;
PROCEDURE FWRITECHAR(VAR F: FIB; CH: CHAR; RLENG: INTEGER);
  FORWARD;
PROCEDURE FREADSTRING(VAR F: FIB; VAR S: STRING; SLENG: INTEGER);
  FORWARD;
PROCEDURE FWRITESTRING(VAR F: FIB; VAR S: STRING; RLENG: INTEGER);
  FORWARD;
PROCEDURE FWRITEBYTES(VAR F: FIB; VAR A: WINDOW; RLENG,ALENG: INTEGER);
  FORWARD;
PROCEDURE FREADLN(VAR F: FIB);
  FORWARD;
PROCEDURE FWRITELN(VAR F: FIB);
  FORWARD;
PROCEDURE SCONCAT(VAR DEST,SRC: STRING; DESTLENG: INTEGER);
  FORWARD;
PROCEDURE SINSERT(VAR SRC,DEST: STRING; DESTLENG,INSINX: INTEGER);
  FORWARD;
PROCEDURE SCOPY(VAR SRC,DEST: STRING; SRCINX,COPYLENG: INTEGER);
```

```
    FORWARD;
  PROCEDURE SDELETE(VAR DEST: STRING; DELINX,DELLENG: INTEGER);
    FORWARD;
  FUNCTION SPOS(VAR TARGET,SRC: STRING): INTEGER;
    FORWARD;
  FUNCTION FBLOCKIO(VAR F: FIB; VAR A: WINDOW;
                    NBLOCKS,RBLOCK: INTEGER; DOREAD: BOOLEAN): INTEGER;
    FORWARD;
  PROCEDURE FGOTOXY(X,Y: INTEGER);
    FORWARD;

  (* NON FIXED FORWARD DECLARATIONS *)

  FUNCTION VOLSEARCH(VAR FVID: VID; LOOKHARD: BOOLEAN;
                     VAR FDIR: DIRP): UNITNUM;
    FORWARD;
  PROCEDURE WRITEDIR(FUNIT: UNITNUM; FDIR: DIRP);
    FORWARD;
  FUNCTION DIRSEARCH(VAR FTID: TID; FINDPERM: BOOLEAN; FDIR: DIRP): DIRRANGE;
    FORWARD;
  FUNCTION SCANTITLE(FTITLE: STRING; VAR FVID: VID; VAR FTID: TID;
                     VAR FSEGS: INTEGER; VAR FKIND: FILEKIND): BOOLEAN;
    FORWARD;
  PROCEDURE DELENTRY(FINX: DIRRANGE; FDIR: DIRP);
    FORWARD;
  PROCEDURE INSENTRY(VAR FENTRY: DIRENTRY; FINX: DIRRANGE; FDIR: DIRP);
    FORWARD;
  PROCEDURE HOMECURSOR;
    FORWARD;
  PROCEDURE CLEARSCREEN;
    FORWARD;
  PROCEDURE CLEARLINE;
    FORWARD;
  PROCEDURE PROMPT;
    FORWARD;
  FUNCTION SPACEWAIT(FLUSH: BOOLEAN): BOOLEAN;
    FORWARD;
  FUNCTION GETCHAR(FLUSH: BOOLEAN): CHAR;
    FORWARD;
  PROCEDURE COMMAND;
    FORWARD;

  (*Dummy level 0 outerblock*)
  BEGIN END.

{ +-----------------------------------------------------------------+
  |                                                                 |
  |                     F     I     N     I     S                   |
  |                                                                 |
  +-----------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 PIO
```

```
########################################################################
### FILE: UCSD Pascal 1.5 PIO Unit
########################################################################

 (* UCSD PASCAL I.5 PASCAL I/O UNIT *)
 (*-------------------------------------------------------*)

 SEPARATE UNIT PASCALIO;
 INTERFACE

   TYPE DECMAX = INTEGER[36];

   PROCEDURE FSEEK(VAR F: FIB; RECNUM: INTEGER);
   PROCEDURE FREADREAL(VAR F: FIB; VAR X: REAL);
   PROCEDURE FWRITEREAL(VAR F: FIB; X: REAL; W, D: INTEGER);
   PROCEDURE FREADDEC(VAR F: FIB; VAR D: TRICKARRAY; L: INTEGER);
   PROCEDURE FWRITEDEC(VAR F: FIB; D: DECMAX; RLENG: INTEGER);

 IMPLEMENTATION

   PROCEDURE FSEEK(*VAR F: FIB; RECNUM: INTEGER*);
     LABEL 1;
     VAR BYTE,BLOCK,N: INTEGER;
   BEGIN SYSCOM^.IORSLT := INOERROR;
     IF F.FISOPEN THEN
       WITH F,FHEADER DO
         BEGIN BLOCK := 0; BYTE := FBLKSIZE;
           IF (RECNUM < 0) OR NOT FSOFTBUF OR
                   ((DFKIND = TEXTFILE) AND (FRECSIZE = 1)) THEN
             GOTO 1; (*NO SEEK ALLOWED*)
           IF FRECSIZE < FBLKSIZE THEN
             BEGIN N := FBLKSIZE DIV FRECSIZE;
               WHILE RECNUM-N >= 0 DO
                 BEGIN RECNUM := RECNUM-N;
                   BYTE := BYTE+N*FRECSIZE;
                   WHILE BYTE > FBLKSIZE DO
                     BEGIN BLOCK := BLOCK+1;
                       BYTE := BYTE-FBLKSIZE
                     END
                 END
             END;
           WHILE RECNUM > 0 DO
             BEGIN RECNUM := RECNUM-1;
               BYTE := BYTE+FRECSIZE;
               WHILE BYTE > FBLKSIZE DO
                 BEGIN BLOCK := BLOCK+1;
                   BYTE := BYTE-FBLKSIZE
                 END
             END;
           N := DLASTBLK-DFIRSTBLK;
           IF (BLOCK > N) OR ((BLOCK = N) AND (BYTE >= DLASTBYTE)) THEN
             BEGIN BLOCK := N; BYTE := DLASTBYTE END;
           IF BLOCK <> FNXTBLK THEN
             BEGIN
               IF FBUFCHNGD THEN
                 BEGIN FBUFCHNGD := FALSE; FMODIFIED := TRUE;
                   UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+FNXTBLK-1);
                   IF IORESULT <> ORD(INOERROR) THEN GOTO 1
                 END;
```

```
            IF (BLOCK <= FMAXBLK) AND (BYTE <> FBLKSIZE) THEN
              BEGIN
                UNITREAD(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+BLOCK-1);
                IF IORESULT <> ORD(INOERROR) THEN GOTO 1
              END
          END;
        IF FNXTBLK > FMAXBLK THEN
          BEGIN FMAXBLK := FNXTBLK; FMAXBYTE := FNXTBYTE END
        ELSE
          IF (FNXTBLK = FMAXBLK) AND (FNXTBYTE > FMAXBYTE) THEN
            FMAXBYTE := FNXTBYTE;
        FEOF := FALSE; FEOLN := FALSE; FREPTCNT := 0;
        IF FSTATE <> FJANDW THEN FSTATE := FNEEDCHAR;
        FNXTBLK := BLOCK; FNXTBYTE := BYTE
      END
  ELSE SYSCOM^.IORSLT := INOTOPEN;
1:
END (*FSEEK*) ;

PROCEDURE FREADREAL(*VAR F: FIB; VAR X: REAL*);
  LABEL 1;
  VAR CH: CHAR; NEG,XVALID: BOOLEAN; IPOT: INTEGER;
BEGIN
  WITH F DO
      BEGIN X := 0; NEG := FALSE; XVALID := FALSE;
        IF FSTATE = FNEEDCHAR THEN FGET(F);
        WHILE (FWINDOW^[0] = ' ') AND NOT FEOF DO FGET(F);
        IF FEOF THEN GOTO 1;
        CH := FWINDOW^[0];
        IF (CH = '+') OR (CH = '-') THEN
          BEGIN NEG := CH = '-'; FGET(F); CH := FWINDOW^[0] END;
        WHILE (CH IN DIGITS) AND NOT FEOF DO
          BEGIN XVALID := TRUE;
            X := X*10 + (ORD(CH)-ORD('0'));
            FGET(F); CH := FWINDOW^[0]
          END;
        IF FEOF THEN GOTO 1;
        IPOT := -1;
        IF CH = '.' THEN
          BEGIN IPOT := 0;
            REPEAT FGET(F); CH := FWINDOW^[0];
              IF CH IN DIGITS THEN
                BEGIN XVALID := TRUE; IPOT := IPOT + 1;
                  X := X + (ORD(CH)-ORD('0'))/PWROFTEN(IPOT)
                END
            UNTIL FEOF OR NOT (CH IN DIGITS);
            IF FEOF THEN GOTO 1
          END;
        IF ((CH = 'e') OR (CH = 'E')) AND (XVALID OR (IPOT < 0)) THEN
          BEGIN
            IF FSTATE = FJANDW THEN FGET(F)
            ELSE FSTATE := FNEEDCHAR;
            FREADINT(F,IPOT);
            IF FEOF THEN GOTO 1;
            IF NOT XVALID THEN X := 1; XVALID := TRUE;
            IF IPOT < 0 THEN X := X/PWROFTEN(ABS(IPOT))
            ELSE X := X*PWROFTEN(IPOT)
          END;
        IF XVALID THEN
          IF NEG THEN X := -X
```

```
          ELSE
        ELSE SYSCOM^.IORSLT := IBADFORMAT
      END;
1:
END (*FREADREAL*) ;

PROCEDURE FWRITEREAL(*X:REAL;  W, D: INTEGER*);
VAR J, TRUNCX, EXPX: INTEGER;
    NORMX: REAL;  S: STRING[30];

BEGIN
(* Check W and D for validity *)
IF (W < 0) OR (D < 0) THEN  BEGIN W := 0;  D := 0 END;

(* Take abs(x), normalize it and calculate exponent *)
IF X < 0 THEN  BEGIN X := -X;  S[1] := '-' END
         ELSE  S[1] := ' ';
EXPX := 0;  NORMX := X;
IF X >= PWROFTEN(0) THEN  (* divide down to size *)
  WHILE NORMX >= PWROFTEN(1) DO
    BEGIN EXPX := EXPX+1;  NORMX := X/PWROFTEN(EXPX) END
ELSE
  IF X <> 0 THEN  (* multiply up to size *)
    REPEAT
      EXPX := EXPX-1;  NORMX := X*PWROFTEN(-EXPX)
    UNTIL NORMX >= PWROFTEN(0);

(* Round number according to some very tricky rules *)
IF (D=0) OR (D+EXPX+1 > 6) THEN  (* scientific notation, or decimal places *)
  NORMX := NORMX + 5/PWROFTEN(6)        (* overspecified *)
ELSE IF D+EXPX+1 >= 0 THEN
  NORMX := NORMX + 5/PWROFTEN(D+EXPX+1);
(* if D+EXPX+1 < 0, then number is effectively 0.0 *)

(* If we just blew normalized stuff then fix it up *)
IF NORMX >= PWROFTEN(1) THEN
  BEGIN  EXPX := EXPX+1;  NORMX := NORMX/PWROFTEN(1) END;

(* Put the digits into a string *)
FOR J := 3 TO 8 DO
  BEGIN
    TRUNCX := TRUNC(NORMX);
    S[J] := CHR(TRUNCX+ORD('0'));
    NORMX := (NORMX-TRUNCX)*PWROFTEN(1)
  END;

(* Put number into proper form *)
IF (D=0) OR (EXPX >= 6) THEN  (* scientific notation *)
  BEGIN
    S[2] := S[3];
    S[3] := '.';
    J := 8;
    IF EXPX <> 0 THEN
      BEGIN
        J := 9;
        S[9] := 'E';
        IF EXPX < 0 THEN
          BEGIN J := 10;  S[10] := '-';  EXPX := -EXPX END;
        IF EXPX > 9 THEN
          BEGIN
```

```
            J := J+1;
            S[J] := CHR(EXPX DIV 10 + ORD('0'));
          END;
        J := J+1;
        S[J] := CHR(EXPX MOD 10 + ORD('0'))
      END;
    S[0] := CHR(J);
  END
ELSE  (* some kind of fixed point notation *)
  IF EXPX >= 0 THEN
    BEGIN
      MOVELEFT(S[3], S[2], EXPX+1);
      S[3+EXPX] := '.';
      FILLCHAR(S[9], D-(5-EXPX), ' '); (* blank fill at end if precision *)
      S[0] := CHR(3+D+EXPX);           (* was over-specified *)
    END
  ELSE
    BEGIN
      MOVERIGHT(S[3], S[3-EXPX], 6);  (* make room for leading zeroes *)
      S[2] := '0';
      S[3] := '.';
      FILLCHAR(S[4], -EXPX-1, '0');  (* put in leading zeroes *)
      FILLCHAR(S[9-EXPX], D-6+EXPX, ' ');(* put in blanks for over-precision*)
      S[0] := CHR(3+D);
    END;
IF W < LENGTH(S) THEN W := LENGTH(S);
FWRITESTRING( F, S, W );
END;  (*procedure write_real *)

PROCEDURE FWRITEDEC(*VAR F: FIB; D: DECMAX; RLENG: INTEGER*);
VAR S: STRING[38]; I: INTEGER;
BEGIN
  STR(D,S);
  FWRITESTRING(F,S,RLENG)
END (*FWRITEDEC*) ;

PROCEDURE FREADDEC(*VAR F:FIB; VAR D: TRICKARRAY; L: INTEGER*);
  LABEL 1;
  CONST DECSIZE = 8; (*MAX SIZE OF LONG INTEGER IN WORDS*)
  VAR DX: RECORD CASE BOOLEAN OF
            FALSE:( D: DECMAX );
            TRUE: ( WD: TRICKARRAY )
          END;
      CH: CHAR;
      NEG,DVALID: BOOLEAN; I: INTEGER;
BEGIN
  WITH F DO
      BEGIN
        DX.D := 0; NEG := FALSE; DVALID := FALSE;
        IF FSTATE = FNEEDCHAR THEN FGET(F);
        WHILE (FWINDOW^[0] = ' ') AND NOT FEOF DO FGET(F);
        IF FEOF THEN GOTO 1;
        CH := FWINDOW^[0];
        IF (CH = '+') OR (CH = '-') THEN
          BEGIN NEG := CH = '-'; FGET(F); CH := FWINDOW^[0] END;
        WHILE (CH IN DIGITS) AND NOT FEOF DO
          BEGIN DVALID := TRUE;
            DX.D := DX.D*10 + ORD(CH) - ORD('0');
            FGET(F); CH := FWINDOW^[0]
          END;
```

```
          IF DVALID OR FEOF THEN
            BEGIN
              IF NEG THEN DX.D := -DX.D;
              (*Transfer result into input var and check for overflow*)
              FOR I := L-1 DOWNTO 0 DO D[I] := DX.WD[I+DECSIZE-L];
              NEG := D[0] < 0;
              FOR I := DECSIZE-L-1 DOWNTO 0 DO
                IF ((NOT NEG) AND (DX.WD[I] <> 0))
                  OR (NEG AND (DX.WD[I] <> -1))  THEN DVALID := FALSE
            END;
          IF NOT (DVALID OR FEOF) THEN SYSCOM^.IORSLT := IBADFORMAT
        END;
   1:
   END(*FREADDEC*) ;

 END { PASCALIO } ;

{ +-----------------------------------------------------------------+
  |                                                                 |
  |                     F     I     N     I     S                   |
  |                                                                 |
  +-----------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 PIO Unit
```

```
########################################################################
### FILE: UCSD Pascal 1.5 Radix
########################################################################


 (* UCSD PASCAL I.5 P-SYSTEM "RADIX" *)

 PROGRAM CONVERSION;


 CONST  ORDA = 65;    {ASCII value of the character 'A'; ORD('A') }
        ORD0 = 48;    {ASCII value of the character '0'; ORD('0') }

        { The following values are terminal DEPENDENT }

           EEOLN = 29;   {Erase to end of line}
           EEOS = 11;    {Erase to end of screen}
           ESCAPE = 27;


 TYPE
      OREC = PACKED ARRAY[0..4] OF 0..7;  {High order bit comes from BREC }
      HREC = PACKED ARRAY[0..3] OF 0..15;
      BREC = PACKED ARRAY[0..15] OF 0..1;

      LETSET = SET OF '0'..'F';

      OCTSTR = STRING[6];    { These types are declared so as to allow type    }
      HEXSTR = STRING[4];    { checking of parameters passed to procedures in  }
      BINSTR = STRING[16];   { this program.  Octal and Integer have the same  }
                             { maximum length. No need to declare twice }

      XRANGE = 0..2; { This is for the arrays which determine the position of }
      YRANGE = 0..2; { where to writeout the information }

      ACROSS = 0..79;  { 80 Characters across screen }
      DOWN   = 0..23;  { 24 Lines down the screen    }


 VAR  R: RECORD CASE INTEGER OF  { Takes all the packed arrays above plus an   }
         1: (INT: INTEGER);      { Integer and assigns them all to read out of }
         2: (OCTREC: OREC);      { the same word.  Thus an octal value placed  }
         3: (HEXREC: HREC);      { in the record can be read out as an integer }
         4: (BINREC: BREC);      { using INT. }
        END;

      CH: CHAR;

      OCTLET, BINLET, DECLET, HEXLET: LETSET;  { Test sets for valid input }

      OCTX, NUMX, BINX, INTX, HEXX: PACKED ARRAY[XRANGE] OF ACROSS;
      OCTY, NUMY, BINY, INTY, HEXY: PACKED ARRAY[YRANGE] OF DOWN;
           { Arrays for positioning output correctly }

