[illegible]

(MLK)MODS   (MLK)MODS   (MLK)MODS   (MLK)MODS   (MLK)MODS   (MLK)MODS

```
< NLS, MODS.NLS;62, >, 11-OCT-74 21:58 KIRK ;;;;
The following is a list of the bugs to fix, and the modifications to be
made  before we bring the new system up at OFFICE-1.The Tasks are
ordered in the approximate order in which they will be implemented by
the developement staff. The developent staff will implement as many of
these as is possible before the OFFICE-1 NLS s FROZEN ON Sept 1st. An
accurate estimate of exactly how far down the list we will be by Sept
1st will be published by the development staff by August 15. The
priority of each task (first digit of statement name) is a product of
the relative importance of the task (middle character of statement
name, a=most important, c=least important) and the difficulty of the
task (last digit of statement name, 1= least difficult, 3=most
difficult). The tasks listed below include all of the tasks in category
one and some of the tasks from category two in <GJOURNAL,23653,>. The
numbers immediately following the statement name are references to the
SID of the associated statement in the aforementioned Journal document.
                                                                     09
(BUGS)  <see -- bgs,>
in approximate order of expected execution ...                     0369
(MODS)                                                              010
   EKM 1-MAY-74  MISSING COMMANDS                                  0745
      The following commands fell through a crack:                 0746
         Show Selections                                           0747
         Show Control Mark                                         0748
         Show Statement #'s in leveladjust                         0749
         Show Upper Case                                           0750
         Set Tabs                                                  0751
         Show Tabs                                                 0752
      They should be implemented both in the SET and SHOW Editor
      commands and in the User Options subsystem.                  0753
   (3a3) -027- Valid alternatives should be available in Help exactly
   as stated in the response to questionmark, especially such things as
   <tab>, <insert>, etc.,  as well as ANY other response.
   (feedback,fdbk,03191)                                            014
      (We acknowledge that this may require changes to the Help/Query
      command recognition algorithm.)  Not to be done now           015
    ------- Mods above this line are the only ones development has
   committed itself to implement before October 1. -----           0910
   KEV 9-OCT-74 09:22  24171
   things i didn't get doen for nls-8
   Location: (JJOURNAL, 24171, 1:W)
   *****Note:  [ INFO-ONLY ] *****
                                                                  01014
      the following is a list of things that were (are) on my todo list
      for nls-8.  due to time pressures and relative priorities they
      did not get done.  i am distributing this in the hopes that maybe
      others might have some time in the future to implement them. 01015
         provide a meta-language to implement an interactive process
         commands ability                                         01016
         implement what i call mouse button macro strings:        01017
            this associates a string (kept in user-options) for each of
            the keyset chords associated with mouse case-shift 101.
            thus when a user hits a chord on the keyset while mouse
            buttons 101 are down, it would be equivalent to entering
            the string associated with that chord.                01018
         fix the bug associated with improper prompting if both
      level-adjust and optional prompting are turned off.       01019
         this bug should go away when we rewrite the interpretter.
                                                                01020
      implement the following commands:                         01021
         subsystem XXX: Detach                                  01022
         subsystem BASE:                                        01023
            Undelete Modifications                              01024
            ↑ and linefeed in DNLS                              01025
            Copy, Show, and Expunge Archive Directory            01026
            Retreive, Delete, and Undelete Archive File          01027
            Show Who is on                                      01028
            Show Where is user                                  01029
(3cl) -0103- Let words containing hyphens break at the hyphen when
it is at the end of a line. (feedback,fdbk,02775)                 021
   (and commas, and periods?)                                    0928
   From time to time I have had such a long link  e. g.
   (xjournal,12345,lab)  that the disply won't make a reasonable
   line because it breaks only at invisisbles. If the display looked
   for ', as well it would  save us an occasional really ugly
   screen.                                                       0926
      You might or might not want long numbers with commas to be
      broken. If you did not want them broken it would be easey to
      check if a comma were in a number. (21714,) (DVN)          0927
(3cl) -039- /dsm/ Setting name delimiters should change the
Statement Signature. (feedback,fdbk,02717)                       022
   Setting name delimiters does not change the Statement Signature.
   (22059,) (KIRK)                                               0905
(3cl) -041- Since a person is told a file is bad, he should also be
told it is good.  "file verify in progress" is an o.k. message, akin
to "output quickprint in progress". (feedback,fdbk,02764)         023
(3cl) -047- <tab> is listed as an alternative in response to
questionmark, but if not typed in the proper context it's responded
to by "Illegal Search Type". (feedback,fdbk,02651) This err msg
should be changed to  "<tab> valid only to repeat a previous search"
                                                                 024
   Is it reasonable to get an error message when an alternative is
   typed, i.e. Illegal search type is displayed when a user types
   <tab>. (MEJ)                                                  0899
(3cl) -050- The validity of the characters used for name delimiters
should be checked as they are typed in. (feedback,fdbk,02718)     025
    Check validity of name deliminator when being set in Set command
   (22802,) (RLL)                                                0906
(3cl) -056- The herald should be settable as an option to zero
length leaving just a prompt. (feedback,fdbk,02907)               026
   RLL 15-MAY-74 19:20  23011
   herald length zero. why not?
   Message: Why can't tthe herald be set to length zero? The prompt
   (if at least partially on) will be sufficient to key the user.
   The terse option could be defaoult to be length zero if any
   prompt is on (partial or full). otherwise leave terse as is when
   no prompt is on.
   *****Note:  / ACTION / *****
                                                                 0879
(3cl) -058- Put the "Process Commands Branch" command in the Editor
subsystem.                                                        027
(3cl) -092- Allow viewspec "o" and "p" to be set before the
```

completion of freeze and release commands and let "release all" OPTIONALLY result in viewspec "p". (feedback,fdbk,03249) 028

(4b2) Make a new viewspec which would turn all indenting off putting all text left justified regardless of structure. 029

(6b3) -069- CONFIRM should work for recognition so that <sp> or <esc> is not required when an entire command has been typed. (feedback,fdbk,02709) i.e. have CA as a right delimiter but not swallowed by the #cml

(c2) -0111- Since ;filter; is defined as a viewspec in a link it should be a valid viewspec whenever the prompt V: appears. (feedback,fdbk,02741) 031

(6c2) -059- Resolve the present bug in newnls that makes it impossible to "jump to name" in the identfile when the name is enclosed in single quotes. i.e. implement by removing the first single quote ' from the last names in the ident file. 032

(6c2) Review TNLS CALCULATOR and DNLS CALCULATOR 033

(6c2) in the output to terminal command add an output to file option which outputs to a sequential file from the output processor. 034

(6c3) -023- Rather than having left-over prompts at the top of the screen, such as "Replace Text at through by through", display the actual text typed, following and on the same line as the respective prompts. Need an appropriate symbol for a bug mark (possibly the word or character if text; the statement number, if structure; or a symbol such as "<bug>"). This would more closely approximate TNLS. (feedback,fdbk,01927), (feedback,fdbk,03236) Development feels that the current implementation of noisewords () is OK, we may display "<bug>" for bug selections. 035

  in inset and substitue commands where multiple words or characters can be input the text (s) should appear after word or character (Ithought we had decided to do that may months ago, why didn't we?) (22940,) (RWW) 0902

(9c3) -065- After typing a space and one character, a backspace character should result in your being able to type another second-level command. (feedback,fdbk,03151) 036

(9c 3) -070- Have a new user option to set the escape charater (and its echo) to be other than "<sp>". 037

  Setting expert second level recognition key from space to period. (22061,) (KIRK) 0904

For the default in TNLS, don't repeat prompts, E.G. Update C: File OK:/C C: Compact OK:. Both C:'s aren't needed (22964,) (RLL) 0898

(DONE) to be documented

this also contains the "DO" list for documentation. 0453

  new Syntax command 0946

  Syntax user-subsystem. (currently not in directory programs) 0947

  1 b. The question mark facility has several inadequacies: 0922

    d. the valid alternatives are not always described in the HELP system data base. 0923

      The valid alternatives should be available in the HELP data base EXACTLY as shown in the question-mark response. 0924

      When they're not there, or listed differently, this leads of course to utter frustration for a new NEW NLS user: he can't easily find out what to do in a given situation. I personally find this to be one of the most significant reasons that I don't use NEW NLS very much. (23116,) (MDK) 0925

  RLL help subsystem suggestion: list the subsystem which the command is in.

```
  _aug_     :32 (GJOURNAL, 23766, 1:w)
*****Note:  [ ACTION ] *****
                                                                      0654
     In the help subsystem: suggestion you include he relevant
     subsystem name whenever referring to a command.  One could get to
     a command (such as show status) and not know what the subsystem
     is.  In the case of the editor commands I think they too should
     be clearly marked as beng in the editor(this is a big job and
     would be satisfied if no subsystem was mentioned it was assumed t
     be an edito command.)                                            0655
  RWW                                                                 01031
     Address discussion: Stuff should be written for novces which menu
     items for more detail.                                           01032
     Provide Classes of emergencies                                   01033
     Add synonyms for classes like viewing, jumping, modifying,
     editing, etc.                                                    01034
     Add plurals linking to appropriate places.                       01035
  Add explanations to OLDFILELINK and NEWFILELINK                     01037
  Write the PROTECTION concept (fields in filename)                   01038
  do a sample content analyzer                                        01036
  check to make sure there is a name for every word that's not already
  in the Database and that appears as a result of "?" in nls.         0791
  Document the new Run, and Kill Tenex Subsystem, and Show Tenex
  Subsystem Status.                                                   0792
  Interface to OP users guide                                         01039
   Interface to document locatos                                      01040
  Inerface to Content Anayzer guide                                   01041
  Inerface to L10 guide.                                              01042
  Interface to TENEX user's guide                                     01043
  Write Identification Subsystem                                      01044
  Rewrite the example for the new Protect command in Editor,  see the
  revised syntax statement (see--0248).                               0793
  check to make sure all commands from the old system that are not in
  the new system link from the old command to their new counter parts
  in help.                                                            0977
  Write EXAMPLES for Editor commands:  Insert Date; Insert TIme and
  date                                                                0803
  add angle-bracket in DAE as concept.                                0807
  anglebracket (left and right)                                       0808
  greater-than and greaterthan                                        0809
  less-than, lessthan                                                 0810
(DOCUMENTED) <see -- feedback,fdbk, documented>                       0368
```