      X: XRANGE; { Global indices for the above arrays }
      Y: YRANGE;


 PROCEDURE PROMPT(S: STRING);  { Displays any string on the top line }
```

```
BEGIN
  GOTOXY(0,0);
  WRITE(S);
  WRITE(CHR(ESCAPE),CHR(EEOLN));  { Clears the line after the string }
END; (* PROMPT *)

PROCEDURE CLEARSCREEN;  { Clears the entire screen }
BEGIN
  GOTOXY(0,0);
  WRITE(CHR(ESCAPE),CHR(EEOS));
END;  (* CLEARSCREEN *)

PROCEDURE INIT;    { Initialize }
VAR I: INTEGER;
BEGIN
  HEXLET:=['A'..'F'];  { Initializes the test sets for input testing }
  DECLET:=['0'..'9'];
  OCTLET:=['0'..'7'];
  BINLET:=['0'..'1'];
  FOR I:=0 TO 2 DO     { Initializes the writeout positioning arrays }
    BEGIN
      NUMX[I]:=9 + (I * 27);
      INTX[I]:=9 + (I * 27);
      HEXX[I]:=9 + (I * 27);
      OCTX[I]:=9 + (I * 27);
      BINX[I]:=9 + (I * 27);
    END;
  FOR I:=0 TO 2 DO
    BEGIN
      NUMY[I]:=3 + (I * 6);
      INTY[I]:=4 + (I * 6);
      HEXY[I]:=5 + (I * 6);
      OCTY[I]:=6 + (I * 6);
      BINY[I]:=7 + (I * 6);
    END;
END;  (* INIT *)

PROCEDURE INITSCREEN; { Initializes the screen and the screen indices, X and Y }
VAR I,J,K: INTEGER;
    NAME: PACKED ARRAY[3..7] OF STRING;
BEGIN
  CLEARSCREEN;
  X:=0;
  Y:=0;
  NAME[3]:='NUMBER :';
  NAME[4]:='INTEGER:';
  NAME[5]:='HEX    :';
  NAME[6]:='OCTAL  :';
  NAME[7]:='BINARY :';
  FOR I:=3 TO 7 DO
    FOR J:=0 TO 2 DO
      FOR K:=0 TO 2 DO
        BEGIN
          GOTOXY(J * 27, I + (K * 6));
          WRITE(NAME[I]);
        END;
END; (* INITSCREEN *)

PROCEDURE DECTO(NUM: OCTSTR; VAR NUMVALID: BOOLEAN);
  { Procedure takes a string and converts it into an Integer }
```

```
VAR I: INTEGER;
    MINUS: BOOLEAN;
BEGIN
  MINUS:=FALSE;
  NUMVALID:=TRUE;
  WITH R DO
    BEGIN
      INT:=0;
      IF NUM[1] = '-' THEN
        BEGIN
          MINUS:=TRUE;
          DELETE(NUM,1,1);
        END;
      I:=1;
      WHILE (I <= LENGTH(NUM)) AND NUMVALID DO
        BEGIN  { Loop reads from left to right and adds value of new c }
               { character to 10 times the old value. Also checks for }
               { overflow and valid input. }

          IF NUM[I] IN DECLET THEN
            IF (INT < 3277) AND ( ORD(NUM[I]) - ORD0 <= 8 ) THEN
              INT:=(INT*10) + ORD(NUM[I]) - ORD0
            ELSE
              NUMVALID:=FALSE
          ELSE
            NUMVALID:=FALSE;
          I:=I+1;
        END; (* WHILE *)
      IF MINUS THEN          { This works on -32768 because -32768 is its }
        IF INT <= 32767 THEN { own negation in two's complement }
          INT:= -INT
        ELSE
          NUMVALID:=FALSE;
    END; (* WITH *)
  IF NOT NUMVALID THEN
    BEGIN
      GOTOXY(NUMX[X],NUMY[Y]);
      WRITE(' ':16);
      PROMPT('INVALID INTEGER NUMBER. Type <space> to continue');
    END;
END;  (* DECTO *)

PROCEDURE HEXTO(NUM: HEXSTR; VAR NUMVALID: BOOLEAN);
  { Procedure takes a string and converts it into a Hexadecimal number }
VAR I,J: INTEGER;
BEGIN
  WITH R DO
    BEGIN
      FOR I:=0 TO 3 DO
        HEXREC[I]:=0;
      I:=0;
      NUMVALID:=TRUE;
      J:=LENGTH(NUM);
      WHILE (J >= 1) AND NUMVALID DO
        BEGIN { Loop reads from right to left and puts the value of the }
              { character into the next array element. Also checks for  }
              { valid input }

          IF NUM[J] IN HEXLET THEN
```

```
            HEXREC[I]:= ORD(NUM[J])-ORDA + 10
          ELSE
            IF NUM[J] IN DECLET THEN
              HEXREC[I]:=ORD(NUM[J])-ORD0
            ELSE
              NUMVALID:=FALSE;
          J:=J-1;
          I:=I+1;
        END; (* WHILE *)
    END; (* WITH *)
  IF NOT NUMVALID THEN
    BEGIN
      GOTOXY(NUMX[X],NUMY[Y]);
      WRITE(' ':16);
      PROMPT('INVALID HEXADECIMAL NUMBER. Type <space> to continue');
    END;
END;  (* HEXTO *)

PROCEDURE OCTTO(NUM: OCTSTR; VAR NUMVALID: BOOLEAN);
  { Procedure takes a string and converts it to an Octal number }
VAR I,J: INTEGER;
BEGIN
  WITH R DO
    BEGIN
      FOR I:=0 TO 4 DO
        OCTREC[I]:=0;
      IF LENGTH(NUM) = 6 THEN  { If there is a high order byte get its value }
        BEGIN
          BINREC[15]:=ORD(NUM[1])-ORD0;
          DELETE(NUM,1,1);
        END
      ELSE                     { or else set it to zero }
        BINREC[15]:=0;
      I:=0;
      NUMVALID:=TRUE;
      J:=LENGTH(NUM);
      WHILE (J >= 1) AND NUMVALID DO
        BEGIN  { Loop reads from right to left and puts the value of the }
               { character into the next array element.  Also checks for }
               { valid input }

          IF NUM[J] IN OCTLET THEN
            OCTREC[I]:=ORD(NUM[J])-ORD0
          ELSE
            NUMVALID:=FALSE;
          J:=J-1;
          I:=I+1;
        END;  (* WHILE *)
    END;  (* WITH *)
  IF NOT NUMVALID THEN
    BEGIN
      GOTOXY(NUMX[X],NUMY[Y]);
      WRITE(' ':16);
      PROMPT('INVALID OCTAL NUMBER. Type <space> to continue');
    END;
END;  (* OCTTO *)

PROCEDURE BINTO(NUM: BINSTR; VAR NUMVALID: BOOLEAN);
  { Procedure takes a string a converts it into a binary number }
VAR I,J: INTEGER;
```

```
BEGIN
  WITH R DO
    BEGIN
      FOR I:=0 TO 15 DO
        BINREC[I]:=0;
      I:=LENGTH(NUM);
      NUMVALID:=TRUE;
      J:=0;
      WHILE (I >= 1) AND NUMVALID DO
        BEGIN  { Loop reads from right to left and puts the value of the }
               { character into the next array element.  Also checks for }
               { valid input. }

          IF NUM[I] IN BINLET THEN
            BINREC[J]:=ORD(NUM[I])-ORD0
          ELSE
            NUMVALID:=FALSE;
          I:=I-1;
          J:=J+1;
        END; (* WHILE *)
    END;  (* WITH *)
  IF NOT NUMVALID THEN
    BEGIN
      GOTOXY(NUMX[X],NUMY[Y]);
      WRITE(' ':16);
      PROMPT('INVALID BINARY NUMBER. Type <space> to continue');
    END;
END;  (* BINTO *)

PROCEDURE WRITEOUT;
{ Procedure writes out all the elements of global variable R to the appropiate }
{ section of the screen and then increments X and Y }
VAR I: INTEGER;
BEGIN
  GOTOXY(INTX[X],INTY[Y]);
  WRITE(R.INT);

  GOTOXY(HEXX[X],HEXY[Y]);
  FOR I:=3 DOWNTO 0 DO
    IF R.HEXREC[I] < 10 THEN WRITE(R.HEXREC[I])
    ELSE WRITE(CHR(ORD(R.HEXREC[I])-10+ORDA));

  GOTOXY(OCTX[X],OCTY[Y]);
  WRITE(R.BINREC[15],R.OCTREC[4],R.OCTREC[3],R.OCTREC[2],R.OCTREC[1],
        R.OCTREC[0]);

  GOTOXY(BINX[X],BINY[Y]);
  FOR I:=15 DOWNTO 0 DO
    WRITE(R.BINREC[I]);

  IF X = 2 THEN  { If end of row }
    BEGIN
      IF Y = 2 THEN  { If end of screen }
        BEGIN
          PROMPT('Type <space> to clear the screen and continue');
          READ(CH);
          INITSCREEN;
         END
      ELSE
        Y:=Y+1;
```

```
      X:=0;
    END
  ELSE
    X:=X+1;

END;  (* WRITEOUT *)

PROCEDURE OUTER;
{ This procedure is the outer loop and only working loop of the program. }
{ It reads all user input and calls the appropiate procedure. }
VAR CH: CHAR;
STR: STRING;
    NUMVALID,VALID: BOOLEAN;
    O: OCTSTR;     { For passing strings to the procedures.  Octal and }
    H: HEXSTR;     { Integer have the same size string. }
    B: BINSTR;
BEGIN
  INIT;
  INITSCREEN;
  REPEAT
    VALID:=TRUE;
    PROMPT(
'Type the number followed by the Radix (H,O,I,B), or type C(learscreen, Q(uit');
    GOTOXY(NUMX[X],NUMY[Y]);
    READLN(STR);
    IF LENGTH(STR) = 0 THEN
      VALID:=FALSE
    ELSE
      IF STR[LENGTH(STR)] IN ['H','O','B','I','C','Q'] THEN
        CASE STR[LENGTH(STR)] OF
          'Q' : EXIT(OUTER);

          'C' : INITSCREEN;

          'I' : BEGIN
                  DELETE(STR,LENGTH(STR),1); { Delete radix character }
                  IF (LENGTH(STR) > 6) OR (LENGTH(STR) = 0) THEN
                    VALID:=FALSE
                  ELSE
                    BEGIN
                      O:=STR;
                      DECTO(O,NUMVALID);
                      IF NUMVALID THEN WRITEOUT
                      ELSE READ(CH);
                    END;
                END;

          'H' : BEGIN
                  DELETE(STR,LENGTH(STR),1); { Delete radix character }
                  IF (LENGTH(STR) > 4) OR (LENGTH(STR) = 0) THEN
                    VALID:=FALSE
                  ELSE
                    BEGIN
                      H:=STR;
                      HEXTO(H,NUMVALID);
                      IF NUMVALID THEN WRITEOUT
                      ELSE READ(CH);
                    END;
                END;
```

```
          'O' : BEGIN
                  DELETE(STR,LENGTH(STR),1); { Delete radix character }
                  IF (LENGTH(STR) > 6) OR ( LENGTH(STR) = 0) THEN
                    VALID:=FALSE
                  ELSE
                    BEGIN
                      O:=STR;
                      OCTTO(O,NUMVALID);
                      IF NUMVALID THEN WRITEOUT
                      ELSE READ(CH);
                    END;
                END;

          'B' : BEGIN
                  DELETE(STR,LENGTH(STR),1);
                  IF (LENGTH(STR) > 16) OR (LENGTH(STR) = 0)  THEN
                    VALID:=FALSE
                  ELSE
                    BEGIN
                      B:=STR;
                      BINTO(B,NUMVALID);
                      IF NUMVALID THEN WRITEOUT
                      ELSE READ(CH);
                    END;
                END;
        END  (* CASE *)
      ELSE
        VALID:=FALSE;  (* END OF IF RADIX IN SET *)
    IF NOT VALID THEN
      BEGIN
        PROMPT('INVALID INPUT.  Type <space> to continue.');
        GOTOXY(NUMX[X],NUMY[Y]);
        WRITE(' ':17);  { Blank out bad input }
        READ(CH);
      END;
  UNTIL 1 = 2; {FOREVER}
END; (* OUTER *)

PROCEDURE HEADER;
BEGIN
  CLEARSCREEN;

  WRITELN('     This program will convert numbers specified as HEX, OCTAL,');
  WRITELN('INTEGER and BINARY numbers to the other radices.');
  WRITELN;
  WRITELN('     Simply type the number followed by the first letter of the');
  WRITELN('radix it is in, (i.e. 7BC9H, -12789I, 110010100B, 177760O )');
  WRITELN('followed by a carriage return.');
  WRITELN;
  WRITELN('     This program works only with uppercase characters.  Lower ');
  WRITELN('case characters will give "INVALID INPUT" responses. ');
  WRITELN;
  WRITELN('     For further information see the document.');
  WRITELN;
  WRITELN;
  WRITELN('     Type a <space> to continue;   Q(uit');
END;

BEGIN (* MAIN *)
  HEADER;
```

```
   READ(CH);
   OUTER;
 END.


{ +-------------------------------------------------------------------+
  |                                                                   |
  |                     F     I     N     I     S                     |
  |                                                                   |
  +-------------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 Radix
```

```
##########################################################################
### FILE: UCSD Pascal 1.5 System
##########################################################################


{ $I GLOBALS }
(*$U-,S+*)

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

PROGRAM PASCALSYSTEM;

(*************************************************)
(*                                               *)
(*    UCSD PASCAL OPERATING SYSTEM               *)
(*                                               *)
(*    RELEASE LEVEL:  I.3   AUGUST, 1977         *)
(*                      I.4   JANUARY, 1978      *)
(*                    I.5   SEPTEMBER, 1978      *)
(*                                               *)
(*    WRITTEN BY ROGER T. SUMNER                 *)
(*    WINTER 1977                                *)
(*                                               *)
(*    INSTITUTE FOR INFORMATION SYSTEMS          *)
(*    UC SAN DIEGO, LA JOLLA, CA                 *)
(*                                               *)
(*    KENNETH L. BOWLES, DIRECTOR                *)
(*                                               *)
(*************************************************)

CONST
     MMAXINT = 32767;   (*MAXIMUM INTEGER VALUE*)
     MAXUNIT = 12;      (*MAXIMUM PHYSICAL UNIT # FOR UREAD*)
     MAXDIR = 77;       (*MAX NUMBER OF ENTRIES IN A DIRECTORY*)
     VIDLENG = 7;       (*NUMBER OF CHARS IN A VOLUME ID*)
     TIDLENG = 15;      (*NUMBER OF CHARS IN TITLE ID*)
     MAXSEG = 15;       (*MAX CODE SEGMENT NUMBER*)
     FBLKSIZE = 512;    (*STANDARD DISK BLOCK LENGTH*)
     DIRBLK = 2;        (*DISK ADDR OF DIRECTORY*)
     AGELIMIT = 300;    (*MAX AGE FOR GDIRP...IN TICKS*)
     EOL = 13;          (*END-OF-LINE...ASCII CR*)
     DLE = 16;          (*BLANK COMPRESSION CODE*)

TYPE

     IORSLTWD = (INOERROR,IBADBLOCK,IBADUNIT,IBADMODE,ITIMEOUT,
                 ILOSTUNIT,ILOSTFILE,IBADTITLE,INOROOM,INOUNIT,
                 INOFILE,IDUPFILE,INOTCLOSED,INOTOPEN,IBADFORMAT,
                 ISTRGOVFL);

                                        (*COMMAND STATES...SEE GETCMD*)
```

```
CMDSTATE = (HALTINIT,DEBUGCALL,
            UPROGNOU,UPROGUOK,SYSPROG,
            COMPONLY,COMPANDGO,COMPDEBUG,
            LINKANDGO,LINKDEBUG);

                                    (*CODE FILES USED IN GETCMD*)

SYSFILE = (ASSMBLER,COMPILER,EDITOR,FILER,LINKER);

                                    (*ARCHIVAL INFO...THE DATE*)

DATEREC = PACKED RECORD
            MONTH: 0..12;           (*0 IMPLIES DATE NOT MEANINGFUL*)
            DAY: 0..31;             (*DAY OF MONTH*)
            YEAR: 0..100            (*100 IS TEMP DISK FLAG*)
          END (*DATEREC*) ;

                                    (*VOLUME TABLES*)
UNITNUM = 0..MAXUNIT;
VID = STRING[VIDLENG];

                                    (*DISK DIRECTORIES*)
DIRRANGE = 0..MAXDIR;
TID = STRING[TIDLENG];

FILEKIND = (UNTYPEDFILE,XDSKFILE,CODEFILE,TEXTFILE,
            INFOFILE,DATAFILE,GRAFFILE,FOTOFILE,SECUREDIR);

DIRENTRY = RECORD
             DFIRSTBLK: INTEGER;    (*FIRST PHYSICAL DISK ADDR*)
             DLASTBLK: INTEGER;     (*POINTS AT BLOCK FOLLOWING*)
             CASE DFKIND: FILEKIND OF
               SECUREDIR,
               UNTYPEDFILE: (*ONLY IN DIR[0]...VOLUME INFO*)
                  (DVID: VID;               (*NAME OF DISK VOLUME*)
                   DEOVBLK: INTEGER;        (*LASTBLK OF VOLUME*)
                   DNUMFILES: DIRRANGE;     (*NUM FILES IN DIR*)
                   DLOADTIME: INTEGER;      (*TIME OF LAST ACCESS*)
                   DLASTBOOT: DATEREC);     (*MOST RECENT DATE SETTING*)
               XDSKFILE,CODEFILE,TEXTFILE,INFOFILE,
               DATAFILE,GRAFFILE,FOTOFILE:
                  (DTID: TID;               (*TITLE OF FILE*)
                   DLASTBYTE: 1..FBLKSIZE;  (*NUM BYTES IN LAST BLOCK*)
                   DACCESS: DATEREC)        (*LAST MODIFICATION DATE*)
           END (*DIRENTRY*) ;

DIRP = ^DIRECTORY;

DIRECTORY = ARRAY [DIRRANGE] OF DIRENTRY;

                                    (*FILE INFORMATION*)

CLOSETYPE = (CNORMAL,CLOCK,CPURGE,CCRUNCH);
WINDOWP = ^WINDOW;
WINDOW = PACKED ARRAY [0..0] OF CHAR;
FIBP = ^FIB;

FIB = RECORD
        FWINDOW: WINDOWP;  (*USER WINDOW...F^, USED BY GET-PUT*)
        FEOF,FEOLN: BOOLEAN;
```

```
        FSTATE: (FJANDW,FNEEDCHAR,FGOTCHAR);
        FRECSIZE: INTEGER; (*IN BYTES...0=>BLOCKFILE, 1=>CHARFILE*)
        CASE FISOPEN: BOOLEAN OF
          TRUE: (FISBLKD: BOOLEAN; (*FILE IS ON BLOCK DEVICE*)
                 FUNIT: UNITNUM;   (*PHYSICAL UNIT #*)
                 FVID: VID;        (*VOLUME NAME*)
                 FREPTCNT,         (* # TIMES F^ VALID W/O GET*)
                 FNXTBLK,          (*NEXT REL BLOCK TO IO*)
                 FMAXBLK: INTEGER; (*MAX REL BLOCK ACCESSED*)
                 FMODIFIED:BOOLEAN;(*PLEASE SET NEW DATE IN CLOSE*)
                 FHEADER: DIRENTRY;(*COPY OF DISK DIR ENTRY*)
                 CASE FSOFTBUF: BOOLEAN OF (*DISK GET-PUT STUFF*)
                   TRUE: (FNXTBYTE,FMAXBYTE: INTEGER;
                          FBUFCHNGD: BOOLEAN;
                          FBUFFER: PACKED ARRAY [0..FBLKSIZE] OF CHAR))
      END (*FIB*) ;

                                  (*USER WORKFILE STUFF*)

INFOREC = RECORD
            SYMFIBP,CODEFIBP: FIBP;        (*WORKFILES FOR SCRATCH*)
            ERRSYM,ERRBLK,ERRNUM: INTEGER; (*ERROR STUFF IN EDIT*)
            SLOWTERM,STUPID: BOOLEAN;      (*STUDENT PROGRAMMER ID!!*)
            ALTMODE: CHAR;                 (*WASHOUT CHAR FOR COMPILER*)
            GOTSYM,GOTCODE: BOOLEAN;       (*TITLES ARE MEANINGFUL*)
            WORKVID,SYMVID,CODEVID: VID;   (*PERM&CUR WORKFILE VOLUMES*)
            WORKTID,SYMTID,CODETID: TID    (*PERM&CUR WORKFILES TITLE*)
          END (*INFOREC*) ;

                                  (*CODE SEGMENT LAYOUTS*)

SEGRANGE = 0..MAXSEG;
SEGDESC = RECORD
            DISKADDR: INTEGER;     (*REL BLK IN CODE...ABS IN SYSCOM^*)
            CODELENG: INTEGER      (*# BYTES TO READ IN*)
          END (*SEGDESC*) ;

                                  (*DEBUGGER STUFF*)

BYTERANGE = 0..255;
TRICKARRAY = ARRAY [0..0] OF INTEGER; (* FOR MEMORY DIDDLING*)
MSCWP = ^ MSCW;            (*MARK STACK RECORD POINTER*)
MSCW = RECORD
         STATLINK: MSCWP;  (*POINTER TO PARENT MSCW*)
         DYNLINK: MSCWP;   (*POINTER TO CALLER'S MSCW*)
         MSSEG,MSJTAB: ^TRICKARRAY;
         MSIPC: INTEGER;
         LOCALDATA: TRICKARRAY
       END (*MSCW*) ;

                                  (*SYSTEM COMMUNICATION AREA*)
                                  (*SEE INTERPRETERS...NOTE  *)
                                  (*THAT WE ASSUME BACKWARD  *)
                                  (*FIELD ALLOCATION IS DONE *)

SYSCOMREC = RECORD
              IORSLT: IORSLTWD;    (*RESULT OF LAST IO CALL*)
              XEQERR: INTEGER;     (*REASON FOR EXECERROR CALL*)
              SYSUNIT: UNITNUM;    (*PHYSICAL UNIT OF BOOTLOAD*)
              BUGSTATE: INTEGER;   (*DEBUGGER INFO*)
```

```
                    GDIRP: DIRP;          (*GLOBAL DIR POINTER,SEE VOLSEARCH*)
                    LASTMP,STKBASE,BOMBP: MSCWP;
                    MEMTOP,SEG,JTAB: INTEGER;
                    BOMBIPC: INTEGER;     (*WHERE XEQERR BLOWUP WAS*)
                    HLTLINE: INTEGER;     (*MORE DEBUGGER STUFF*)
                    BRKPTS: ARRAY [0..3] OF INTEGER;
                    RETRIES: INTEGER;     (*DRIVERS PUT RETRY COUNTS*)
                    EXPANSION: ARRAY [0..8] OF INTEGER;
                    HIGHTIME,LOWTIME: INTEGER;
                    MISCINFO: PACKED RECORD
                                NOBREAK,STUPID,SLOWTERM,
                                HASXYCRT,HASLCCRT,HAS8510A,HASCLOCK: BOOLEAN;
                                USERKIND:(NORMAL, AQUIZ, BOOKER, PQUIZ)
                              END;
                    CRTTYPE: INTEGER;
                    CRTCTRL: PACKED RECORD
                                RLF,NDFS,ERASEEOL,ERASEEOS,HOME,ESCAPE: CHAR;
                                BACKSPACE: CHAR;
                                FILLCOUNT: 0..255;
                                CLEARSCREEN, CLEARLINE: CHAR;
                                PREFIXED: PACKED ARRAY [0..8] OF BOOLEAN
                              END;
                    CRTINFO: PACKED RECORD
                                WIDTH,HEIGHT: INTEGER;
                                RIGHT,LEFT,DOWN,UP: CHAR;
                                BADCH,CHARDEL,STOP,BREAK,FLUSH,EOF: CHAR;
                                ALTMODE,LINEDEL: CHAR;
                                BACKSPACE,ETX,PREFIX: CHAR;
                                PREFIXED: PACKED ARRAY [0..13] OF BOOLEAN
                              END;
                    SEGTABLE: ARRAY [SEGRANGE] OF
                                RECORD
                                  CODEUNIT: UNITNUM;
                                  CODEDESC: SEGDESC
                                END
                  END (*SYSCOM*);

     MISCINFOREC = RECORD
                     MSYSCOM: SYSCOMREC
                   END;

VAR
    SYSCOM: ^SYSCOMREC;                 (*MAGIC PARAM...SET UP IN BOOT*)
    GFILES: ARRAY [0..5] OF FIBP;       (*GLOBAL FILES, 0=INPUT, 1=OUTPUT*)
    USERINFO: INFOREC;                  (*WORK STUFF FOR COMPILER ETC*)
    EMPTYHEAP: ^INTEGER;                (*HEAP MARK FOR MEM MANAGING*)
    INPUTFIB,OUTPUTFIB,                 (*CONSOLE FILES...GFILES ARE COPIES*)
    SYSTERM,SWAPFIB: FIBP;              (*CONTROL AND SWAPSPACE FILES*)
    SYVID,DKVID: VID;                   (*SYSUNIT VOLID & DEFAULT VOLID*)
    THEDATE: DATEREC;                   (*TODAY...SET IN FILER OR SIGN ON*)
    DEBUGINFO: ^INTEGER;                (*DEBUGGERS GLOBAL INFO WHILE RUNIN*)
    STATE: CMDSTATE;                    (*FOR GETCOMMAND*)
    PL: STRING;                         (*PROMPTLINE STRING...SEE PROMPT*)
    IPOT: ARRAY [0..4] OF INTEGER;      (*INTEGER POWERS OF TEN*)
    FILLER: STRING[11];                 (*NULLS FOR CARRIAGE DELAY*)
    DIGITS: SET OF '0'..'9';
    UNITABLE: ARRAY [UNITNUM] OF (*0 NOT USED*)
                RECORD
                  UVID: VID;    (*VOLUME ID FOR UNIT*)
                  CASE UISBLKD: BOOLEAN OF
```

```
                TRUE: (UEOVBLK: INTEGER)
            END (*UNITABLE*) ;
    FILENAME: ARRAY [SYSFILE] OF STRING[23];

(*-------------------------------------------------------------------------*)
(* SYSTEM PROCEDURE FORWARD DECLARATIONS *)
(* THESE ARE ADDRESSED BY OBJECT CODE... *)
(*  DO NOT MOVE WITHOUT CAREFUL THOUGHT  *)

PROCEDURE EXECERROR;
  FORWARD;
PROCEDURE FINIT(VAR F: FIB; WINDOW: WINDOWP; RECWORDS: INTEGER);
  FORWARD;
PROCEDURE FRESET(VAR F: FIB);
  FORWARD;
PROCEDURE FOPEN(VAR F: FIB; VAR FTITLE: STRING;
              FOPENOLD: BOOLEAN; JUNK: FIBP);
  FORWARD;
PROCEDURE FCLOSE(VAR F: FIB; FTYPE: CLOSETYPE);
  FORWARD;
PROCEDURE FGET(VAR F: FIB);
  FORWARD;
PROCEDURE FPUT(VAR F: FIB);
  FORWARD;
PROCEDURE XSEEK;
  FORWARD;
FUNCTION FEOF(VAR F: FIB): BOOLEAN;
  FORWARD;
FUNCTION FEOLN(VAR F: FIB): BOOLEAN;
  FORWARD;
PROCEDURE FREADINT(VAR F: FIB; VAR I: INTEGER);
  FORWARD;
PROCEDURE FWRITEINT(VAR F: FIB; I,RLENG: INTEGER);
  FORWARD;
PROCEDURE XREADREAL;
  FORWARD;
PROCEDURE XWRITEREAL;
  FORWARD;
PROCEDURE FREADCHAR(VAR F: FIB; VAR CH: CHAR);
  FORWARD;
PROCEDURE FWRITECHAR(VAR F: FIB; CH: CHAR; RLENG: INTEGER);
  FORWARD;
PROCEDURE FREADSTRING(VAR F: FIB; VAR S: STRING; SLENG: INTEGER);
  FORWARD;
PROCEDURE FWRITESTRING(VAR F: FIB; VAR S: STRING; RLENG: INTEGER);
  FORWARD;
PROCEDURE FWRITEBYTES(VAR F: FIB; VAR A: WINDOW; RLENG,ALENG: INTEGER);
  FORWARD;
PROCEDURE FREADLN(VAR F: FIB);
  FORWARD;
PROCEDURE FWRITELN(VAR F: FIB);
  FORWARD;
PROCEDURE SCONCAT(VAR DEST,SRC: STRING; DESTLENG: INTEGER);
  FORWARD;
PROCEDURE SINSERT(VAR SRC,DEST: STRING; DESTLENG,INSINX: INTEGER);
  FORWARD;
PROCEDURE SCOPY(VAR SRC,DEST: STRING; SRCINX,COPYLENG: INTEGER);
  FORWARD;
PROCEDURE SDELETE(VAR DEST: STRING; DELINX,DELLENG: INTEGER);
  FORWARD;
```

```
FUNCTION SPOS(VAR TARGET,SRC: STRING): INTEGER;
  FORWARD;
FUNCTION FBLOCKIO(VAR F: FIB; VAR A: WINDOW;
                  NBLOCKS,RBLOCK: INTEGER; DOREAD: BOOLEAN): INTEGER;
  FORWARD;
PROCEDURE FGOTOXY(X,Y: INTEGER);
  FORWARD;

(* NON FIXED FORWARD DECLARATIONS *)

FUNCTION VOLSEARCH(VAR FVID: VID; LOOKHARD: BOOLEAN;
                   VAR FDIR: DIRP): UNITNUM;
  FORWARD;
PROCEDURE WRITEDIR(FUNIT: UNITNUM; FDIR: DIRP);
  FORWARD;
FUNCTION DIRSEARCH(VAR FTID: TID; FINDPERM: BOOLEAN; FDIR: DIRP): DIRRANGE;
  FORWARD;
FUNCTION SCANTITLE(FTITLE: STRING; VAR FVID: VID; VAR FTID: TID;
                   VAR FSEGS: INTEGER; VAR FKIND: FILEKIND): BOOLEAN;
  FORWARD;
PROCEDURE DELENTRY(FINX: DIRRANGE; FDIR: DIRP);
  FORWARD;
PROCEDURE INSENTRY(VAR FENTRY: DIRENTRY; FINX: DIRRANGE; FDIR: DIRP);
  FORWARD;
PROCEDURE HOMECURSOR;
  FORWARD;
PROCEDURE CLEARSCREEN;
  FORWARD;
PROCEDURE CLEARLINE;
  FORWARD;
PROCEDURE PROMPT;
  FORWARD;
FUNCTION SPACEWAIT(FLUSH: BOOLEAN): BOOLEAN;
  FORWARD;
FUNCTION GETCHAR(FLUSH: BOOLEAN): CHAR;
  FORWARD;
PROCEDURE COMMAND;
  FORWARD;
{ $I SYSSEGS  }

    (*************************************************************)
    (*                                                           *)
    (*  Copyright (c) 1978 Regents of the University of California.   *)
    (*  Permission to copy or distribute this software or documen-    *)
    (*  tation in hard or soft copy granted only by written license   *)
    (*  obtained from the Institute for Information Systems.          *)
    (*                                                           *)
    (*************************************************************)

SEGMENT PROCEDURE USERPROGRAM(INPUT,OUTPUT: FIBP);
BEGIN FWRITELN(SYSTERM^);
  PL := 'No user program';
  FWRITESTRING(SYSTERM^,PL,0)
END (*USERPROGRAM*) ;

SEGMENT PROCEDURE DEBUGGER;
BEGIN FWRITELN(SYSTERM^);
  PL := 'No debugger in system';
  FWRITESTRING(SYSTERM^,PL,0)
END (*DEBUGGER*) ;
```

```
SEGMENT PROCEDURE PRINTERROR(XEQERR,IORSLT: INTEGER);
  VAR S: STRING[40];
BEGIN S := 'Unknown run-time error';
  CASE XEQERR OF
    1:  S := 'Value range error';
    2:  S := 'No proc in seg-table';
    3:  S := 'Exit from uncalled proc';
    4:  S := 'Stack overflow';
    5:  S := 'Integer overflow';
    6:  S := 'Divide by zero';
    7:  S := 'NIL pointer reference';
    8:  S := 'Program interrupted by user';
    9:  S := 'System IO error';
   10:  BEGIN S := 'unknown cause';
          CASE IORSLT OF
            1:  S := 'parity (CRC)';
            2:  S := 'illegal unit #';
            3:  S := 'illegal IO request';
            4:  S := 'data-com timeout';
            5:  S := 'vol went off-line';
            6:  S := 'file lost in dir';
            7:  S := 'bad file name';
            8:  S := 'no room on vol';
            9:  S := 'vol not found';
           10:  S := 'file not found';
           11:  S := 'dup dir entry';
           12:  S := 'file already open';
           13:  S := 'file not open';
           14:  S := 'bad input format'
          END (*IO ERRORS*) ;
        INSERT('IO error: ',S,1)
      END;
   11:  S := 'Unimplemented instruction';
   12:  S := 'Floating point error';
   13:  S := 'String overflow';
   14:  S := 'Programmed HALT';
   15:  S := 'Programmed break-point'
  END (*XEQ ERRORS*) ;

  WRITELN(OUTPUT,S);
  WITH SYSCOM^.BOMBP^ DO
    WRITE(OUTPUT,'S# ',MSSEG^[0] MOD 256,
                 ', P# ',MSJTAB^[0] MOD 256,
                 ', I# ',MSIPC-(ORD(MSJTAB)-2-MSJTAB^[-1]))
END (*PRINTERROR*) ;

SEGMENT PROCEDURE INITIALIZE;
  VAR DOTRITON,JUSTBOOTED: BOOLEAN; LTITLE: STRING[40];
      MONTHS: ARRAY [0..15] OF STRING[3];
      DISPLAY: ARRAY [0..79,0..19] OF INTEGER; (*FOR TRITON*)
      STKFILL: ARRAY [0..1199] OF INTEGER;

  PROCEDURE INITSYSCOM;
    VAR TITLE: STRING;
        F: FILE OF MISCINFOREC;
    BEGIN
     (* FIRST SOME GLOBALS *)
        FILLER[0] := CHR(SYSCOM^.CRTCTRL.FILLCOUNT);
        FILLCHAR( FILLER[1], SYSCOM^.CRTCTRL.FILLCOUNT, CHR(0) );
```