[illegible]

```
< NLS, NDDT.NLS;56, >, 14-AUG-74 11:08 KJM ;;;;< NLS, NDDT.NLS;55, >,
26-JUN-74 11:09 KEV ;
FILE nddt  % L10 <REL-NLS>nddt.rel;99 %% (L10,) (rel-nls,nddt.rel;99,) %
                                                                        02
UND-OK                                                                  03
% pattern for getting table of contents - $NP '( $LD ') ;  %            04
%* TABLE OF CONTENTS OF CONTAINED ROUTINES %                            05
%Defines and declaractions%                                             06
   %Defines...alphabetically listed (not compiled)                      07
      DEFINE addop = 1#;                                                08
      DEFINE addressmode = rd.adrmod#;                                  09
      DEFINE areg1 =db[12]#;                                           010
      DEFINE areg2 =db[13]#;                                           011
      DEFINE areg3 =db[14]#;                                           012
      DEFINE areg4 =db[15]#;                                           013
      DEFINE backupchar = bkjfnjsys()#;                                014
      DEFINE bptblentsz = 4#;                                          015
      DEFINE bptblsz =40#;                                             016
      DEFINE calldefineproc = &calprc←defineproc; calprc#;             017
      DEFINE callmother=&calprc←mother; calprc#;                       018
      DEFINE callreplaceproc = &calprc←rd.replact;calprc#;             019
      DEFINE calltyperoutine = &calprc←typeroutine; calprc#;           020
      DEFINE came =312B9#;                                             021
      DEFINE camg =317B9#;                                             022
      DEFINE camge=315B9#;                                             023
      DEFINE caml =311B9#;                                             024
      DEFINE camle=313B9#;                                             025
      DEFINE camn =316B9#;                                             026
      DEFINE continsig = 2#;                                           027
      DEFINE currentfld=&rd+rdsize+(IF fieldnumber >= numberfields THEN
      err($"Field Out of bounds") ELSE fieldnumber)*fielddescsz#;      028
      DEFINE currenttypemode =db[21]#;                                 029
      DEFINE daughter = rd.kidrec#;                                    030
      DEFINE ddtlimit = 500B#;                                         031
      DEFINE ddtsymbol = symtype.ddtsm#;                               032
      DEFINE ddttablesz = 38#;                                         033
      DEFINE defaultmode = 0#;                                         034
      DEFINE defaultreplace = 1#;                                      035
      DEFINE defineproc=rd.rdef#;                                      036
      DEFINE divop = 4#;                                               037
      DEFINE dynamicrecord = rd.rdproc#;                               038
      DEFINE enterddt =!JSP 0,770002B#;                                039
      DEFINE entity = rd.enttyp#;                                      040
      DEFINE escchar = db[17]#;                                        041
      DEFINE fielddescsz=2#;                                           042
      DEFINE fieldentity = entity=fieldtype#;                          043
      DEFINE fieldnumber = rd.rindex#;                                 044
      DEFINE fieldptr=fld.fpointr#;                                    045
      DEFINE fieldstart = &rd+rdsize#;                                 046
      DEFINE fieldsymbol = symtype.fldsig#;                            047
      DEFINE fieldtype=2#;                                             048
      DEFINE fixoverflow =fixoutofbounds(&rd, overflowaction,
      overflow)#;                                                      049
      DEFINE fixunderflow =fixoutofbounds(&rd, underflowaction,
      underflow)#;                                                     050
      DEFINE framebase=db[24]#;                                        051
DEFINE framebasetype =2#;
DEFINE framemark =[framep].marklocfield#;
DEFINE framep = db[19]#;
DEFINE framesize=db[23]#;
DEFINE frametop =[stacktop].marklocfield#;
DEFINE frametoptype =1#;
DEFINE global = 1#;
DEFINE gosig = 1#;
DEFINE indirectmode=1B6#;
DEFINE indirectsym = indddtsym#;
DEFINE initialised =rd.rdinit#;
DEFINE internalfield = internalsym + 1B9#;
DEFINE internalstacksym = internalsym+1B6#;
DEFINE internalsym = 2B6#;
DEFINE jrst =254B9#;
DEFINE jsp = 265B9#;
DEFINE lbptbl=db[26]#;
DEFINE lbptblentsz = 1#;
DEFINE lbptblsz=nbkpts#;
DEFINE lc =db[36]#;
DEFINE leftbyteptrrj = 4307B8#;
DEFINE lv = db[16]#;
DEFINE macro = 0#;
DEFINE macromode =  ADDRESSMODE=macro#;
DEFINE maxnumfields = 50#;
DEFINE maxprms = 20#;
DEFINE maxwordsinstring = 402??;
DEFINE micro = 1#;
DEFINE micromode =addressmode=micro#;
DEFINE mother = rd.momrec#;
DEFINE movei =201B#;
DEFINE mulop = 3#;
DEFINE nbkpts=10#;
DEFINE nextchar = binjsys()#;
DEFINE noindexmask = 777760777777B#;
DEFINE nparms=db[22]#;
DEFINE number = symtype.numsym#;
DEFINE numberfields = rd.nfield#;
DEFINE numericmode = 1#;
DEFINE opcall0=1#;
DEFINE opcall1=2#;
DEFINE opcallm=3#;
DEFINE opcodmask =9M9#;
DEFINE optabl = 24B#;
DEFINE overflow=1#;
DEFINE overflowaction = rd.topact#;
DEFINE proceedaction = 1#;
DEFINE proctblentsz =2#;
DEFINE proctblsz = 10#;
DEFINE progm  db[9]#;
DEFINE progp =db[7]#;
DEFINE programsymbol = symtype.progsym#;
DEFINE progrp =db[10]#;
DEFINE progs =db[8]#;
DEFINE quitsig = 3#;
DEFINE radix=8#;
DEFINE rdsize = 6#; %%Number of PDP10 words in record%%
DEFINE recordentity = entity=recordtype#;
DEFINE recordname = rd.rdef#;
DEFINE recordsize = rd.rsize#;
DEFINE recordstart = rd.raddr#;
DEFINE recordtype=3#;
DEFINE recp =db[37]#;
DEFINE recprecordaddr =22B8#;
DEFINE recprecsize =2222B8#;
DEFINE reg1 =db#;
DEFINE reg2=db[1]#;
DEFINE reg3 =db[2]#;
DEFINE reg4 =db[3]#;
DEFINE reg5 =db[4]#;
DEFINE reg6 =db[5]#;
DEFINE reg7 =db[6]#;
DEFINE replaceaction = rd.replact#;
DEFINE returnloc =[framep].retlocfield#;
DEFINE rfnmfg = db[20]#;
DEFINE ribsize = 106#; %%rdsize + maxnumfields*fielddescsz%%
DEFINE rpreg =db[11]#;
DEFINE seqsymbol = 4B6#;
DEFINE sequencename = symtype.seqsym#;
DEFINE setfieldptr =&fld ← currentfld#;
DEFINE specialsymbol = stacksymbol .V sequencename#;
DEFINE stackref=sequence#;
DEFINE stacksymbol = symtype.stksym#;
DEFINE stacksymv =1B6#;
DEFINE stacktop=db[25]#;
DEFINE strbyteptr =440700000001B#;
DEFINE stringentity = entity=stringtype#;
DEFINE stringlength =22B8#;
DEFINE stringmax = 2222B8#;
DEFINE stringtype=4#;
DEFINE subop = 2#;
DEFINE suppressloc = rd.suploc#;
DEFINE symbolicmode = 2#;
DEFINE symbolproc = symtype.spaddr#;
DEFINE symflg = db[18]#;
DEFINE textmode = 3#;
DEFINE typedefault=typemode=defaultmode#;
DEFINE typemode = fld.ftypmod#;
DEFINE typenumeric = typemode=1#;
DEFINE typeroutine=typemode#;
DEFINE typespecial =typemode>1000#;
DEFINE typesymbolic = typemode=2#;
DEFINE typetext = typemode=3#;
DEFINE undefaction = 0#;
DEFINE underflow=2#;
DEFINE underflowaction = rd.botact#;
DEFINE wordbyteptr =44B8#;
DEFINE wordentity = entity=wordtype#;
DEFINE wordtype = 1#;
DEFINE xpointer = fieldptr.xptr=77B#;
DEFINE xptraddr = fieldptr.xptadd/xptrbytesperword#;
DEFINE xptrbitpos =36-((fieldptr.xptadd MOD
      xptrbytesperword)*fieldptr.xbytsz)#;
      DEFINE xptrbytesperword=(36/fieldptr.xbytsz)#;
      DEFINE xptrbytesz = fieldptr.xbytsz#;
      DEFINE xptrsize = fieldptr.xnbyte#;
   %
   %Defines%
      %Miscellaneous%
         %Miscellaneous masks, I/O, values, instructions%
            DEFINE radix=8#;
            DEFINE global = 1#;
            DEFINE gosig = 1#, continsig = 2#, quitsig = 3#;
            DEFINE noindexmask = 777760777777B#;
            DEFINE enterddt =!JSP 0,770002B??;
            DEFINE nextchar = binjsys()#;
            DEFINE backupchar = bkjfnjsys()#;
            DEFINE addop = 1#, subop = 2#, mulop = 3#, divop = 4#;
         %Hash table defines%
            DEFINE hashbase=19#, hashtblsz =300#;
            DEFINE hashtable=hsymtb#, hashstack=hstkhed#, hashptr
            =hsymptr#;
         %Byte pointers and fields%
            DEFINE strbyteptr =440700000001B#;
            DEFINE recprecordaddr =22B8#, recprecsize =2222B8#;
            DEFINE wordbyteptr =44B8#;
            DEFINE stringmax = 2222B8#, stringlength =22B8#;
            DEFINE leftbyteptrrj = 4307B8#;
         %Opcodes%
            DEFINE movei =201B#;
            DEFINE jrst =254B9#;
            DEFINE caml =311B9#, came =312B9#, camle=313B9#,
            camge=315B9#, camn =316B9#, camg =317B9#;
            DEFINE jsp = 265B9#;
            DEFINE opcodmask =9M9#;
            DEFINE opcall0=1#, opcall1=2#, opcallm=3#;
         %Sizes%
            DEFINE proctblsz = 10#, proctblentsz =2#;
            DEFINE maxwordsinstring = 402#;
            DEFINE maxprms = 20#;
            DEFINE bptblentsz = 4#, nbkpts=10#, bptblsz =40#;
            DEFINE ddtlimit = 500B#;
            DEFINE optabl = 24B#;
      %Record Information Block Defines (RIB)%
         %Recdesc defines%
            DEFINE recordstart = rd.raddr#;
            DEFINE fieldnumber = rd.rindex#;
            DEFINE numberfields = rd.nfield#;
            DEFINE recordsize = rd.rsize#;
            DEFINE mother = rd.momrec#;
            DEFINE callmother=&calprc←mother; calprc#;
            DEFINE daughter = rd.kidrec#;
            DEFINE recordname = rd.rdef#;
            DEFINE replaceaction = rd.replact#;
            DEFINE defaultreplace = 1#;
            DEFINE callreplaceproc = &calprc←rd.replact;calprc#;
            DEFINE overflow=1#;
            DEFINE undefaction = 0#, proceedaction = 1#;
      DEFINE overflowaction = rd.topact#;
      DEFINE fixoverflow =fixoutofbounds(&rd, overflowaction,
      overflow)#;
      DEFINE underflow=2#;
      DEFINE underflowaction = rd.botact#;
      DEFINE fixunderflow =fixoutofbounds(&rd, underflowaction,
      underflow)#;
      DEFINE dynamicrecord = rd.rdproc#;
      DEFINE defineproc=rd.rdef#;
      DEFINE calldefineproc = &calprc←defineproc; calprc#;
      DEFINE entity = rd.enttyp#;
      DEFINE wordtype = 1#, fieldtype=2#, recordtype=3#,
      stringtype=4#;
      DEFINE wordentity = entity=wordtype#;
      DEFINE fieldentity = entity=fieldtype#;
      DEFINE recordentity = entity=recordtype#;
      DEFINE stringentity = entity=stringtype#;
      DEFINE addressmode = rd.adrmod#;
      DEFINE macro = 0#, micro = 1#;
      DEFINE macromode = addressmode=macro#;
      DEFINE micromode =addressmode=micro#;
      DEFINE suppressloc = rd.suploc#;
      DEFINE initialised =rd.rdinit#;
      DEFINE rdsize = 6#; %Number of PDP10 words in record%
   %Record Field Pointers%
      DEFINE fieldstart = &rd+rdsize#;
      DEFINE fielddescsz=2#;
      DEFINE currentfld=&rd+rdsize+(IF fieldnumber >= numberfields
      THEN err(S"Field Out of bounds") ELSE
      fieldnumber)*fielddescsz#;
      DEFINE setfieldptr =&fld ← currentfld#;
      DEFINE fieldptr=fld.fpointr#;
      DEFINE typemode = fld.ftypmod#;
      DEFINE defaultmode = 0#;
      DEFINE numericmode = 1#, symbolicmode = 2#, textmode = 3# ;

      DEFINE typedefault=typemode=defaultmode#;
      DEFINE typenumeric = typemode=1#;
      DEFINE typesymbolic = typemode=2#;
      DEFINE typetext = typemode=3#;
      DEFINE typespecial =typemode>1000#;
      DEFINE typeroutine=typemode#;
      DEFINE calltyperoutine = &calprc←typeroutine; calprc#;
   %X-pointers%
      DEFINE xptrbytesperword=(36/fieldptr.xbytsz)#;
      DEFINE xptraddr = fieldptr.xptadd/xptrbytesperword#;
      DEFINE xptrbitpos =36-((fieldptr.xptadd MOD
      xptrbytesperword)*fieldptr.xbytsz)#;
      DEFINE xptrsize = fieldptr.xnbyte#;
      DEFINE xptrbytesz = fieldptr.xbytsz#;
      DEFINE xpointer = fieldptr.xptr=77B#;
   %Misz sizes, etc%
      DEFINE maxnumfields = 50#;
      DEFINE ribsize = 106#; %rdsize + maxnumfields*fielddescsz%

%DDT Symbol value and type defines%
```

```
      DEFINE symbolproc = symtype.spaddr#;                          0265
      DEFINE stacksymbol = symtype.stksym#;                         0266
      DEFINE ddtsymbol = symtype.ddtsm#;                            0267
      DEFINE sequencename = symtype.seqsym#;                        0268
      DEFINE fieldsymbol = symtype.fldsig#;                         0269
      DEFINE number = symtype.numsym#;                              0270
      DEFINE programsymbol = symtype.progsym#;                      0271
      DEFINE specialsymbol = stacksymbol .V sequencename#;          0272
      DEFINE indirectsym = indddtsym#  indirectmode=1B6#;           0273
      DEFINE internalsym = 2B6#, internalfield = internalsym + 1B9#,
      internalstacksym = internalsym+1B6#;                          0274
      DEFINE seqsymbol = 4B6#, stacksymv =1B6#;                     0275
      DEFINE stackref=sequence#;                                    0276
   %Data Block Defines (DB)%                                        0277
      DEFINE reg1 = db#;                                            0278
      DEFINE reg2 = db[1]#;                                         0279