```
    DEBUGINFO := NIL;
    IPOT[0] := 1; IPOT[1] := 10; IPOT[2] := 100;
    IPOT[3] := 1000; IPOT[4] := 10000; DIGITS := ['0'..'9'];
    WITH SYSCOM^ DO
      BEGIN
      XEQERR := 0;    IORSLT := INOERROR;
      BUGSTATE :=0
      END;
    TITLE := '*SYSTEM.MISCINFO' ;
    RESET( F, TITLE );
    IF IORESULT = ORD(INOERROR) THEN
      BEGIN
      IF NOT EOF( F ) THEN
        WITH SYSCOM^, F^ DO
          BEGIN
          MISCINFO := MSYSCOM.MISCINFO;
          CRTTYPE := MSYSCOM.CRTTYPE;
          CRTCTRL := MSYSCOM.CRTCTRL;
          CRTINFO := MSYSCOM.CRTINFO;
          FILLER[0] := CHR(SYSCOM^.CRTCTRL.FILLCOUNT);
          FILLCHAR( FILLER[1], SYSCOM^.CRTCTRL.FILLCOUNT, CHR(0) );
          END;
      CLOSE( F, NORMAL )
      END;
    UNITCLEAR(1) (*GIVE BIOS NEW SOFT CHARACTERS FOR CONSOLE*)
  END (*INITSYSCOM*) ;

PROCEDURE INITUNITABLE;
  VAR LUNIT: UNITNUM; LDIR: DIRP;
BEGIN
  FOR LUNIT := 0 TO MAXUNIT DO
    WITH UNITABLE[LUNIT] DO
      BEGIN UVID := '';
        UISBLKD := LUNIT IN [4,5,9..12];
        IF UISBLKD THEN UEOVBLK := MMAXINT;
        UNITCLEAR(LUNIT);
      END;
  UNITABLE[1].UVID := 'CONSOLE';
  UNITABLE[2].UVID := 'SYSTERM';
  SYVID := '';
  LUNIT := VOLSEARCH(SYVID,TRUE,LDIR);
  SYVID := UNITABLE[SYSCOM^.SYSUNIT].UVID;
  IF LENGTH(SYVID) = 0 THEN HALT;
  IF JUSTBOOTED THEN DKVID := SYVID;
  LUNIT := VOLSEARCH(SYVID,FALSE,LDIR);
  IF LDIR = NIL THEN HALT;
  THEDATE := LDIR^[0].DLASTBOOT;
  UNITCLEAR(6);
  IF IORESULT = ORD(INOERROR) THEN
    UNITABLE[6].UVID := 'PRINTER';
  UNITCLEAR(8);
  IF IORESULT = ORD(INOERROR) THEN
    UNITABLE[8].UVID := 'REMOTE';
END (*INITUNITABLE*) ;

PROCEDURE INITFNAMES;
  VAR F: SYSFILE;
      ALLOFEM, FOUND: SET OF SYSFILE;
      LUNIT: UNITNUM;
      LFIB: FIB;
```

```
BEGIN
  FILENAME[ASSMBLER] := 'ASSMBLER';
  FILENAME[COMPILER] := 'COMPILER';
  FILENAME[EDITOR] := 'EDITOR';
  FILENAME[FILER] := 'FILER';
  FILENAME[LINKER] := 'LINKER';
  FINIT(LFIB, NIL,-1);
  FOUND := [];
  FOR F := ASSMBLER TO LINKER DO
    BEGIN
      INSERT(':SYSTEM.', FILENAME[F], 1);
      LTITLE := CONCAT(SYVID,FILENAME[F]);
      FOPEN(LFIB, LTITLE, TRUE, NIL);
      IF LFIB.FISOPEN THEN
        BEGIN
          FILENAME[F] := LTITLE;
          FOUND := FOUND + [F]
        END;
      FCLOSE(LFIB, CNORMAL)
    END;
  LUNIT := 1;
  ALLOFEM := [ASSMBLER,COMPILER,EDITOR,FILER,LINKER];
  WHILE FOUND <> ALLOFEM DO
    BEGIN
      WITH UNITABLE[LUNIT] DO
        IF UISBLKD THEN
          IF UVID <> '' THEN
            FOR F := ASSMBLER TO LINKER DO
              IF NOT (F IN FOUND) THEN
                BEGIN
                  LTITLE := CONCAT(UVID, FILENAME[F]);
                  FOPEN(LFIB,LTITLE,TRUE,NIL);

                  IF LFIB.FISOPEN THEN
                    BEGIN
                      FILENAME[F] := LTITLE;
                      FOUND := FOUND + [F]
                    END;
                  FCLOSE(LFIB, CNORMAL)
                END;
      IF LUNIT = MAXUNIT THEN
        FOUND := ALLOFEM
      ELSE
        LUNIT := LUNIT+1
    END { WHILE }
END (*INITFNAMES*) ;

PROCEDURE INITCHARSET;
TYPE CHARSET= ARRAY [32..127] OF
                PACKED ARRAY [0..9] OF 0..255;
VAR I: INTEGER;
    TRIX: RECORD CASE BOOLEAN OF
            TRUE:  (CHARADDR: INTEGER);
            FALSE: (CHARBUFP: ^ CHAR)
          END;
    CHARBUF: RECORD
               SET1: CHARSET;
               FILLER1: PACKED ARRAY [0..63] OF CHAR;
               SET2: CHARSET;
               FILLER2: PACKED ARRAY [0..63] OF CHAR;
```

```
                TRITON: ARRAY [0..63,0..3] OF INTEGER
              END (*CHARBUF*) ;
    LFIB: FIB;
BEGIN FINIT(LFIB,NIL,-1);
  LTITLE := '*SYSTEM.CHARSET';
  FOPEN(LFIB,LTITLE,TRUE,NIL);
  IF LFIB.FISOPEN THEN
    BEGIN UNITWRITE(3,TRIX,128);
      IF IORESULT = ORD(INOERROR) THEN
        BEGIN
          WITH LFIB.FHEADER DO
            BEGIN DOTRITON := DLASTBLK-DFIRSTBLK > 4;
              UNITREAD(LFIB.FUNIT,CHARBUF,SIZEOF(CHARBUF),DFIRSTBLK)
            END;
          TRIX.CHARADDR := 512-8192;  (*UNIBUS TRICKYNESS!*)
          FOR I := 32 TO 127 DO
            BEGIN
              MOVERIGHT(CHARBUF.SET1[I],TRIX.CHARBUFP^,10);
              TRIX.CHARADDR := TRIX.CHARADDR+16
            END;
          TRIX.CHARADDR := 512-6144;
          FOR I := 32 TO 127 DO
            BEGIN
              MOVERIGHT(CHARBUF.SET2[I],TRIX.CHARBUFP^,10);
              TRIX.CHARADDR := TRIX.CHARADDR+16
            END;
          UNITABLE[3].UVID := 'GRAPHIC';
          UNITWRITE(3,I,0)
        END
    END
  ELSE
    SYSCOM^.MISCINFO.HAS8510A := FALSE;
  IF DOTRITON THEN
    BEGIN (*INITIALIZE DISPLAY ARRAY*)
      FILLCHAR(DISPLAY,SIZEOF(DISPLAY),0);
      FOR I := 0 TO 63 DO
        MOVELEFT(CHARBUF.TRITON[I],DISPLAY[I,10],8)
    END;
  FCLOSE(LFIB,CNORMAL)
END (*INITCHARSET*) ;

PROCEDURE INITHEAP;
VAR LWINDOW: WINDOWP;
BEGIN (*BASIC FILE AND HEAP SETTUP*)
  SYSCOM^.GDIRP := NIL; (* MUST PRECEDE THE FIRST "NEW" EXECUTED *)
  NEW(SWAPFIB,TRUE,FALSE); FINIT(SWAPFIB^,NIL,-1);
  NEW(INPUTFIB,TRUE,FALSE);  NEW(LWINDOW);
  FINIT(INPUTFIB^,LWINDOW,0);
  NEW(OUTPUTFIB,TRUE,FALSE);  NEW(LWINDOW);
  FINIT(OUTPUTFIB^,LWINDOW,0);
  NEW(SYSTERM,TRUE,FALSE);  NEW(LWINDOW);
  FINIT(SYSTERM^,LWINDOW,0);
  GFILES[0] := INPUTFIB; GFILES[1] := OUTPUTFIB;
  WITH USERINFO DO
    BEGIN
      NEW(SYMFIBP,TRUE,FALSE); FINIT(SYMFIBP^,NIL,-1);
      NEW(CODEFIBP,TRUE,FALSE); FINIT(CODEFIBP^,NIL,-1)
    END;
  MARK(EMPTYHEAP)
END (*INITHEAP*) ;
```

```
  PROCEDURE INITWORKFILE;
  BEGIN
    WITH USERINFO DO
      BEGIN (*INITIALIZE WORK FILES ETC*)
        ERRNUM := 0; ERRBLK := 0; ERRSYM := 0;
        IF JUSTBOOTED THEN
          BEGIN
            SYMTID := ''; CODETID := ''; WORKTID := '';
            SYMVID := SYVID; CODEVID := SYVID; WORKVID := SYVID
          END;
        IF LENGTH(SYMTID) > 0 THEN
          LTITLE := CONCAT(SYMVID,':',SYMTID)
        ELSE
          LTITLE := '*SYSTEM.WRK.TEXT';
        FOPEN(SYMFIBP^,LTITLE,TRUE,NIL);
        GOTSYM := SYMFIBP^.FISOPEN;
        IF GOTSYM THEN
          BEGIN SYMVID := SYMFIBP^.FVID;
            SYMTID := SYMFIBP^.FHEADER.DTID
          END;
        FCLOSE(SYMFIBP^,CNORMAL);
        IF LENGTH(CODETID) > 0 THEN
          LTITLE := CONCAT(CODEVID,':',CODETID)
        ELSE
          LTITLE := '*SYSTEM.WRK.CODE';
        FOPEN(CODEFIBP^,LTITLE,TRUE,NIL);
        GOTCODE := CODEFIBP^.FISOPEN;
        IF GOTCODE THEN
          BEGIN CODEVID := CODEFIBP^.FVID;
            CODETID := CODEFIBP^.FHEADER.DTID
          END;
        FCLOSE(CODEFIBP^,CNORMAL);
        ALTMODE := SYSCOM^.CRTINFO.ALTMODE;
        SLOWTERM := SYSCOM^.MISCINFO.SLOWTERM;
        STUPID := SYSCOM^.MISCINFO.STUPID
      END
  END (*INITWORKFILE*) ;

PROCEDURE INITFILES;
  BEGIN
    FCLOSE(SWAPFIB^,CNORMAL);
    FCLOSE(USERINFO.SYMFIBP^,CNORMAL);
    FCLOSE(USERINFO.CODEFIBP^,CNORMAL);
    FCLOSE(INPUTFIB^,CNORMAL);
    FCLOSE(OUTPUTFIB^,CNORMAL);
    LTITLE := 'CONSOLE:';
    FOPEN(INPUTFIB^,LTITLE,TRUE,NIL);
    FOPEN(OUTPUTFIB^,LTITLE,TRUE,NIL);
    IF JUSTBOOTED THEN
      BEGIN LTITLE := 'SYSTERM:';
        FOPEN(SYSTERM^,LTITLE,TRUE,NIL)
      END;
    GFILES[0] := INPUTFIB;
    GFILES[1] := OUTPUTFIB;
    GFILES[2] := SYSTERM;
    GFILES[3] := NIL; GFILES[4] := NIL; GFILES[5] := NIL;
  END (*INITFILES*) ;

BEGIN (*INITIALIZE*)
```

```
  JUSTBOOTED := EMPTYHEAP = NIL;
  DOTRITON := FALSE;
  MONTHS[ 0] := '???'; MONTHS[ 1] := 'Jan';
  MONTHS[ 2] := 'Feb'; MONTHS[ 3] := 'Mar';
  MONTHS[ 4] := 'Apr'; MONTHS[ 5] := 'May';
  MONTHS[ 6] := 'Jun'; MONTHS[ 7] := 'Jul';
  MONTHS[ 8] := 'Aug'; MONTHS[ 9] := 'Sep';
  MONTHS[10] := 'Oct'; MONTHS[11] := 'Nov';
  MONTHS[12] := 'Dec'; MONTHS[13] := '???';
  MONTHS[14] := '???'; MONTHS[15] := '???';
  IF JUSTBOOTED THEN INITHEAP
  ELSE RELEASE(EMPTYHEAP);
  INITUNITABLE; (*AND THEDATE*)
  INITFNAMES;
  INITFILES;
  INITWORKFILE;
  IF SYSCOM^.MISCINFO.HAS8510A THEN
    INITCHARSET;
  INITSYSCOM; (*AND SOME GLOBALS*)
  CLEARSCREEN; WRITELN(OUTPUT);
  IF JUSTBOOTED THEN
    BEGIN
      IF DOTRITON THEN
        BEGIN (*ASSUME DATA MEDIA SCREEN*)
          WRITE(OUTPUT,CHR(30),CHR(32),CHR(41));
          UNITWRITE(3,DISPLAY[-80],23)
        END;
      WRITELN(OUTPUT,'Welcome  ',SYVID,',  to');
      IF DOTRITON THEN WRITELN(OUTPUT);
      WRITELN(OUTPUT,'U.C.S.D.  Pascal  System  I.5');
      IF DOTRITON THEN WRITELN(OUTPUT);
      WITH THEDATE DO
        WRITE(OUTPUT,'Current date is  ',DAY,'-',MONTHS[MONTH],'-',YEAR)
    END
  ELSE
    WRITE(OUTPUT,'System re-initialized')
END (*INITIALIZE*) ;

SEGMENT FUNCTION GETCMD(LASTST: CMDSTATE): CMDSTATE;
  CONST ASSEMONLY = LINKANDGO;
  VAR CH: CHAR; BADCMD: BOOLEAN;

  PROCEDURE RUNWORKFILE(OKTOLINK, RUNONLY: BOOLEAN);
    FORWARD;

  FUNCTION ASSOCIATE(TITLE: STRING; OKTOLINK, RUNONLY: BOOLEAN): BOOLEAN;
    LABEL 1;
    VAR RSLT: IORSLTWD; LSEG: SEGRANGE;
        SEGTBL: RECORD
                  DISKINFO: ARRAY [SEGRANGE] OF SEGDESC;
                  SEGNAME: ARRAY [SEGRANGE] OF
                             PACKED ARRAY [0..7] OF CHAR;
                  SEGKIND: ARRAY [SEGRANGE] OF
                             (LINKED,HOSTSEG,SEGPROC,UNITSEG,SEPRTSEG);
                  FILLER: ARRAY [0..143] OF INTEGER
                END { SEGTBL } ;
  BEGIN ASSOCIATE := FALSE;
    FOPEN(USERINFO.CODEFIBP^,TITLE,TRUE,NIL);
    RSLT := SYSCOM^.IORSLT;
    IF RSLT <> INOERROR THEN
```

```
  BEGIN
    IF TITLE <> '*SYSTEM.STARTUP' THEN
      IF RSLT = IBADTITLE THEN
        WRITE(OUTPUT,'Illegal file name')
      ELSE
        WRITE(OUTPUT,'No file ',TITLE);
    GOTO 1
  END;
WITH USERINFO,SYSCOM^ DO
  IF CODEFIBP^.FHEADER.DFKIND <> CODEFILE THEN
    BEGIN
      WRITE(OUTPUT,TITLE,' not code');
      GOTO 1
    END
  ELSE
    BEGIN
      UNITREAD(CODEFIBP^.FUNIT,SEGTBL,SIZEOF(SEGTBL),
                  CODEFIBP^.FHEADER.DFIRSTBLK);
      IF IORESULT <> ORD(INOERROR) THEN
        BEGIN
          WRITE(OUTPUT,'Bad block #0');
          GOTO 1
        END;
      WITH SEGTBL DO
        FOR LSEG := 0 TO MAXSEG DO
          IF (SEGKIND[LSEG]<LINKED) OR (SEGKIND[LSEG]>SEPRTSEG) THEN
            BEGIN { PRE I.5 CODE...FIX UP! }
              FILLCHAR(SEGKIND, SIZEOF(SEGKIND), ORD(LINKED));
              FILLCHAR(FILLER, SIZEOF(FILLER), 0);
              UNITWRITE(CODEFIBP^.FUNIT, SEGTBL, SIZEOF(SEGTBL),
                          CODEFIBP^.FHEADER.DFIRSTBLK)
            END;
      WITH SEGTBL DO
        FOR LSEG := 0 TO MAXSEG DO
          IF SEGKIND[LSEG] <> LINKED THEN
          BEGIN
            IF OKTOLINK THEN
              BEGIN WRITELN(OUTPUT,'Linking...');
                FCLOSE(CODEFIBP^, CNORMAL);
                IF ASSOCIATE(FILENAME[LINKER], FALSE, FALSE) THEN
                  BEGIN
                    IF RUNONLY THEN GETCMD := LINKANDGO
                    ELSE GETCMD := LINKDEBUG;
                    EXIT(GETCMD)
                  END
              END
            ELSE
              IF NOT (LASTST IN [LINKANDGO, LINKDEBUG]) THEN
                WRITE(OUTPUT,'Must L(ink first');
            GOTO 1
          END;
      FOR LSEG := 1 TO MAXSEG DO
        IF (LSEG = 1) OR (LSEG >= 7) THEN
          WITH SEGTABLE[LSEG],SEGTBL.DISKINFO[LSEG] DO
            BEGIN CODEUNIT := CODEFIBP^.FUNIT;
              CODEDESC.CODELENG := CODELENG;
              CODEDESC.DISKADDR := DISKADDR+
                                   CODEFIBP^.FHEADER.DFIRSTBLK
            END
    END;
```