      DEFINE reg3 = db[2]#;                                         0280
      DEFINE reg4 = db[3]#;                                         0281
      DEFINE reg5 = db[4]#;                                         0282
      DEFINE reg6 = db[5]#;                                         0283
      DEFINE reg7 = db[6]#;                                         0284
      DEFINE progp = db[7]#;                                        0285
      DEFINE progs = db[8]#;                                        0286
      DEFINE progm = db[9]#;                                        0287
      DEFINE progrp = db[10]#;                                      0288
      DEFINE rpreg = db[11]#;                                       0289
      DEFINE areg1 = db[12]#;                                       0290
      DEFINE areg2 = db[13]#;                                       0291
      DEFINE areg3 = db[14]#;                                       0292
      DEFINE areg4 = db[15]#;                                       0293
      DEFINE lv = db[16]#;                                          0294
      DEFINE escchar = db[17]#;                                     0295
      DEFINE symflg = db[18]#;                                      0296
      DEFINE framep = db[19]#;                                      0297
      DEFINE rfnmfg = db[20]#;                                      0298
      DEFINE currenttypemode = db[21]#;                             0299
      DEFINE nparms = db[22]#;                                      0300
      DEFINE framesize = db[23]#;                                   0301
      DEFINE framebase = db[24]#;                                   0302
      DEFINE framebasetype = 2#;                                    0303
      DEFINE stacktop = db[25]#;                                    0304
      DEFINE lbptbl = db[26]#;                                      0305
      DEFINE lbptblsz = nbkpts#;                                    0306
      DEFINE lbptblentsz = 1#;                                      0307
      DEFINE lc = db[36]#;                                          0308
      DEFINE recp = db[37]#;                                        0309
      DEFINE ddttablesz = 38#;                                      0310
      DEFINE frametop =[stacktop].marklocfield#;                    0311
         DEFINE frametoptype =1#;                                   0312
      DEFINE returnloc =[framep].retlocfield#;                      0313
      DEFINE framemark =[framep].marklocfield#;                     0314
%Temporary stuff-- remove when integrating -- COMMENTED OUT==CHI    0315
   SET sin = 52B, bkjfn = 42B,atljb = 437B,jobtm = 424B, pbout = 74B,
   sout = 53B, pbin = 73B, kfork = 153B, nout = 224B, haltf = 170B,
   psout = 76B, gtjfn = 20B, openf = 21B, rljfn = 23B, closf = 22B,
   bin = 50B, bout = 51B, nin = 225B, rout = 55B, sizef = 36B, dtach
```

```
    = 115B, gjinf = 13B, stdir = 40B, login = 1, rpcap = 150B, epcap =
    151B, sir = 125B, ati = 137B, eir = 126B, aic = 131B, reset =
    147B, sfbsz = 46B, sevec = 204B;%
REGISTER r0=0,r1 = 1, r2 = 2, r3 = 3, r4 = 4, r6 = 6, s = 11B, m =
12B, rp = 13B, a1 = 14B, a2 = 15B, a3 = 16B, a4 = 17B;
%Declarations%
   %Data Formats and meanings%
      %Record Information Block (RIB)
         There are two parts to a record information block:
            (1) Record Descriptor
               Identifies an entity generically
            Record Pointers...Point to th entity%
      (recdesc) RECORD %Record Descriptor...used for identifying  all
      entities used within DDT%
         raddr[18] , %Address of start of data area%
         rindex[18], %Index into record (by field)%
         nfield[18], %Number of fields in record%
         rsize[18], %Record size in PDP10 Words%
         momrec[18], %Address of rd of mother record%
            %0 if none%
         kidrec[18], %address of inferior record%
            %0 if none%
         rdef[36], %Address of record definition%
         replact[18], %Action to take when replacing value%
         topact[18], %Action for running off top of record%
         botact[18], %Action for running off bottom (end)%
         rdproc[1], %True if record defined by a procedure%
         enttyp[6], %Type of entity at last display%
         adrmod[1], %Address Mode%
         suploc[1], %Su[ress loc type flag%
         rdinit[1]; %True if record has been initialised%
      (ribent)RECORD %Format of entry in RIB%
         fpointr[36], %byte or x-pointer to field%
         ftypmod[18]; %Type out mode indicator...%
      (xpnter)RECORD %A Special Format Byte Pnter used in RIB's%
         xptadd[13], %Relative address in  bytes from start of
         record%
         xnbyte[11], %Numbr of bytes in field%
         xbytsz[6], %Byte Size%
         xptr[6]; %Must be 77 for Byte Pointer to be xpointer%
      %Symbol types, etc%
         %Assumes symtype contains type as returned from getsym%
         (symboltype)RECORD
            spaddr[18], %Address of routine for parsing special
            symbols%
            stksym[1], %stack symbol%
            ddtsm[1], %DDT Symbol%
            seqsym[1], %Sequence name%
            fldsig[1], %Field designator%
            numsym[1], %Number%
            progsym[1]; %Program symbol name%
      %Getsym options%
         (symopt)RECORD
            ddtadr[1], %Return address of ddt symbol%
            escape[1]; %Escape flag%
         DECLARE FIELD indddtsym=[0(rp), 1:18];
    (bptblent)RECORD %Break Pont Table Entry%
       bpaddr[18], %Address of breakpoint%
       bpinst[36], %Instructio replaced by break%
       bptest[36], %Breakpoint test instruction
          =0: Unconditional break
          1 = trace
          1-777777B =  test procedure address
          >777777B = Compare instruction
       %
       bpval[36]; %Value used with compare%
    (inst10)RECORD %PDP10 instruction parts%
       addr10[18],
       index10[4],
       indir10[1],
       accum10[4],
       opcod10[9];
    (instformat) RECORD pdummy[21], longopcode[15];
    DECLARE FIELD retlocfield = [0(rp), 18:0],
       siglocfield = [0(rp), 18:18],
       marklocfield = [-1(rp), 18:0];
    % comment out -- move writeable data to DATA === CFD
       DECLARE STRING ddtlit[150];
       DECLARE bptbl[bptblsz]; %%nbkpts * bpentsz%%
       DECLARE EXTERNAL db;
       DECLARE ddtrloc, ddtrp, ddtinst, ddtjunk, ddtrg1, ddtrg2,
       ddtrg3; %%Globals used by ddt during processing of
       breakpoints%%
       DECLARE hsymtb[hashtblsz], hstkhed[hashbase], hsymptr;
       DECLARE levtbl, displayflag;
       DECLARE proctbl[proctblsz];
    === %
    DECLARE symloc = 116B, smanipumask = 77774B7, smanipulator
    =27044B7;
    REF db;
%Control Procedures%
  (ddtcalls)PROC:
    %Various calling routines for ddt%
    %General register saving routine...saves 2-17B, assumes that 1(s)
    may be clobbered.  assumes r1 contains dest on call%
       (ddtsvr1):
          !MOVEM 0, ddtrg3;
          !MOVEI 0,16B(1); %Destination end%
          !HRLI 1,2;
          !BLT 1,@0: %Do the move%
          !JRST @ddtrg3; %RETURN%
       (usvreg): %Save off the user registers, r1-r6 and a1-a4%
          !MOVEM 0,ddtrg1;
          !PUSH s,1;
          !MOVEI 1,1(s);
          !JSP ddtsvr1; %Registers on stack now%
          !POP s,0(s);
          !HRRZ 1,db; %Destination%
          !MOVEI 0,5(1); %Dest%
          !HRLI 1,1(s); %Source%
          !BLT 1,@0: %r1 - r6%
          areg1 ← [s+$a1]:
```

```
      areg2 ← [s+$a2];
      areg3 ← [s+$a3];
      areg4 ← [s+$a4];
      !JRST @ddtrg1;
   (urstrg):
      !MOVEM 0,ddtrg3;
      r1.LH ← $areg1;
      r1.RH ← $a1;
      r0 ← $a4;
      !BLT 1,@0; %a1 - a4%
      !MOVE 1,db;
      !HRLI 1,1(1);
      !HRRI 1,2;
      !MOVEI 0,6;
      !BLT 1,@0;
      !MOVE 1,@db;
      !JRST @ddtrg3;
   (callddt): %Garden variety call%
      !SETZM 3(s); %Not called by breakpoint%
   (ddtcallcom):
      %Common call code...r0 contains return loc, 3(s) break parm%
      !HRRZM 0,2(s); %Save off  eturn loc%
      !MOVEM 1,4(s); %Save off r1%
      !MOVEI 1,5(s); %Destination for BLT%
      !JSP ddtsvr1; %Save 2-17%
      !PUSH s,m;
      !PUSH s,1(s);
      m ← s;
      !PUSH s,1(s);
      GOTO runddt;
   (breakddt):
      %Breakpoiont call...top of stack contains break loc%
      !POP s,2(s); %Move break loc to parm position%
      GOTO ddtcallcom;
 END.
(nddtarm)PROC; % called from NLS %
 s#thint();
 RETURN;
 END.
(nddtdisarm)PROC; % called from NLS %
 unsethint();
 RETURN;
 END.
(runddt)PROC(breakadr);
 LOCAL dblock[ddttablesz]; %Must be first thing on stack%
 LOCAL xdb;
 LOCAL  c;
 LOCAL dspmode, dspsave;
 LOCAL xnlmode, xbuffs, xbuffn;
 LOCAL rd[ribsize];
 LOCAL i, tbuff[40];
 LOCAL xrawchr; % saved getchar dispatch %
 %Save off global values%
    % save input buffer (buff) into temporary buffer (tbuff) %
   FOR i ← 0 UP UNTIL > buffsz DO
```

```
      tbuff[i] ← buff[i];
   xbuffs ← buffs;
   xbuffn ← buffn;
   xdb ← &db;
% save rawchar dispatch and supply normal one%
   %done so record/playback and auxinput stuff can use nddt%
   xrawchr ← rawchr;  %save current away%
   rawchr ← $getchar; %and use system standard%
%Initialise%
   % save away display area if in DNLS and no tty simulation area
   has been allocated %
   %Save away nlmode; set it to typewriter for proper input%
      IF (xnlmode ← nlmode) THEN
         BEGIN
         IF defttysim THEN
            BEGIN % dnls and no tty simulation area %
            !JSYS 451B; % tsnda %
            displayflag ← FALSE;
            dspsave ← 2;
            END
         ELSE
            BEGIN % dnls and tty simulation area %
            displayflag ← TRUE;
            dspsave ← 1;
            END;
         % set control character output codes %
            r1 ← 18M;
            r2 ← ttycoc;
            r3 ← 1B3;
            !JSYS sfcoc;
         END
      ELSE
         BEGIN % tty device %
         displayflag ← FALSE;
         dspsave ← FALSE;
         END;
      nlmode ← typewriter;
   &db ← $dblock;
   %When called, dblock [0-14] contains registers 1-17%
   BUMP rubabt; %For stopping loops  n debugging mode%
   %Set up frame pointer%
      stacktop ← m.RH;
      framebase ← $gstack+2;
      framep ← frametop;
      initframe();
   %Set up other stuff%
      recp ← rpreg;
      escchar ← ': ;
      symflg ← TRUE;
      rfnmfg ← TRUE; % turn on printing of names of records %
      lv ← 0;
      lc ← 0;
      c ← -1;
      WHILE (c←c+1) <= lbptblsz DO [$lbptbl+c] ← 0;
   %Restore all breakpoints, and type message if called from
   breakpoint.  Notice that in order to use breakpoints in nddt,
  you must call nddt from nddt (↑H), set the breakpoint, and then
  return with a continue%
     restbp();
     IF breakadr THEN
        BEGIN
        typbrk(breakadr-1):
        END;
  %Initialise Symbol Table%
     initht();
%Now start parsing%
  clrbuf(0);
  typeas($"NLS DDT");
  LOOP
     BEGIN
     ON SIGNAL
        =quitsig:
           IF breakadr THEN callsig(continsig)
           ELSE
              BEGIN
              % set sysmsg to 0 so that it won't fake out the
              parser %
                 sysmsg ← 0;
              %Restore globals%
              rawchr ← xrawchr:
              &db ← xdb;
              FOR i ← 0 UP UNTIL > buffsz DO buff[i] ← tbuff[i];

              buffn ← xbuffn;
              buffs ← xbuffs;
              nlmode ← xnlmode:
              IF dspsave THEN
                 BEGIN
                 % set control character output codes %
                    r1 ← 18M;
                    r2 ← dpycoc;
                    r3 ← 1B3;
                    !JSYS sfcoc;
                 %  restore DNLS display area if it was saved %

                    IF dspsave = 2 THEN
                       !JSYS 452B; %tsfda%
                 END:
              RETURN:
              END;
        =continsig:
           BEGIN
           %Restore globals%
           % reset sysmsg so system won't think it has to put out
           an error message %
              sysmsg ← 0;
           rawchr ← xrawchr;
           &db ← xdb:
           FOR i ← 0 UP UNTIL > buffsz DO buff[i] ← tbuff[i];

           buffn ← xbuffn;
           buffs ← xbuffs;
        nlmode ← xnlmode;
        IF dspsave THEN
           BEGIN
           % set control character output codes %
              r1 ← 18M;
              r2 ← dpycoc;
              r3 ← 1B3;
              !JSYS sfcoc;
           %  restore DNLS display area if it was saved %
              IF dspsave = 2 THEN
                 !JSYS 452B; %tsfda%
           END;
        !MOVEI 17B; %Last word to move into%
        !HRLZI 1,3(m); %From%
        !HRRI 1,2; %to%
        !BLT 1,@0; %xfer regs 2-17%
        %Note that s now points to the proper m-2 for the
        frame we are actually in, and m has been restored to
        previous frame%
        !MOVE 1,4(s); %Restore register 1%
        !HRRZ 0,2(s); %Return loc%
        !JRST @0; %RETURN%
        END;
     =gosig:
        BEGIN
        %Sysmsg contains stat location%
        %fix up breakad and fake a return from a breakpoint%
        breakadr ← sysmsg+1; %One past first instruction is
        proper return loc%
        [m].retlocfield ← $breakret;
        callsig(continsig);