```
    ASSOCIATE := TRUE;
1:  FCLOSE(USERINFO.CODEFIBP^,CNORMAL)
  END (*ASSOCIATE*) ;

  PROCEDURE STARTCOMPILE(NEXTST: CMDSTATE);
    LABEL 1;
    VAR TITLE: STRING[40];
  BEGIN
    IF NEXTST = ASSEMONLY THEN
      WRITE(OUTPUT,'Assembling')
    ELSE
      WRITE(OUTPUT,'Compiling');
    WRITELN(OUTPUT,'...');

    IF NEXTST = ASSEMONLY THEN
      TITLE := FILENAME[ASSMBLER]
    ELSE
      TITLE := FILENAME[COMPILER];
    IF ASSOCIATE(TITLE, FALSE, FALSE) THEN
      WITH USERINFO DO
        BEGIN
            IF GOTSYM THEN
              TITLE := CONCAT(SYMVID,':',SYMTID)
            ELSE
              BEGIN
                IF NEXTST = ASSEMONLY THEN
                  WRITE(OUTPUT, 'Assemble')
                ELSE
                  WRITE(OUTPUT, 'Compile');
                WRITE(OUTPUT,' what text? ');
                READLN(INPUT, TITLE);
                IF TITLE = '' THEN GOTO 1;
                INSERT('.TEXT', TITLE, LENGTH(TITLE)+1);
                GOTCODE := FALSE
              END;
            FOPEN(SYMFIBP^,TITLE,TRUE,NIL);
            IF IORESULT <> ORD(INOERROR) THEN
              BEGIN
                WRITE(OUTPUT,'Can''t find ', TITLE);
                GOTSYM := FALSE; GOTO 1
              END;
            TITLE := '*SYSTEM.SWAPDISK';
            FOPEN(SWAPFIB^,TITLE,TRUE,NIL);
            TITLE := '*SYSTEM.WRK.CODE[*]';
            FOPEN(CODEFIBP^,TITLE,FALSE,NIL);
            IF IORESULT <> ORD(INOERROR) THEN
              BEGIN
                WRITE(OUTPUT,'Code open error!');
                GOTO 1
              END;
            ERRNUM := 0; ERRBLK := 0; ERRSYM := 0;
            IF NEXTST = ASSEMONLY THEN
              NEXTST := COMPONLY;
            GETCMD := NEXTST; EXIT(GETCMD)
        END;
  1:
  END (*STARTCOMPILE*) ;

  PROCEDURE FINISHCOMPILE;
  BEGIN
```

```
  FCLOSE(USERINFO.SYMFIBP^,CNORMAL);
  FCLOSE(SWAPFIB^,CNORMAL);
  IF SYSCOM^.MISCINFO.HAS8510A THEN
    UNITCLEAR(3);
  WITH USERINFO DO
    IF ERRNUM > 0 THEN
      BEGIN GOTCODE := FALSE;
        FCLOSE(CODEFIBP^,CPURGE);
        IF ERRBLK > 0 THEN
          BEGIN CLEARSCREEN; WRITELN(OUTPUT);
            IF ASSOCIATE(FILENAME[EDITOR], FALSE, FALSE) THEN
              BEGIN GETCMD := SYSPROG; EXIT(GETCMD) END
          END
      END
    ELSE
      BEGIN GOTCODE := TRUE;
        CODEVID := CODEFIBP^.FVID;
        CODETID := CODEFIBP^.FHEADER.DTID;
        FCLOSE(CODEFIBP^,CLOCK);
        IF LASTST IN [COMPANDGO,COMPDEBUG] THEN
          RUNWORKFILE(TRUE, LASTST = COMPANDGO)
      END
END (*FINISHCOMPILE*) ;

PROCEDURE EXECUTE;
  VAR TITLE: STRING[255];
BEGIN
  WRITE(OUTPUT,'Execute');
  IF NOT SYSCOM^.MISCINFO.SLOWTERM THEN
    WRITE(OUTPUT,' what file');
  WRITE(OUTPUT,'? '); READLN(TITLE);
  IF LENGTH(TITLE) > 0 THEN
    BEGIN
      IF TITLE[LENGTH(TITLE)] = '.' THEN
        DELETE(TITLE,LENGTH(TITLE),1)
      ELSE
        INSERT('.CODE',TITLE,LENGTH(TITLE)+1);
      IF ASSOCIATE(TITLE, FALSE, FALSE) THEN
        BEGIN GETCMD := SYSPROG; EXIT(GETCMD) END
    END
END (*EXECUTE*) ;

PROCEDURE RUNWORKFILE;
BEGIN
  WITH USERINFO DO
  IF GOTCODE THEN
    BEGIN CLEARSCREEN;
      IF ASSOCIATE(CONCAT(CODEVID,':',CODETID), OKTOLINK, RUNONLY) THEN
        BEGIN
          WRITELN(OUTPUT,'Running...');
          IF RUNONLY THEN
              GETCMD := SYSPROG
          ELSE
              GETCMD := DEBUGCALL;
          EXIT(GETCMD)
        END;
      IF NOT (LASTST IN [LINKANDGO, LINKDEBUG]) THEN
        GOTCODE := FALSE
    END
  ELSE
```

```
      IF RUNONLY THEN
        STARTCOMPILE(COMPANDGO)
      ELSE
        STARTCOMPILE(COMPDEBUG)
  END { RUNWORKFILE } ;

BEGIN (*GETCMD*)
  FRESET(INPUTFIB^); FRESET(OUTPUTFIB^); FRESET(SYSTERM^);
  GFILES[0] := INPUTFIB; GFILES[1] := OUTPUTFIB;
  IF LASTST = HALTINIT THEN
    IF ASSOCIATE('*SYSTEM.STARTUP',FALSE,FALSE) THEN
      BEGIN CLEARSCREEN;
        WRITELN(OUTPUT,'Initializing...');
        GETCMD := SYSPROG; EXIT(GETCMD)
      END;
  IF LASTST IN [COMPONLY,COMPANDGO,COMPDEBUG] THEN
    FINISHCOMPILE;
  IF LASTST IN [LINKANDGO,LINKDEBUG] THEN
    RUNWORKFILE(FALSE, LASTST = LINKANDGO);
  IF SYSCOM^.MISCINFO.USERKIND = AQUIZ THEN
    IF LASTST = HALTINIT THEN
      BEGIN LASTST := COMPANDGO; RUNWORKFILE(TRUE, TRUE) END
    ELSE
      BEGIN
        EMPTYHEAP := NIL;
        GETCMD := HALTINIT;
        EXIT(GETCMD)
      END;
  WITH USERINFO DO
    BEGIN ERRNUM := 0; ERRBLK := 0; ERRSYM := 0 END;
  BADCMD := FALSE;
  REPEAT
    PL :=
'Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, D(ebug,? [I.5e]';
    PROMPT; CH := GETCHAR(BADCMD); CLEARSCREEN;
    IF CH = '?' THEN
      BEGIN PL := 'Command: U(ser restart, I(nitialize, H(alt';
        PROMPT; CH := GETCHAR(BADCMD); CLEARSCREEN
      END;
    BADCMD := NOT (CH IN ['E','R','F','C','L','X','A','D','U','I','H','?']);
    IF NOT BADCMD THEN
      CASE CH OF
        'E':  BEGIN WRITELN(OUTPUT);
                IF ASSOCIATE(FILENAME[EDITOR], FALSE, FALSE) THEN
                  BEGIN GETCMD := SYSPROG; EXIT(GETCMD) END
              END;
        'F':  BEGIN WRITELN(OUTPUT);
                IF ASSOCIATE(FILENAME[FILER], FALSE, FALSE) THEN
                  BEGIN GETCMD := SYSPROG; EXIT(GETCMD) END
              END;
        'L':  BEGIN WRITELN(OUTPUT,'Linking...');
                IF ASSOCIATE(FILENAME[LINKER], FALSE, FALSE) THEN
                  BEGIN GETCMD := SYSPROG; EXIT(GETCMD) END
              END;
        'X':  EXECUTE;
        'C':  STARTCOMPILE(COMPONLY);
        'A':  STARTCOMPILE(ASSEMONLY);
        'U':  IF LASTST <> UPROGNOU THEN
                BEGIN
                  WRITELN(OUTPUT,'Restarting...');
```

```
                GETCMD := SYSPROG; EXIT(GETCMD)
              END
            ELSE
              BEGIN WRITELN(OUTPUT); WRITE(OUTPUT,'U not allowed') END;
    'R','D':  RUNWORKFILE(TRUE, CH = 'R');
    'I','H':  BEGIN
              GETCMD := HALTINIT;
              IF CH = 'H' THEN
                EMPTYHEAP := NIL;
              EXIT(GETCMD)
            END
      END
  UNTIL FALSE
END (*GETCMD*) ;
{ $I SYSTEM.B }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

PROCEDURE EXECERROR;
BEGIN
  WITH SYSCOM^ DO
    BEGIN
      IF XEQERR = 4 THEN
        BEGIN RELEASE(EMPTYHEAP);
          PL := '*STK OFLOW*';
          UNITWRITE(2,PL[1],LENGTH(PL));
          EXIT(COMMAND)
        END;
      BOMBP^.MSIPC := BOMBIPC;
      IF BUGSTATE <> 0 THEN
        BEGIN DEBUGGER; XEQERR := 0 END
      ELSE
        BEGIN RELEASE(EMPTYHEAP);
          GFILES[0] := INPUTFIB; GFILES[1] := OUTPUTFIB;
          BOMBIPC := IORESULT; FWRITELN(SYSTERM^);
          IF UNITABLE[SYSUNIT].UVID = SYVID THEN
            PRINTERROR(XEQERR,BOMBIPC)
          ELSE
            BEGIN
              WRITE(OUTPUT,'Exec err # ',XEQERR);
              IF XEQERR = 10 THEN
                WRITE(OUTPUT,',',BOMBIPC)
            END;
          WRITELN(OUTPUT);
          IF NOT SPACEWAIT(TRUE) THEN EXIT(COMMAND)
        END
    END
END (*EXECERROR*) ;

FUNCTION CHECKDEL(CH: CHAR; VAR SINX: INTEGER): BOOLEAN;
BEGIN CHECKDEL := FALSE;
  WITH SYSCOM^,CRTCTRL DO
    BEGIN
```

```
      IF CH = CRTINFO.LINEDEL THEN
        BEGIN CHECKDEL := TRUE;
          IF (BACKSPACE = CHR(0)) OR (ERASEEOL = CHR(0)) THEN
            BEGIN SINX := 1;
              WRITELN(OUTPUT,'<DEL')
            END
          ELSE
            BEGIN
              WHILE SINX > 1 DO
                BEGIN SINX := SINX-1; WRITE(OUTPUT,BACKSPACE) END;
              WRITE(OUTPUT,ESCAPE,ERASEEOL)
            END
        END;
      IF CH = CRTINFO.CHARDEL THEN
        BEGIN CHECKDEL := TRUE;
          IF SINX > 1 THEN
            BEGIN SINX := SINX-1;
              IF BACKSPACE = CHR(0) THEN
                IF CRTINFO.CHARDEL < ' ' THEN
                  WRITE(OUTPUT,'_')
                ELSE (*ASSUME PRINTABLE*)
              ELSE
                BEGIN
                  IF CRTINFO.CHARDEL <> BACKSPACE THEN
                    WRITE(OUTPUT,BACKSPACE);
                  WRITE(OUTPUT,' ',BACKSPACE)
                END
            END
          ELSE
            IF CRTINFO.CHARDEL = BACKSPACE THEN
              WRITE(OUTPUT,' ')
        END
    END
END (*CHECKDEL*) ;


PROCEDURE PUTPREFIXED(WHICH:INTEGER; COMMANDCHAR:CHAR);
BEGIN
  WITH SYSCOM^ DO
    IF COMMANDCHAR <> CHR(0) THEN
      BEGIN
        IF CRTCTRL.PREFIXED[WHICH] THEN
          WRITE(OUTPUT,CRTCTRL.ESCAPE);
        WRITE(OUTPUT,COMMANDCHAR);
        IF LENGTH(FILLER)>0 THEN
          WRITE(OUTPUT,FILLER);
      END;
END;

PROCEDURE HOMECURSOR;
BEGIN
  PUTPREFIXED(4,SYSCOM^.CRTCTRL.HOME);
END (*HOMECURSOR*) ;

PROCEDURE CLEARSCREEN;
BEGIN HOMECURSOR;
  WITH SYSCOM^,CRTCTRL DO
    BEGIN
      IF MISCINFO.HAS8510A THEN UNITCLEAR(3);
      IF ERASEEOS <> CHR(0) THEN
```

```
        PUTPREFIXED(3,ERASEEOS)
      ELSE
        PUTPREFIXED(6,CLEARSCREEN)
    END
END (*CLEARSCREEN*) ;

PROCEDURE CLEARLINE;
BEGIN
  PUTPREFIXED(2,SYSCOM^.CRTCTRL.ERASEEOL)
END (*CLEARLINE*) ;

PROCEDURE PROMPT;
  VAR I: INTEGER;
BEGIN HOMECURSOR;
  WITH SYSCOM^ DO
    BEGIN
      CLEARLINE;
      IF MISCINFO.SLOWTERM THEN
        BEGIN
          I := SCAN(LENGTH(PL),=':',PL[1]);
          IF I <> LENGTH(PL) THEN PL[0] := CHR(I+1)
        END
    END;
  WRITE(OUTPUT,PL)
END (*PROMPT*) ;

PROCEDURE FGOTOXY(*X,Y: INTEGER*);
BEGIN (*ASSUME DATA MEDIA*)
  WITH SYSCOM^.CRTINFO DO
    BEGIN
      IF X < 0 THEN X := 0;
      IF X > WIDTH THEN X := WIDTH;
      IF Y < 0 THEN Y := 0;
      IF Y > HEIGHT THEN Y := HEIGHT
    END;
  WRITE(OUTPUT,CHR(30),CHR(X+32),CHR(Y+32))
END (*GOTOXY*) ;

FUNCTION GETCHAR(*FLUSH: BOOLEAN*);
  VAR CH: CHAR;
BEGIN
  IF FLUSH THEN UNITCLEAR(1);
  IF INPUTFIB^.FEOF THEN EXIT(COMMAND);
  INPUTFIB^.FSTATE := FNEEDCHAR;
  READ(INPUT,CH);
  IF (CH >= 'a') AND (CH <= 'z') THEN
    CH := CHR(ORD(CH)-ORD('a')+ORD('A'));
  GETCHAR := CH
END (*GETCHAR*) ;

FUNCTION SPACEWAIT(*FLUSH: BOOLEAN*);
  VAR CH: CHAR;
BEGIN
  REPEAT
    WRITE(OUTPUT,'Type <space>');
    IF NOT SYSCOM^.MISCINFO.SLOWTERM THEN
      WRITE(OUTPUT,' to continue');
    CH := GETCHAR(FLUSH);
    IF NOT EOLN(INPUT) THEN
      WRITELN(OUTPUT);
```

```
    CLEARLINE
  UNTIL (CH = ' ') OR (CH = SYSCOM^.CRTINFO.ALTMODE);
  SPACEWAIT := CH <> ' '
END (*SPACEWAIT*) ;

FUNCTION SCANTITLE(*FTITLE: STRING; VAR FVID: VID; VAR FTID: TID;
                    VAR FSEGS: INTEGER; VAR FKIND: FILEKIND*);
  VAR I,RBRACK: INTEGER; CH: CHAR; OK: BOOLEAN;
BEGIN
  FVID := ''; FTID := '';
  FSEGS := 0; FKIND := UNTYPEDFILE;
  SCANTITLE := FALSE; I := 1;
  WHILE I <= LENGTH(FTITLE) DO
    BEGIN CH := FTITLE[I];
      IF CH <= ' ' THEN DELETE(FTITLE,I,1)
      ELSE
        BEGIN
          IF (CH >= 'a') AND (CH <= 'z') THEN
            FTITLE[I] := CHR(ORD(CH)-ORD('a')+ORD('A'));
          I := I+1
        END
    END;
  IF LENGTH(FTITLE) > 0 THEN
    BEGIN
      IF FTITLE[1] = '*' THEN
        BEGIN FVID := SYVID; DELETE(FTITLE,1,1) END;
      I := POS(':',FTITLE);
      IF I <= 1 THEN
        BEGIN
          IF LENGTH(FVID) = 0 THEN FVID := DKVID;
          IF I = 1 THEN DELETE(FTITLE,1,1)
        END
      ELSE
        IF I-1 <= VIDLENG THEN
          BEGIN
            FVID := COPY(FTITLE,1,I-1);
            DELETE(FTITLE,1,I)
          END;
      IF LENGTH(FVID) > 0 THEN
        BEGIN
          I := POS('[',FTITLE);
          IF I > 0 THEN I := I-1
          ELSE I := LENGTH(FTITLE);
          IF I <= TIDLENG THEN
            BEGIN
              IF I > 0 THEN
                BEGIN FTID := COPY(FTITLE,1,I); DELETE(FTITLE,1,I) END;
              IF LENGTH(FTITLE) = 0 THEN OK := TRUE
              ELSE
                BEGIN OK := FALSE;
                  RBRACK := POS(']',FTITLE);
                  IF RBRACK = 2 THEN OK := TRUE
                  ELSE
                    IF RBRACK > 2 THEN
                      BEGIN OK := TRUE; I := 2;
                        REPEAT CH := FTITLE[I];
                          IF CH IN DIGITS THEN
                            FSEGS := FSEGS*10+(ORD(CH)-ORD('0'))
                          ELSE OK := FALSE;
                          I := I+1
```