        END;
     =statesig: REPEAT LOOP;
     ELSE
        BEGIN
        IF sysmsg > 1000 THEN
           BEGIN
           crlf();
           typeas(sysmsg);
           END;
        REPEAT LOOP;
        END;
  crlf();
  typech('>);
  echoff();
  CASE nextchar OF
     ='B:  %Breakpoint set, clear, print%
        xbrkpt();
     ='C:  %Continue program -- return from ↑H%
        xcontin();
     ='D: % Define symbol %
        xdddef();
     ='F:  % find content %
        xddfind();
     ='G: %go to location%
```

```
                 xgo();
               ='M:  % mark symbol table %
                 xnmark();
               ='P:  %Procedure call, replace, backup%
                 xprocc();
               ='R:  %Record Pointer Set%
                 xrpset();
               ='S:  %Show Record, String, Location, Preceeding, Next%
                 xnshow($rd, 1);
               ='T:   %Trace%
                 xtrace();
               ='V:  %Value of symbol%
                 xtval();
               ='=: %type numeric value of symbol%
                 BEGIN
                 typech('=);
                 typval(lv, numericmode);
                 END:
               =LF,  %show next%
               ='←,   % assign a value %
               ='↑,  %show previous%
               =TAB:  %show value of current symbol%
                 BEGIN
                 backupchar;
                 xnshow($rd, 0);
                 END:
               ENDCASE typeas($" ??"):
             END;
        RETURN;
        END.
%* Low Level syntactic recognizers %
   (chkcacr)PROC;
        LOCAL char;
        char ← nextchar:
        IF char # CA AND char # EOL THEN err($"?? CA Expected");
        RETURN;
        END.
   (deblnk)PROC:
        WHILE nextchar = SP DO NULL;
        backupchar;
        RETURN;
        END.
   (octnum)PROC(value);
        LOCAL bp;
        %Uses ddtlit%
        bp ← r1 ← strbyteptr + $ddtlit;
        r2 ← value:
        r3 ← 8;
        IF NOT SKIP !JSYS nout THEN err($"Error in Nout");
        ddtlit.L ← slngth(bp, r1);
        RETURN($ddtlit):
        END.
   (rdnum)PROC( base):
        LOCAL char, bp, v;
        LOCAL STRING tempsr[20];
   %Read a number, and return value plus number if ok%
   IF (char ← nextchar) IN ['0, '9] OR char = '-   THEN
      BEGIN
      *tempsr* ← char;
         LOOP
            BEGIN
            CASE char ← nextchar OF
               =BC:
                  IF tempsr.L > 0 THEN
                     BEGIN
                     typech( '\ );
                     typech( *tempsr*[tempsr.L] ); % echo last
                     character %
                     tempsr.L ← tempsr.L-1;
                     END;
               =BW, =21B: RETURN( rdnum(base) );
               IN ['0, '9]: *tempsr* ← *tempsr*, char;
               ENDCASE
                  BEGIN
                  backupchar;
                  bp ← strbyteptr + $tempsr;
                  *tempsr* ← *tempsr*, 0;
                  IF NOT ninjsys($bp, base: v) THEN err($"Illegal
                  Number");
                  RETURN(TRUE, v);
                  END;
            END;
      END;
   backupchar;
   RETURN(FALSE);
   END.
(rdsym)PROC(astr);
   LOCAL char;
   REF astr;
   %Read a symbol into astr, returning false if none%
   *astr* ← NULL;
   IF (char ← nextchar) IN ['A, 'Z] THEN
      LOOP
         BEGIN
         CASE char OF
            =BC:
               IF astr.L > 0 THEN
                  BEGIN
                  typech( '\ );
                  typech( *astr*[astr.L] ); % echo last character %

                  astr.L ← astr.L-1;
                  END;
            =BW, =21B: astr.L ← 0;
            IN ['A, 'Z], IN ['0, '9]: *astr* ← *astr*, char;
            ENDCASE
               BEGIN
               backupchar;
               RETURN(astr.L);
               END;
         char ← nextchar;
         END:
      backupchar;
      RETURN(FALSE);
      END.
%* Typeout routines %
   (trfname)PROC(bp, rd);
      %Types out a field of a record with the appropriate name%
      LOCAL STRING tempsr[40]:
      REF rd;
      ddgtsymbol($tempsr, recordname.RH + fieldnumber);
      crlf();
      typeas($tempsr):
      IF currenttypemode = numericmode THEN typech('/)
      ELSE IF currenttypemode = textmode THEN typech('")
      ELSE typech('/);
      typval(.bp, currenttypemode);
      RETURN;
      END.
   (typentity)PROC(rd, curmode);
      %Types out the entity described by rd in appropriate manner%
      REF rd;
      lc ← recordstart;
      IF micromode THEN typfield(&rd, curmode)
      ELSE
         IF numberfields = 0 THEN
            BEGIN
            crlf();
            typeas($"Record Empty");
            END
         ELSE
            BEGIN
            fieldnumber ← 0:
            WHILE fieldnumber < numberfields DO
               BEGIN
               typfield(&rd, curmode):
               fieldnumber ← fieldnumber + 1
               END;
            END:
      RETURN:
      END.
   (typfield)PROC(rd, curmode);
      %rd = address of Record Descriptor, curmode = default mode%
      LOCAL index, fld, mode, t, bp, calprc, count;
      REF rd, fld, calprc:
      setfieldptr:
      IF typespecial THEN
         BEGIN
         calltyperoutine((fieldptr + recordstart) .A noindexmask, &rd);

         END
      ELSE
         BEGIN %Type otherwise%
         mode ← IF typedefault THEN curmode ELSE typemode;
         IF xpointer THEN
            BEGIN
            bp ← 0;
            count ← xptrsize;
            bp.bpadr ← recordstart + xptraddr;
            bp.bpbitpos ← xptrbitpos;
            bp.bpsize ← xptrbytesz;
            WHILE (count ← count - 1) > -1 DO
               typval(↑bp, mode);
            END
         ELSE
            BEGIN %An ordinary poiter%
            bp ← (fieldptr + recordstart) .A noindexmask;
            typval(.bp, mode);
            END;
         END;
      RETURN;
      END.
   (typloc)PROC(rd);
      LOCAL fld, mode;
      LOCAL STRING tempsr[75];
      REF rd, fld;
      IF suppressloc THEN RETURN;
      setfieldptr;
      mode ← IF typedefault THEN currenttypemode ELSE typemode;
      IF mode = numericmode THEN *tempsr* ← *[octnum(recordstart)]*
      ELSE ddgtsymbol($tempsr, recordstart);
      typeas($tempsr);
      IF NOT typespecial THEN
         BEGIN
         IF mode =  numericmode THEN typech('[)
         ELSE IF mode = textmode THEN typech('")
         ELSE typech('/);
         END;
      RETURN;
      END.
   (typval)PROC(value, mode);
      LOCAL bp;
      LOCAL STRING tempsr[75];
      lv ← value;
      IF mode = numericmode THEN
         IF value.LH # 0 THEN