```
                    UNTIL (I = RBRACK) OR NOT OK;
                    IF (I = 3) AND (RBRACK = 3) THEN
                      IF FTITLE[I-1] = '*' THEN
                        BEGIN FSEGS := -1; OK := TRUE END
                  END
              END;
            SCANTITLE := OK;
            IF OK AND (LENGTH(FTID) > 5) THEN
              BEGIN
                FTITLE := COPY(FTID,LENGTH(FTID)-4,5);
                IF FTITLE = '.TEXT' THEN FKIND := TEXTFILE
                ELSE
                IF FTITLE = '.CODE' THEN FKIND := CODEFILE
                ELSE
                IF FTITLE = '.INFO' THEN FKIND := INFOFILE
                ELSE
                IF FTITLE = '.GRAF' THEN FKIND := GRAFFILE
                ELSE
                IF FTITLE = '.FOTO' THEN FKIND := FOTOFILE
              END
          END
      END
  END (*SCANTITLE*) ;

  (* VOLUME AND DIRECTORY HANDLERS *)

  FUNCTION FETCHDIR(FUNIT: UNITNUM): BOOLEAN;
    VAR LINX: DIRRANGE; OK: BOOLEAN; HNOW: INTEGER;
  BEGIN FETCHDIR := FALSE;
    WITH SYSCOM^,UNITABLE[FUNIT] DO
      BEGIN (*READ IN AND VALIDATE DIR*)
        IF GDIRP = NIL THEN NEW(GDIRP);
        UNITREAD(FUNIT,GDIRP^,SIZEOF(DIRECTORY),DIRBLK);
        OK := IORSLT = INOERROR;
        IF OK THEN
          WITH GDIRP^[0] DO
            BEGIN OK := FALSE; (*CHECK OUT DIR*)
              IF (DFIRSTBLK = 0) AND
                  (  (MISCINFO.USERKIND=BOOKER)
                OR ( (MISCINFO.USERKIND IN [AQUIZ,PQUIZ]) AND (DFKIND=SECUREDIR) )
                    OR ( (MISCINFO.USERKIND=NORMAL) AND (DFKIND=UNTYPEDFILE) ) )
                THEN
                IF (LENGTH(DVID) > 0) AND (LENGTH(DVID) <= VIDLENG) AND
                   (DNUMFILES >= 0) AND (DNUMFILES <= MAXDIR) THEN
                  BEGIN OK := TRUE; (*SO FAR SO GOOD*)
                    IF DVID <> UVID THEN
                      BEGIN (*NEW VOLUME IN UNIT...CAREFUL*)
                        LINX := 1;
                        WHILE LINX <= DNUMFILES DO
                          WITH GDIRP^[LINX] DO
                            IF (LENGTH(DTID) <= 0) OR
                               (LENGTH(DTID) > TIDLENG) OR
                               (DLASTBLK < DFIRSTBLK) OR
                               (DLASTBYTE > FBLKSIZE) OR
                               (DLASTBYTE <= 0) OR
                               (DACCESS.YEAR >= 100) THEN
                              BEGIN OK := FALSE; DELENTRY(LINX,GDIRP) END
                            ELSE
                              LINX := LINX+1;
```

```
                    IF NOT OK THEN
                      BEGIN (*MUST HAVE BEEN CHANGED...WRITEIT*)
                        UNITWRITE(FUNIT,GDIRP^,
                              (DNUMFILES+1)*SIZEOF(DIRENTRY),DIRBLK);
                        OK := IORSLT = INOERROR
                      END
                  END
              END;
            IF OK THEN
              BEGIN UVID := DVID; UEOVBLK := DEOVBLK;
                TIME(HNOW,DLOADTIME)
              END
          END;
      FETCHDIR := OK;
      IF NOT OK THEN
        BEGIN UVID := ''; UEOVBLK := MMAXINT;
          RELEASE(GDIRP); GDIRP := NIL
        END
    END
END (*FETCHDIR*) ;

PROCEDURE WRITEDIR(*FUNIT: UNITNUM; FDIR: DIRP*);
  VAR HNOW,LNOW: INTEGER; OK: BOOLEAN; LDE: DIRENTRY;
BEGIN
  WITH UNITABLE[FUNIT],FDIR^[0] DO
    BEGIN OK := (UVID = DVID) AND ((DFKIND = UNTYPEDFILE) OR
                                   (DFKIND = SECUREDIR));
      IF OK THEN
        BEGIN TIME(HNOW,LNOW);
          OK := (LNOW-DLOADTIME <= AGELIMIT) AND
                ((LNOW-DLOADTIME) >= 0) AND
                    SYSCOM^.MISCINFO.HASCLOCK;
          IF NOT OK THEN
            BEGIN (*NO CLOCK OR TOO OLD*)
              UNITREAD(FUNIT,LDE,SIZEOF(DIRENTRY),DIRBLK);
              IF IORESULT = ORD(INOERROR) THEN
                OK := DVID = LDE.DVID;
            END;

          IF OK THEN
            BEGIN (*WE GUESS ALL IS SAFE...WRITEIT*)
              DFIRSTBLK := 0;   (*DIRTY FIX FOR YALOE BUGS*)
              UNITWRITE(FUNIT,FDIR^,
                        (DNUMFILES+1)*SIZEOF(DIRENTRY),DIRBLK);
              OK := IORESULT = ORD(INOERROR);
              IF DLASTBLK = 10 THEN (*REDUNDANT AFTERTHOUGHT*)
                UNITWRITE(FUNIT,FDIR^,
                          (DNUMFILES+1)*SIZEOF(DIRENTRY),6);
              IF OK THEN TIME(HNOW,DLOADTIME)
            END
        END;
      IF NOT OK THEN
        BEGIN SYSCOM^.IORSLT := ILOSTUNIT;
          UVID := ''; UEOVBLK := MMAXINT
        END
    END
END (*WRITEDIR*) ;

FUNCTION VOLSEARCH(*VAR FVID: VID; LOOKHARD: BOOLEAN; VAR FDIR: DIRP*);
  VAR LUNIT: UNITNUM; OK,PHYSUNIT: BOOLEAN; HNOW,LNOW: INTEGER;
```

```
BEGIN VOLSEARCH := 0; FDIR := NIL;
  OK := FALSE; PHYSUNIT := FALSE;
  IF LENGTH(FVID) > 0 THEN
    BEGIN
      IF (FVID[1] = '#') AND (LENGTH(FVID) > 1) THEN
        BEGIN OK := TRUE;
          LUNIT := 0; HNOW := 2;
          REPEAT
            IF FVID[HNOW] IN DIGITS THEN
              LUNIT := LUNIT*10+ORD(FVID[HNOW])-ORD('0')
            ELSE OK := FALSE;
            HNOW := HNOW+1
          UNTIL (HNOW > LENGTH(FVID)) OR NOT OK;
          PHYSUNIT := OK AND (LUNIT > 0) AND (LUNIT <= MAXUNIT)
        END;
      IF NOT PHYSUNIT THEN
        BEGIN OK := FALSE; LUNIT := MAXUNIT;
          REPEAT
            OK := FVID = UNITABLE[LUNIT].UVID;
            IF NOT OK THEN LUNIT := LUNIT-1
          UNTIL OK OR (LUNIT = 0)
        END
    END;
  IF OK THEN
    IF UNITABLE[LUNIT].UISBLKD THEN
      WITH SYSCOM^ DO
        BEGIN OK := FALSE;  (*SEE IF GDIRP IS GOOD*)
          IF GDIRP <> NIL THEN
            IF FVID = GDIRP^[0].DVID THEN
              BEGIN TIME(HNOW,LNOW);
                OK := LNOW-GDIRP^[0].DLOADTIME <= AGELIMIT
              END;
          IF NOT OK THEN
            BEGIN OK := PHYSUNIT;
              IF FETCHDIR(LUNIT) THEN
                IF NOT PHYSUNIT THEN
                  OK := FVID = GDIRP^[0].DVID
            END
        END;
  IF NOT OK AND LOOKHARD THEN
    BEGIN LUNIT := MAXUNIT; (*CHECK EACH DISK UNIT*)
      REPEAT
        WITH UNITABLE[LUNIT] DO
          IF UISBLKD THEN
            IF FETCHDIR(LUNIT) THEN
              OK := FVID = UVID;
        IF NOT OK THEN LUNIT := LUNIT-1
      UNTIL OK OR (LUNIT = 0)
    END;
  IF OK THEN
    WITH UNITABLE[LUNIT] DO
      BEGIN VOLSEARCH := LUNIT;
        IF LENGTH(UVID) > 0 THEN FVID := UVID;
        IF UISBLKD AND (SYSCOM^.GDIRP <> NIL) THEN
          BEGIN FDIR := SYSCOM^.GDIRP;
            TIME(HNOW,FDIR^[0].DLOADTIME)
          END
      END
END (*VOLSEARCH*) ;
```

```
FUNCTION DIRSEARCH(*VAR FTID: TID; FINDPERM: BOOLEAN; FDIR: DIRP*);
  VAR I: DIRRANGE; FOUND: BOOLEAN;
BEGIN DIRSEARCH := 0; FOUND := FALSE; I := 1;
  WHILE (I <= FDIR^[0].DNUMFILES) AND NOT FOUND DO
    BEGIN
      WITH FDIR^[I] DO
        IF DTID = FTID THEN
          IF FINDPERM = (DACCESS.YEAR <> 100) THEN
            BEGIN DIRSEARCH := I; FOUND := TRUE END;
      I := I+1
    END
END (*DIRSEARCH*) ;

PROCEDURE DELENTRY(*FINX: DIRRANGE; FDIR: DIRP*);
  VAR I: DIRRANGE;
BEGIN
  WITH FDIR^[0] DO
    BEGIN
      FOR I := FINX TO DNUMFILES-1 DO
        FDIR^[I] := FDIR^[I+1];
      FDIR^[DNUMFILES].DTID := '';
      DNUMFILES := DNUMFILES-1
    END
END (*DELENTRY*) ;

PROCEDURE INSENTRY(*VAR FENTRY: DIRENTRY; FINX: DIRRANGE; FDIR: DIRP*);
  VAR I: DIRRANGE;
BEGIN
  WITH FDIR^[0] DO
    BEGIN
      FOR I := DNUMFILES DOWNTO FINX DO
        FDIR^[I+1] := FDIR^[I];
      FDIR^[FINX] := FENTRY;
      DNUMFILES := DNUMFILES+1
    END
END (*INSENTRY*) ;

FUNCTION ENTERTEMP(VAR FTID: TID; FSEGS: INTEGER;
                   FKIND: FILEKIND; FDIR: DIRP): DIRRANGE;
  VAR I,LASTI,DINX,SINX: DIRRANGE; RT11ISH: BOOLEAN;
      SSEGS: INTEGER; LDE: DIRENTRY;

  PROCEDURE FINDMAX(CURINX: DIRRANGE; FIRSTOPEN,NEXTUSED: INTEGER);
    VAR FREEAREA: INTEGER;
  BEGIN
    FREEAREA := NEXTUSED-FIRSTOPEN;
    IF FREEAREA > FSEGS THEN
      BEGIN
        SINX := DINX; SSEGS := FSEGS;
        DINX := CURINX; FSEGS := FREEAREA
      END
    ELSE
      IF FREEAREA > SSEGS THEN
        BEGIN SSEGS := FREEAREA; SINX := CURINX END
  END (*FINDMAX*) ;

BEGIN (*ENTERTEMP*)
  DINX := 0; LASTI := FDIR^[0].DNUMFILES;
  SINX := 0; SSEGS := 0;
  IF FSEGS <= 0 THEN
```

```
    BEGIN RT11ISH := FSEGS < 0;
      FOR I := 1 TO LASTI DO
        FINDMAX(I,FDIR^[I-1].DLASTBLK,FDIR^[I].DFIRSTBLK);
      FINDMAX(LASTI+1,FDIR^[LASTI].DLASTBLK,FDIR^[0].DEOVBLK);
      IF RT11ISH THEN
        IF FSEGS DIV 2 <= SSEGS THEN
          BEGIN FSEGS := SSEGS; DINX := SINX END
        ELSE FSEGS := (FSEGS+1) DIV 2
    END
  ELSE
    BEGIN I := 1;
      WHILE I <= LASTI DO
        BEGIN
          IF FDIR^[I].DFIRSTBLK-FDIR^[I-1].DLASTBLK >= FSEGS THEN
            BEGIN DINX := I; I := LASTI END;
          I := I+1
        END;
      IF DINX = 0 THEN
        IF FDIR^[0].DEOVBLK-FDIR^[LASTI].DLASTBLK >= FSEGS THEN
          DINX := LASTI+1
    END;
  IF LASTI = MAXDIR THEN DINX := 0;
  IF DINX > 0 THEN
    BEGIN
      WITH LDE DO
        BEGIN
          DFIRSTBLK := FDIR^[DINX-1].DLASTBLK;
          DLASTBLK := DFIRSTBLK+FSEGS;
          DFKIND := FKIND; DTID := FTID;
          DLASTBYTE := FBLKSIZE;
          WITH DACCESS DO
            BEGIN MONTH := 0; DAY := 0; YEAR := 100 END
        END;
      INSENTRY(LDE,DINX,FDIR)
    END;
  ENTERTEMP := DINX
END (*ENTERTEMP*) ;

(* FILE STATE HANDLERS *)

PROCEDURE FINIT(*VAR F: FIB; WINDOW: WINDOWP; RECWORDS: INTEGER*);
BEGIN
  WITH F DO
    BEGIN FSTATE := FJANDW;
      FISOPEN := FALSE; FEOF := TRUE;
      FEOLN := TRUE; FWINDOW := WINDOW;
      IF (RECWORDS = 0) OR (RECWORDS = -2) THEN
        BEGIN
          FWINDOW^[1] := CHR(0); FRECSIZE := 1;
          IF RECWORDS = 0 THEN FSTATE := FNEEDCHAR
        END
      ELSE
        IF RECWORDS < 0 THEN
          BEGIN FWINDOW := NIL; FRECSIZE := 0 END
        ELSE FRECSIZE := RECWORDS+RECWORDS
    END
END (*FINIT*) ;

PROCEDURE RESETER(VAR F:FIB);
  VAR BIGGER: BOOLEAN;
```

```
BEGIN
  WITH F DO
     BEGIN FREPTCNT := 0;
        FEOLN := FALSE; FEOF := FALSE;
        IF FISBLKD THEN
          BEGIN BIGGER := FNXTBLK > FMAXBLK;
            IF BIGGER THEN FMAXBLK := FNXTBLK;
            IF FSOFTBUF THEN
              BEGIN
                IF BIGGER THEN FMAXBYTE := FNXTBYTE
                ELSE
                  IF FNXTBLK = FMAXBLK THEN
                    IF FNXTBYTE > FMAXBYTE THEN
                        BEGIN BIGGER := TRUE; FMAXBYTE := FNXTBYTE END;
                IF FBUFCHNGD THEN
                  BEGIN FBUFCHNGD := FALSE; FMODIFIED := TRUE;
                    IF BIGGER THEN
                      FILLCHAR(FBUFFER[FNXTBYTE],FBLKSIZE-FNXTBYTE,0);
                    UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,
                                    FHEADER.DFIRSTBLK+FNXTBLK-1);
                    IF BIGGER AND (FHEADER.DFKIND = TEXTFILE)
                        AND ODD(FNXTBLK) THEN
                      BEGIN FMAXBLK := FMAXBLK+1;
                        FILLCHAR(FBUFFER,FBLKSIZE,0);
                        UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,
                                        FHEADER.DFIRSTBLK+FNXTBLK)
                      END
                  END;
                FNXTBYTE := FBLKSIZE
              END;
            FNXTBLK := 0;
            IF FSOFTBUF AND (FHEADER.DFKIND = TEXTFILE) THEN
                FNXTBLK := 2
          END
     END
END (*RESETER*) ;

PROCEDURE FOPEN(*VAR F: FIB; VAR FTITLE: STRING;
                 FOPENOLD: BOOLEAN; JUNK PARAM*);
  LABEL 1;
  VAR LDIR: DIRP; LUNIT: UNITNUM; LINX: DIRRANGE;
      LSEGS,NBYTES: INTEGER; LKIND: FILEKIND;
      OLDHEAP: ^INTEGER; SWAPPED: BOOLEAN;
      SAVERSLT: IORSLTWD; LVID: VID; LTID: TID;
BEGIN SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN THEN SYSCOM^.IORSLT := INOTCLOSED
    ELSE
      IF SCANTITLE(FTITLE,LVID,LTID,LSEGS,LKIND) THEN
        BEGIN (*GOT AN OK TITLE*)
          IF ORD(FOPENOLD) > 1 THEN (*OLD CODE SPECIAL CASE*)
            FOPENOLD := (ORD(FOPENOLD) = 2) OR (ORD(FOPENOLD) = 4);
          SWAPPED := FALSE;
          WITH SWAPFIB^ DO
            IF FISOPEN AND (SYSCOM^.GDIRP = NIL) THEN
              BEGIN MARK(OLDHEAP);
                NBYTES := ORD(SYSCOM^.LASTMP)-ORD(OLDHEAP);
                IF (NBYTES > 0) AND (NBYTES < SIZEOF(DIRECTORY)+400) THEN
                  BEGIN
                    NBYTES := ORD(OLDHEAP)-ORD(EMPTYHEAP);
```

```
            IF (NBYTES > 0) AND (NBYTES > SIZEOF(DIRECTORY)) AND
                (UNITABLE[FUNIT].UVID = FVID) THEN
              BEGIN
                UNITWRITE(FUNIT,EMPTYHEAP^,SIZEOF(DIRECTORY),
                                FHEADER.DFIRSTBLK);
                RELEASE(EMPTYHEAP); SWAPPED := TRUE
              END
          END
      END;
    LUNIT := VOLSEARCH(LVID,TRUE,LDIR);
    IF LUNIT = 0 THEN SYSCOM^.IORSLT := INOUNIT
    ELSE
      WITH UNITABLE[LUNIT] DO
        BEGIN (*OK...OPEN UP FILE*)
          FISOPEN := TRUE; FMODIFIED := FALSE;
          FUNIT := LUNIT; FVID := LVID;
          FNXTBLK := 0; FISBLKD := UISBLKD;
          FSOFTBUF := UISBLKD AND (FRECSIZE <> 0);
          IF (LDIR <> NIL) AND (LENGTH(LTID) > 0) THEN
            BEGIN (*LOOKUP OR ENTER FHEADER IN DIRECTORY*)
              LINX := DIRSEARCH(LTID,FOPENOLD,LDIR);
              IF FOPENOLD THEN
                IF LINX = 0 THEN
                  BEGIN SYSCOM^.IORSLT := INOFILE; GOTO 1 END
                ELSE FHEADER := LDIR^[LINX]
              ELSE (*OPEN NEW FILE*)
                IF LINX > 0 THEN
                  BEGIN SYSCOM^.IORSLT := IDUPFILE; GOTO 1 END
                ELSE
                  BEGIN (*MAKE A TEMP ENTRY*)
                    IF LKIND = UNTYPEDFILE THEN LKIND := DATAFILE;
                    LINX := ENTERTEMP(LTID,LSEGS,LKIND,LDIR);
                    IF (LINX > 0) AND (LKIND = TEXTFILE) THEN
                      WITH LDIR^[LINX] DO
                        BEGIN
                          IF ODD(DLASTBLK-DFIRSTBLK) THEN
                            DLASTBLK := DLASTBLK-1;
                          IF DLASTBLK-DFIRSTBLK < 4 THEN
                            BEGIN DELENTRY(LINX,LDIR); LINX := 0 END
                        END;
                    IF LINX = 0 THEN
                      BEGIN SYSCOM^.IORSLT := INOROOM; GOTO 1 END;
                    FHEADER := LDIR^[LINX]; FMODIFIED := TRUE;
                    WRITEDIR(LUNIT,LDIR)
                  END
            END
          ELSE (*FHEADER NOT IN DIRECTORY*)
            WITH FHEADER DO
              BEGIN (*DIRECT UNIT OPEN, SET UP DUMMY FHEADER*)
                DFIRSTBLK := 0; DLASTBLK := MMAXINT;
                IF UISBLKD THEN DLASTBLK := UEOVBLK;
                DFKIND := LKIND; DTID := '';
                DLASTBYTE := FBLKSIZE;
                WITH DACCESS DO
                  BEGIN MONTH := 0; DAY := 0; YEAR := 0 END
              END;
          IF FOPENOLD THEN
            FMAXBLK := FHEADER.DLASTBLK-FHEADER.DFIRSTBLK
          ELSE FMAXBLK := 0;
          IF FSOFTBUF THEN
```

```
                BEGIN
                  FNXTBYTE := FBLKSIZE; FBUFCHNGD := FALSE;
                  IF FOPENOLD THEN FMAXBYTE := FHEADER.DLASTBYTE
                  ELSE FMAXBYTE := FBLKSIZE;
                  WITH FHEADER DO
                    IF DFKIND = TEXTFILE THEN
                      BEGIN FNXTBLK := 2;
                        IF NOT FOPENOLD THEN
                          BEGIN (*NEW .TEXT, PUT NULLS IN FIRST PAGE*)
                            FILLCHAR(FBUFFER,SIZEOF(FBUFFER),0);
                            UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK);
                            UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+1)
                          END
                      END
                END;
              IF FOPENOLD THEN FRESET(F)
              ELSE RESETER(F); (*NO GET!*)
      1:      IF IORESULT <> ORD(INOERROR) THEN
                BEGIN FISOPEN := FALSE; FEOF := TRUE; FEOLN := TRUE END
            END;
          IF SWAPPED THEN
            BEGIN RELEASE(OLDHEAP); SYSCOM^.GDIRP := NIL;
              SAVERSLT := SYSCOM^.IORSLT;
              UNITREAD(SWAPFIB^.FUNIT,EMPTYHEAP^,SIZEOF(DIRECTORY),
                                SWAPFIB^.FHEADER.DFIRSTBLK);
              SYSCOM^.IORSLT := SAVERSLT
            END
        END
      ELSE SYSCOM^.IORSLT := IBADTITLE
END (*FOPEN*) ;

PROCEDURE FCLOSE(*VAR F: FIB; FTYPE: CLOSETYPE*);
  LABEL 1;
  VAR LINX,DUPINX: DIRRANGE; LDIR: DIRP; FOUND: BOOLEAN;
BEGIN SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN AND (FWINDOW <> SYSTERM^.FWINDOW) THEN
      BEGIN
        IF FISBLKD THEN
          WITH FHEADER DO
            IF LENGTH(DTID) > 0 THEN
              BEGIN (*FILE IN A DISK DIRECTORY...FIXUP MAYBE*)
                IF FTYPE = CCRUNCH THEN
                  BEGIN FMAXBLK := FNXTBLK;
                    DACCESS.YEAR := 100; FTYPE := CLOCK;
                    IF FSOFTBUF THEN FMAXBYTE := FNXTBYTE
                  END;
                RESETER(F);
                IF FMODIFIED OR (DACCESS.YEAR = 100) OR (FTYPE = CPURGE) THEN
                  BEGIN (*HAVE TO CHANGE DIRECTORY ENTRY*)
                    IF FUNIT <> VOLSEARCH(FVID,FALSE,LDIR) THEN
                      BEGIN SYSCOM^.IORSLT := ILOSTUNIT; GOTO 1 END;
                    LINX := 1; FOUND := FALSE;
                    WHILE (LINX <= LDIR^[0].DNUMFILES) AND NOT FOUND DO
                      BEGIN (*LOOK FOR FIRST BLOCK MATCH*)
                        FOUND := (LDIR^[LINX].DFIRSTBLK = DFIRSTBLK) AND
                                 (LDIR^[LINX].DLASTBLK = DLASTBLK);
                        LINX := LINX + 1
                      END;
                    IF NOT FOUND THEN
```

```
                    BEGIN SYSCOM^.IORSLT := ILOSTFILE; GOTO 1 END;
                  LINX := LINX - 1; (*CORRECT OVERRUN*)
                  IF ((FTYPE = CNORMAL) AND (LDIR^[LINX].DACCESS.YEAR = 100))
                        OR (FTYPE = CPURGE) THEN
                      DELENTRY(LINX,LDIR)  (*ZAP FILE OUT OF EXISTANCE*)
                  ELSE
                    BEGIN (*WELL...LOCK IN A PERM DIR ENTRY*)
                      DUPINX := DIRSEARCH(DTID,TRUE,LDIR);
                      IF (DUPINX <> 0) AND (DUPINX <> LINX) THEN
                        BEGIN (*A DUPLICATE PERM ENTRY...ZAP OLD ONE*)
                          DELENTRY(DUPINX,LDIR);
                          IF DUPINX < LINX THEN LINX := LINX-1
                        END;
                      IF LDIR^[LINX].DACCESS.YEAR = 100 THEN
                        IF DACCESS.YEAR = 100 THEN
                          DACCESS := THEDATE
                        ELSE (*LEAVE ALONE...FILER SPECIAL CASE*)
                      ELSE
                        IF FMODIFIED AND (THEDATE.MONTH <> 0) THEN
                          DACCESS := THEDATE
                        ELSE
                          DACCESS := LDIR^[LINX].DACCESS;
                      DLASTBLK := DFIRSTBLK+FMAXBLK;
                      IF FSOFTBUF THEN DLASTBYTE := FMAXBYTE;
                      FMODIFIED := FALSE; LDIR^[LINX] := FHEADER
                    END;
                  WRITEDIR(FUNIT,LDIR)
                END
              END;
          IF FTYPE = CPURGE THEN
            IF LENGTH(FHEADER.DTID) = 0 THEN
              UNITABLE[FUNIT].UVID := '';
1:        FEOF := TRUE; FEOLN := TRUE; FISOPEN := FALSE
        END
END (*FCLOSE*) ;
{ $I SYSTEM.C }

    (*****************************************************************)
    (*                                                               *)
    (*  Copyright (c) 1978 Regents of the University of California.  *)
    (*  Permission to copy or distribute this software or documen-   *)
    (*  tation in hard or soft copy granted only by written license  *)
    (*  obtained from the Institute for Information Systems.         *)
    (*                                                               *)
    (*****************************************************************)

(* INPUT-OUTPUT PRIMITIVES *)

PROCEDURE XSEEK;
BEGIN
  SYSCOM^.XEQERR := 11; { NOT IMP ERR }
  EXECERROR
END (*XSEEK*) ;

PROCEDURE XREADREAL;
BEGIN
  SYSCOM^.XEQERR := 11; { NOT IMP ERR }
  EXECERROR
END (*XREADREAL*) ;
```

```
PROCEDURE XWRITEREAL;
BEGIN
  SYSCOM^.XEQERR := 11; { NOT IMP ERR }
  EXECERROR
END (*XWRITEREAL*) ;

FUNCTION CANTSTRETCH(VAR F: FIB): BOOLEAN; (*REPLACED BY RJH 2Mar78*)
  LABEL 1;
  VAR LINX: DIRRANGE; FOUND,OK: BOOLEAN; LAVAILBLK: INTEGER; LDIR: DIRP;
BEGIN CANTSTRETCH := TRUE; OK := FALSE;

  WITH F,FHEADER DO
    IF LENGTH(DTID) > 0 THEN
      BEGIN (*IN A DIRECTORY FOR SURE*)
        IF FUNIT <> VOLSEARCH(FVID,FALSE,LDIR) THEN
          BEGIN SYSCOM^.IORSLT := ILOSTUNIT; GOTO 1 END;
        FOUND := FALSE; LINX := 1;
        WHILE (LINX <= LDIR^[0].DNUMFILES) AND NOT FOUND DO
          BEGIN
            FOUND := (LDIR^[LINX].DFIRSTBLK = DFIRSTBLK) AND
                     (LDIR^[LINX].DLASTBLK = DLASTBLK);
            LINX := LINX+1
          END;
        IF NOT FOUND THEN
          BEGIN SYSCOM^.IORSLT := ILOSTFILE; GOTO 1 END;
        IF LINX > LDIR^[0].DNUMFILES THEN
          LAVAILBLK := LDIR^[0].DEOVBLK
        ELSE LAVAILBLK := LDIR^[LINX].DFIRSTBLK;
        IF (DLASTBLK < LAVAILBLK) OR (DLASTBYTE < FBLKSIZE) THEN
          BEGIN
            WITH LDIR^[LINX-1] DO
              BEGIN
                DLASTBLK := LAVAILBLK; DLASTBYTE := FBLKSIZE;
                WRITEDIR(FUNIT,LDIR);
                IF IORESULT <> ORD(INOERROR) THEN GOTO 1
              END;
            FEOF := FALSE; FEOLN := FALSE;
            IF FSTATE <> FJANDW THEN FSTATE := FNEEDCHAR; (*RJH 2Mar78*)
            DLASTBLK := LAVAILBLK; DLASTBYTE := FBLKSIZE;
            DACCESS.YEAR := 100; CANTSTRETCH := FALSE
          END;
        OK := TRUE;
      END;
1:  IF NOT OK THEN
      BEGIN F.FEOF := TRUE; F.FEOLN := TRUE END
END (*CANTSTRETCH*) ;

PROCEDURE FRESET(*VAR F: FIB*);
BEGIN SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN THEN
      BEGIN RESETER(F);
        IF FRECSIZE > 0 THEN
          IF FSTATE = FJANDW THEN FGET(F)
          ELSE FSTATE := FNEEDCHAR
      END
END (*FRESET*) ;

FUNCTION FBLOCKIO(*VAR F: FIB; VAR A: WINDOW;
                   NBLOCKS,RBLOCK: INTEGER; DOREAD: BOOLEAN*);
```

```
BEGIN FBLOCKIO := 0; SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN AND (NBLOCKS >= 0) THEN
      IF FISBLKD THEN
        WITH FHEADER DO
          BEGIN
            IF RBLOCK < 0 THEN RBLOCK := FNXTBLK;
            RBLOCK := DFIRSTBLK+RBLOCK;
            IF RBLOCK+NBLOCKS > DLASTBLK THEN
              IF NOT DOREAD THEN
                IF CANTSTRETCH( F ) THEN;
            IF RBLOCK+NBLOCKS > DLASTBLK THEN
              NBLOCKS := DLASTBLK-RBLOCK;
            FEOF := RBLOCK >= DLASTBLK;
            IF NOT FEOF THEN
              BEGIN
                IF DOREAD THEN
                  UNITREAD(FUNIT,A,NBLOCKS*FBLKSIZE,RBLOCK)
                ELSE
                  BEGIN FMODIFIED := TRUE;
                    UNITWRITE(FUNIT,A,NBLOCKS*FBLKSIZE,RBLOCK)
                  END;
                FBLOCKIO := NBLOCKS;
                RBLOCK := RBLOCK+NBLOCKS;
                FEOF := RBLOCK = DLASTBLK;
                FNXTBLK := RBLOCK-DFIRSTBLK;
                IF FNXTBLK > FMAXBLK THEN FMAXBLK := FNXTBLK
              END
          END
      ELSE
        BEGIN FBLOCKIO := NBLOCKS;
          IF DOREAD THEN
            UNITREAD(FUNIT,A,NBLOCKS*FBLKSIZE,RBLOCK)
          ELSE
            UNITWRITE(FUNIT,A,NBLOCKS*FBLKSIZE,RBLOCK);
          IF IORESULT = ORD(INOERROR) THEN
            IF DOREAD THEN
              BEGIN RBLOCK := NBLOCKS*FBLKSIZE;
                RBLOCK := RBLOCK+SCAN(-RBLOCK,<>CHR(0),A[RBLOCK-1]);
                RBLOCK := (RBLOCK+FBLKSIZE-1) DIV FBLKSIZE;
                FBLOCKIO := RBLOCK;
                FEOF := RBLOCK < NBLOCKS
              END
            ELSE
          ELSE FBLOCKIO := 0
        END
    ELSE
      SYSCOM^.IORSLT := INOTOPEN
END (*FBLOCKIO*) ;

PROCEDURE FGET(*VAR F: FIB*);
  LABEL 1, 2;
  VAR LEFTOGET,WININX,LEFTINBUF,AMOUNT: INTEGER;
      DONE: BOOLEAN;
BEGIN SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN THEN
      BEGIN
        IF FREPTCNT > 0 THEN
          BEGIN FREPTCNT := FREPTCNT-1; IF FREPTCNT > 0 THEN GOTO 2 END;
```

```
IF FSOFTBUF THEN
  WITH FHEADER DO
    BEGIN
      LEFTOGET := FRECSIZE; WININX := 0;
      REPEAT
        IF FNXTBLK = FMAXBLK THEN
          IF FNXTBYTE+LEFTOGET > FMAXBYTE THEN GOTO 1
          ELSE LEFTINBUF := DLASTBYTE-FNXTBYTE
        ELSE LEFTINBUF := FBLKSIZE-FNXTBYTE;
        AMOUNT := LEFTOGET;
        IF AMOUNT > LEFTINBUF THEN AMOUNT := LEFTINBUF;
        IF AMOUNT > 0 THEN
          BEGIN
            MOVELEFT(FBUFFER[FNXTBYTE],FWINDOW^[WININX],AMOUNT);
            FNXTBYTE := FNXTBYTE+AMOUNT;
            WININX := WININX+AMOUNT;
            LEFTOGET := LEFTOGET-AMOUNT
          END;
        DONE := LEFTOGET = 0;
        IF NOT DONE THEN
          BEGIN
            IF FBUFCHNGD THEN
              BEGIN FBUFCHNGD := FALSE; FMODIFIED := TRUE;
                UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+FNXTBLK-1)
              END;
            IF IORESULT <> ORD(INOERROR) THEN GOTO 1;
            UNITREAD(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+FNXTBLK);
            IF IORESULT <> ORD(INOERROR) THEN GOTO 1;
            FNXTBLK := FNXTBLK+1; FNXTBYTE := 0
          END
      UNTIL DONE
    END
ELSE
  BEGIN
    UNITREAD(FUNIT,FWINDOW^,FRECSIZE);
    IF IORESULT <> ORD(INOERROR) THEN GOTO 1
  END;
IF FRECSIZE = 1 THEN (*FILE OF CHAR*)
  BEGIN FEOLN := FALSE;
    IF FSTATE <> FJANDW THEN FSTATE := FGOTCHAR;
    IF FWINDOW^[0] = CHR(EOL) THEN
      BEGIN FWINDOW^[0] := ' '; FEOLN := TRUE; GOTO 2 END;
    IF FWINDOW^[0] = CHR(DLE) THEN
      BEGIN FGET(F);
        AMOUNT := ORD(FWINDOW^[0])-32;
        IF (AMOUNT > 0) AND (AMOUNT <= 127) THEN
          BEGIN
            FWINDOW^[0] := ' ';
            FREPTCNT := AMOUNT;
            GOTO 2
          END;
        FGET(F)
      END;
    IF FWINDOW^[0] = CHR(0) THEN
      BEGIN (*EOF HANDLING*)
        IF FSOFTBUF AND (FHEADER.DFKIND = TEXTFILE) THEN
          BEGIN (*END OF 2 BLOCK PAGE*)
            IF ODD(FNXTBLK) THEN FNXTBLK := FNXTBLK+1;
            FNXTBYTE := FBLKSIZE; FGET(F)
          END
```

```
                ELSE
                  BEGIN FWINDOW^[0] := ' '; GOTO 1 END
              END
          END
      END
    ELSE
      BEGIN
        SYSCOM^.IORSLT := INOTOPEN;
1:      FEOF := TRUE; FEOLN := TRUE
      END;
2:
END (*FGET*) ;

PROCEDURE FPUT(*VAR F: FIB*);
  LABEL 1;
  VAR LEFTOPUT,WININX,LEFTINBUF,AMOUNT: INTEGER;
      DONE: BOOLEAN;
BEGIN SYSCOM^.IORSLT := INOERROR;
  WITH F DO
    IF FISOPEN THEN
      BEGIN
        IF FSOFTBUF THEN
          WITH FHEADER DO
            BEGIN
              LEFTOPUT := FRECSIZE; WININX := 0;
              REPEAT
                IF DFIRSTBLK+FNXTBLK = DLASTBLK THEN
                  IF FNXTBYTE+LEFTOPUT > DLASTBYTE THEN
                    IF CANTSTRETCH( F ) THEN
                        BEGIN SYSCOM^.IORSLT := INOROOM; GOTO 1 END
                    ELSE LEFTINBUF := FBLKSIZE-FNXTBYTE
                  ELSE LEFTINBUF := DLASTBYTE-FNXTBYTE
                ELSE LEFTINBUF := FBLKSIZE-FNXTBYTE;
                AMOUNT := LEFTOPUT;
                IF AMOUNT > LEFTINBUF THEN AMOUNT := LEFTINBUF;
                IF AMOUNT > 0 THEN
                  BEGIN FBUFCHNGD := TRUE;
                    MOVELEFT(FWINDOW^[WININX],FBUFFER[FNXTBYTE],AMOUNT);
                    FNXTBYTE := FNXTBYTE+AMOUNT;
                    WININX := WININX+AMOUNT;
                    LEFTOPUT := LEFTOPUT-AMOUNT
                  END;
                DONE := LEFTOPUT = 0;
                IF NOT DONE THEN
                  BEGIN
                    IF FBUFCHNGD THEN
                      BEGIN FBUFCHNGD := FALSE; FMODIFIED := TRUE;
                        UNITWRITE(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+FNXTBLK-1)
                      END;
                    IF IORESULT <> ORD(INOERROR) THEN GOTO 1;
                    IF FNXTBLK < FMAXBLK THEN
                      UNITREAD(FUNIT,FBUFFER,FBLKSIZE,DFIRSTBLK+FNXTBLK)
                    ELSE
                      FILLCHAR(FBUFFER,FBLKSIZE,CHR(0));
                    IF IORESULT <> ORD(INOERROR) THEN GOTO 1;
                    FNXTBLK := FNXTBLK+1; FNXTBYTE := 0
                  END
              UNTIL DONE;
              IF FRECSIZE = 1 THEN
                IF FWINDOW^[0] = CHR(EOL) THEN
```

```
                IF DFKIND = TEXTFILE THEN
                  IF (FNXTBYTE >= FBLKSIZE-127) AND NOT ODD(FNXTBLK) THEN
                    BEGIN
                      FNXTBYTE := FBLKSIZE-1;
                      FWINDOW^[0] := CHR(0);
                      FPUT(F)
                    END
            END
        ELSE
          BEGIN
            UNITWRITE(FUNIT,FWINDOW^,FRECSIZE);
            IF IORESULT <> ORD(INOERROR) THEN GOTO 1
          END
      END
    ELSE
      BEGIN
        SYSCOM^.IORSLT := INOTOPEN;
1:      FEOF := TRUE; FEOLN := TRUE
      END
END (*FPUT*) ;

FUNCTION FEOF(*VAR F: FIB*);
BEGIN FEOF := F.FEOF END;

(* TEXT FILE INTRINSICS *)

FUNCTION FEOLN(*VAR F: FIB*);
BEGIN FEOLN := F.FEOLN END;

PROCEDURE FWRITELN(*VAR F: FIB*);
BEGIN
  F.FWINDOW^[0] := CHR(EOL); FPUT(F)
END (*FWRITELN*) ;

PROCEDURE FWRITECHAR(*VAR F: FIB; CH: CHAR; RLENG: INTEGER*);
  LABEL  1;
BEGIN
  WITH F DO
    IF FISOPEN THEN
      IF FSOFTBUF THEN
        BEGIN
          WHILE RLENG > 1 DO
            BEGIN FWINDOW^[0] := ' '; FPUT(F);
              RLENG := RLENG-1
            END;
          FWINDOW^[0] := CH; FPUT(F)
        END
      ELSE
        BEGIN
          WHILE RLENG > 1 DO
            BEGIN FWINDOW^[0] := ' ';
              UNITWRITE(FUNIT,FWINDOW^,1);
              RLENG := RLENG-1
            END;
          FWINDOW^[0] := CH;
          UNITWRITE(FUNIT,FWINDOW^,1)
        END
    ELSE SYSCOM^.IORSLT := INOTOPEN;
1:
END (*FWRITECHAR*) ;
```

```
PROCEDURE FWRITEINT(*VAR F: FIB; I,RLENG: INTEGER*);
  LABEL 1;
  VAR POT,COL: INTEGER; CH: CHAR;
      SUPPRESSING: BOOLEAN; S: STRING[10];
BEGIN COL := 1;
  S[0] := CHR(10); SUPPRESSING := TRUE;
  IF I < 0 THEN
    BEGIN I := ABS(I); S[1] := '-'; COL := 2;
      IF I = 0 THEN (*HARDWARE SPECIAL CASE*)
        BEGIN S := '-32768'; GOTO 1 END
    END;
  FOR POT := 4 DOWNTO 0 DO
    BEGIN CH := CHR(I DIV IPOT[POT] + ORD('0'));
      IF (CH = '0') AND (POT > 0) AND SUPPRESSING THEN
      ELSE (*FORMAT THE CHAR*)
        BEGIN SUPPRESSING := FALSE;
          S[COL] := CH; COL := COL+1;
          IF CH <> '0' THEN I := I MOD IPOT[POT]
        END
    END;
  S[0] := CHR(COL-1);
1:IF RLENG < LENGTH(S) THEN
    RLENG := LENGTH(S);
  FWRITESTRING(F,S,RLENG)
END (*FWRITEINT*) ;

PROCEDURE FWRITESTRING(*VAR F: FIB; VAR S: STRING; RLENG: INTEGER*);
  VAR SINX: INTEGER;
BEGIN
  WITH F DO
    IF FISOPEN THEN
      BEGIN
        IF RLENG <= 0 THEN RLENG := LENGTH(S);
        IF RLENG > LENGTH(S) THEN
          BEGIN FWRITECHAR(F,' ',RLENG-LENGTH(S)); RLENG := LENGTH(S) END;
        IF FSOFTBUF THEN
          BEGIN SINX := 1;
            WHILE (SINX <= RLENG) AND NOT FEOF DO
              BEGIN FWINDOW^[0] := S[SINX]; FPUT(F); SINX := SINX+1 END
          END
        ELSE
          UNITWRITE(FUNIT,S[1],RLENG)
      END
    ELSE SYSCOM^.IORSLT := INOTOPEN
END (*FWRITESTRING*) ;

PROCEDURE FREADSTRING(*VAR F: FIB; VAR S: STRING; SLENG: INTEGER*);
  VAR SINX: INTEGER; CH: CHAR;
BEGIN
  WITH F DO
      BEGIN SINX := 1;
        IF FSTATE = FNEEDCHAR THEN FGET(F);
        S[0] := CHR(SLENG); (*NO INV INDEX*)
        WHILE (SINX <= SLENG) AND NOT (FEOLN OR FEOF) DO
          BEGIN CH := FWINDOW^[0];
            IF FUNIT = 1 THEN
              IF CHECKDEL(CH,SINX) THEN
              ELSE
                BEGIN S[SINX] := CH; SINX := SINX + 1 END
```

```
            ELSE
              BEGIN S[SINX] := CH; SINX := SINX + 1 END;
            FGET(F)
          END;
        S[0] := CHR(SINX - 1);
        WHILE NOT FEOLN DO FGET(F)
      END
END (*FREADSTRING*) ;

PROCEDURE FWRITEBYTES(*VAR F: FIB; VAR A: WINDOW; RLENG,ALENG: INTEGER*);
  VAR AINX: INTEGER;
BEGIN
  WITH F DO
    IF FISOPEN THEN
      BEGIN
        IF RLENG > ALENG THEN
          BEGIN FWRITECHAR(F,' ',RLENG-ALENG); RLENG := ALENG END;
        IF FSOFTBUF THEN
          BEGIN AINX := 0;
            WHILE (AINX < RLENG) AND NOT FEOF DO
              BEGIN FWINDOW^[0] := A[AINX]; FPUT(F); AINX := AINX+1 END
          END
        ELSE
          UNITWRITE(FUNIT,A,RLENG)
      END
    ELSE SYSCOM^.IORSLT := INOTOPEN
END (*FWRITEBYTES*) ;

PROCEDURE FREADLN(*VAR F: FIB*);
BEGIN
  WHILE NOT F.FEOLN DO FGET(F);
  IF F.FSTATE = FJANDW THEN FGET(F)
  ELSE
    BEGIN F.FSTATE := FNEEDCHAR; F.FEOLN := FALSE END
END (*FREADLN*) ;

PROCEDURE FREADCHAR(*VAR F: FIB; VAR CH: CHAR*);
BEGIN
  WITH F DO
      BEGIN SYSCOM^.IORSLT := INOERROR;
        IF FSTATE = FNEEDCHAR THEN FGET(F);
        CH := FWINDOW^[0];
        IF FSTATE = FJANDW THEN FGET(F)
        ELSE FSTATE := FNEEDCHAR
      END
END (*FREADCHAR*) ;

PROCEDURE FREADINT(*VAR F: FIB; VAR I: INTEGER*);
  LABEL 1;
  VAR CH: CHAR; NEG,IVALID: BOOLEAN; SINX: INTEGER;
BEGIN
  WITH F DO
      BEGIN I := 0; NEG := FALSE; IVALID := FALSE;
        IF FSTATE = FNEEDCHAR THEN FGET(F);
        WHILE (FWINDOW^[0] = ' ') AND NOT FEOF DO FGET(F);
        IF FEOF THEN GOTO 1;
        CH := FWINDOW^[0];
        IF (CH = '+') OR (CH = '-') THEN
          BEGIN NEG := CH = '-'; FGET(F); CH := FWINDOW^[0] END;
        IF CH IN DIGITS THEN
```

```
          BEGIN  IVALID := TRUE; SINX := 1;
            REPEAT
              I := I*10+ORD(CH)-ORD('0');
              FGET(F); CH := FWINDOW^[0]; SINX := SINX+1;
              IF FUNIT = 1 THEN
                WHILE CHECKDEL(CH,SINX) DO
                  BEGIN
                    IF SINX = 1 THEN I := 0 ELSE I := I DIV 10;
                    FGET(F); CH := FWINDOW^[0]
                  END
            UNTIL NOT (CH IN DIGITS) OR FEOLN
          END;
        IF IVALID OR FEOF THEN
          IF NEG THEN I := -I ELSE (*NADA*)
        ELSE SYSCOM^.IORSLT := IBADFORMAT
      END;
1:
END (*FREADINT*) ;

(* STRING VARIABLE INTRINSICS *)

PROCEDURE SCONCAT(*VAR SRC,DEST: STRING; DESTLENG: INTEGER*);
BEGIN
  IF LENGTH(SRC)+LENGTH(DEST) <= DESTLENG THEN
    BEGIN
      MOVELEFT(SRC[1],DEST[LENGTH(DEST)+1],LENGTH(SRC));
      DEST[0] := CHR(LENGTH(SRC)+LENGTH(DEST))
    END
END (*SCONCAT*) ;

PROCEDURE SINSERT(*VAR SRC,DEST: STRING; DESTLENG,INSINX: INTEGER*);
  VAR ONRIGHT: INTEGER;
BEGIN
  IF (INSINX > 0) AND (LENGTH(SRC) > 0) AND
      (LENGTH(SRC)+LENGTH(DEST) <= DESTLENG) THEN
    BEGIN
      ONRIGHT := LENGTH(DEST)-INSINX+1;
      IF ONRIGHT > 0 THEN
        BEGIN
          MOVERIGHT(DEST[INSINX],DEST[INSINX+LENGTH(SRC)],ONRIGHT);
          ONRIGHT := 0
        END;
      IF ONRIGHT = 0 THEN
        BEGIN
          MOVELEFT(SRC[1],DEST[INSINX],LENGTH(SRC));
          DEST[0] := CHR(LENGTH(DEST)+LENGTH(SRC))
        END
    END
END (*SINSERT*) ;

PROCEDURE SCOPY(*VAR SRC,DEST: STRING; SRCINX,COPYLENG: INTEGER*);
BEGIN DEST := '';
  IF (SRCINX > 0) AND (COPYLENG > 0) AND
      (SRCINX+COPYLENG-1 <= LENGTH(SRC)) THEN
    BEGIN
      MOVELEFT(SRC[SRCINX],DEST[1],COPYLENG);
      DEST[0] := CHR(COPYLENG)
    END
END (*SCOPY*) ;
```

```
PROCEDURE SDELETE(*VAR DEST: STRING; DELINX,DELLENG: INTEGER*);
  VAR ONRIGHT: INTEGER;
BEGIN
  IF (DELINX > 0) AND (DELLENG > 0) THEN
    BEGIN
      ONRIGHT := LENGTH(DEST)-DELINX-DELLENG+1;
      IF ONRIGHT = 0 THEN DEST[0] := CHR(DELINX-1)
      ELSE
        IF ONRIGHT > 0 THEN
          BEGIN
            MOVELEFT(DEST[DELINX+DELLENG],DEST[DELINX],ONRIGHT);
            DEST[0] := CHR(LENGTH(DEST)-DELLENG)
          END
    END
END (*SDELETE*) ;

FUNCTION SPOS(*VAR TARGET, SRC: STRING*);
LABEL 1;
VAR  TEMPLOC,DIST: INTEGER;
     FIRSTCH: CHAR;
     TEMP: STRING;
BEGIN SPOS := 0;
  IF LENGTH(TARGET) > 0 THEN
    BEGIN
      FIRSTCH := TARGET[1];
      TEMPLOC := 1;
      DIST := LENGTH(SRC)-LENGTH(TARGET) + 1;
      TEMP[0] :=  TARGET[0];
      WHILE TEMPLOC <= DIST DO
        BEGIN
          TEMPLOC := TEMPLOC + SCAN(DIST-TEMPLOC,=FIRSTCH,SRC[TEMPLOC]) ;

          IF TEMPLOC>DIST THEN
             GOTO 1;
          MOVELEFT(SRC[TEMPLOC],TEMP[1],LENGTH(TARGET));
          IF TEMP=TARGET THEN
            BEGIN SPOS := TEMPLOC; GOTO 1 END;
          TEMPLOC := TEMPLOC+1
        END
    END;
1:
END (*SPOS*) ;

(* MAIN DRIVER OF SYSTEM *)

PROCEDURE COMMAND;
  VAR T: INTEGER;
BEGIN STATE := HALTINIT;
  REPEAT
    RELEASE(EMPTYHEAP);
    WHILE UNITABLE[SYSCOM^.SYSUNIT].UVID <> SYVID DO
      BEGIN
        PL := 'Put in :';
        INSERT(SYVID,PL,8);
        PROMPT; T := 4000;
        REPEAT T := T-1
        UNTIL T = 0;
        IF FETCHDIR(SYSCOM^.SYSUNIT) THEN
      END;
    STATE := GETCMD(STATE);
```

```
    CASE STATE OF
      UPROGNOU,UPROGUOK,SYSPROG,
      COMPONLY,COMPANDGO,COMPDEBUG,
      LINKANDGO,LINKDEBUG:
        USERPROGRAM(NIL,NIL);
      DEBUGCALL:
        DEBUGGER
    END;
    IF STATE IN [UPROGNOU,UPROGUOK] THEN
      BEGIN
        FCLOSE(GFILES[0]^,CNORMAL);
        FCLOSE(GFILES[1]^,CLOCK)
      END;
    IF UNITBUSY(1) OR UNITBUSY(2) THEN
      UNITCLEAR(1)
  UNTIL STATE = HALTINIT
END (*COMMAND*) ;

BEGIN (*UCSD PASCAL SYSTEM*)
  EMPTYHEAP := NIL;
  INITIALIZE;
  REPEAT
    COMMAND;
    IF EMPTYHEAP <> NIL THEN
      INITIALIZE
  UNTIL EMPTYHEAP = NIL
END (*PASCALSYSTEM*) .

{ +-------------------------------------------------------------------+
  |                                                                   |
  |                     F     I     N     I     S                     |
  |                                                                   |
  +-------------------------------------------------------------------+ }

### END OF FILE UCSD Pascal 1.5 System
```