            *tempsr* ← *[octnum(value.LH)]*, ",,", *[octnum(value.RH)]*

         ELSE
            *tempsr* ← *[octnum(value)]*
      ELSE IF mode = textmode THEN
         BEGIN
         bp ← leftbyteptrrj + $value;
         WHILE bp.bpadr = $value DO IF ↑bp # 0 THEN typech(.bp);
         RETURN;
         END
      ELSE
         ddgtsymbol($tempsr, value);
      typeas($"    ");
      typeas($tempsr);
      RETURN;
      END.
%** Character I/O Routines %
```

```
   (binjsys)PROC;
      LOCAL char;
      RETURN(CASE (char ← inpcuc()) OF
         =CD: callsig(statesig);
         ENDCASE char);
      END.
   (bkjfnjsys)PROC;
      IF (buffs ← buffs - 1) < 0 THEN buffs ← buffsz;
      RETURN (buff[ buffs ]);
      END.
   (ninjsys)PROC(inpbp, base);
      LOCAL rflag, char, v;
      REF inpbp;
      rflag ← 0;
      r1 ← inpbp;
      r3 ← base;
      IF NOT SKIP !JSYS nin THEN rflag ← TRUE;
      inpbp ← r1;
      RETURN(NOT rflag, r2);
      END.
%* Parsing Routines - in Alphabetical order by command name %
   %* Addresses and values%
      (chngadr)PROC(rd,value);
         REF rd;
         IF NOT initialised THEN err($"Relative addressing attempted on
         unitialised entity");
         IF value = 0 THEN RETURN;
         IF macromode THEN
            BEGIN
            WHILE value DO
               IF value < 0 THEN
                  BEGIN
                  fixunderflow;
                  value ← value + 1;
                  END
               ELSE
                  BEGIN
                  fixoverflow;
                  value ← value - 1;
                  END;
            addressmode ← macro;
            END
         ELSE
            BEGIN
            IF (fieldnumber ← fieldnumber + value) NOT IN [0,
            numberfields] THEN
               BEGIN
               IF fieldnumber < 0 THEN fixunderflow
               ELSE fixoverflow;
               addressmode ← micro;
               END;
            END;
         RETURN;
         END.
      (fielddesig)PROC(rd);
         LOCAL value, fld, symtype, symval;
   REF rd, flq;
   IF NOT wordentity AND NOT fieldentity THEN RETURN; %Strings AND
   records may not be followed by fields%
   deblnk();
   IF nextchar # '. THEN
      BEGIN
      backupchar;
      RETURN;
      END;
   deblnk();
   IF NOT getsym( :symtype, symval)
   AND NOT programsymbol
   AND NOT fieldsymbol THEN
      err($"Illegal Symbol in Address");
   IF NOT ckbptr(value ← [symval]) THEN err($"Illegal Field
   Designator");
   setfieldptr;
   IF wordentity THEN fieldptr.bpadr ← value.bpadr
   ELSE
      IF value.bpadr # 0 THEN err($"Illogical Concatenation of
      fields")
      ELSE value.bpadr ← fieldptr.bpadr;
   IF fieldptr.bpsize <= value.bpbitpos
   OR (value.bpbitpos ← value.bpbitpos + fieldptr.bpbitpos) > 35

   OR (value.bpsize ← MIN(fieldptr.bpsize, value.bpsize,
   (fieldptr.bpbitpos ← fieldptr.bpsize) - value.bpbitpos)) <= 0
   THEN err($"Illogical concatenation of fields");
   value.bpindx ← 0;
   entity ← fieldtype;
   fieldptr ← value;
   fielddesig(&rd);
   RETURN;
   END.
(fixoutofbounds)PROC(rd, action, direction);
   LOCAL calprc;
   REF rd, calprc;
   IF action = undefaction THEN err($"Address Out Of Bounds");

   IF action = proceedaction THEN
      recordstart ← recordstart + (IF direction = overflow THEN
      recordsize ELSE -recordsize)
   ELSE
      BEGIN
      &calprc ← action;
      calprc(&rd);
      END;
   initialised ← FALSE;
   initrd(&rd);
   RETURN;
   END.
(getop)PROC;
   RETURN( CASE nextchar OF
      =SP, ='+: addop;
      ='-: subop;
      ='*: mulop;
      ='': divop;
      ENDCASE IF backupchar THEN 0 ELSE 0);
   END.
(getsym)PROC;
   LOCAL option, symtype, symval, char, v1, v2, modesave, bp;
   LOCAL STRING tempsr[30];
   %Check for escape, etc%
      deblnk();
      option ← 0;
      CASE char ← nextchar OF
         = '=:
            BEGIN
            option.ddtadr ← 1; %DDT symbol return rela address not
            contents%
            REPEAT CASE;
            END;
         = escchar:
            BEGIN
            option.escape ← 1; %escape character%
            REPEAT CASE;
            END;
         ENDCASE;
   symtype ← symval ← 0;
   backupchar;
   IF (char = CA) AND displayflag THEN % got a bug %
      BEGIN
      *tempsr* ← NULL;
      modesave ← nlmode; % fake out display mode %
      nlmode ← 1; % full display %
      LOOP
         BEGIN
         INPUT NAME tempsr;
         CASE char ← nextchar OF
            =BC:
               BEGIN
               bmoff(); dn($"  "); % reset stuff %
               REPEAT LOOP;
               END;
         ENDCASE
            BEGIN
            backupchar;
            EXIT LOOP;
            END;
         END;
      bmoff(); dn($"  "); % reset stuff %
      nlmode ← modesave;
      echon();
      echo( $tempsr ); % echo it %
      IF char # CA THEN typech(char);
      IF tempsr.L > 0 THEN
         IF *tempsr*[1] IN ['0, '9] THEN
            BEGIN % got a number%
            bp ← strbyteptr + $tempsr;
            *tempsr* ← *tempsr*, 0;
            IF NOT ninjsys($bp, radix: symval) THEN err($"Illegal
            Number");
         GOTO gotnum;
         END
      ELSE GOTO gotsym;
   END;
IF rdnum( radix: symval) THEN
   BEGIN %Number%
   (gotnum):
   number ← TRUE;
   IF nextchar = ', THEN %Possible Half word Format%
      IF nextchar = ', THEN
         BEGIN
         symval.LH ← symval;
         IF NOT rdnum( radix: v1) THEN err($"Illegal Number");
         symval.RH ← v1;
         END
      ELSE BEGIN
      backupchar; % back over non-comma %
      backupchar; % back over comma %
      END
   ELSE backupchar; % back over non-comma %
   END
ELSE
   BEGIN
   IF NOT rdsym($tempsr) THEN RETURN(FALSE);
   (gotsym):
   IF symflg .X option.escape # 0 THEN %Check as a DDT symbol%
      IF ddtsym($tempsr: symtype, v2) THEN
         BEGIN
         IF ddtsymbol THEN %Simple ddt internal symbol%
            BEGIN
            v2 ← [v2]+$regl; %Actual address for this version%
            symval ← IF v2.indirectsym THEN
               IF option.ddtadr THEN v2.RH
               ELSE [v2.RH]
            ELSE v2.RH;
            END
         ELSE symval ← v2;
         GOTO comsym;
         END;
   IF option.ddtadr THEN err($"Not DDT Symbol after '="); 
   %Now do a ddt lookup%
      IF NOT ddtlookup($tempsr, option.escape: symval) THEN
         BEGIN
         *ddtlit* ← "No Such Symbol ", *tempsr*;
         err($ddtlit);
         END;
      programsymbol ← TRUE;
   END;
%Common symbol lookup stuff%
   (comsym):
RETURN(TRUE, symtype, symval);
END.
```

```
(oper)PROC(value, valuel, op);
   value ← CASE op OF
      =addop: value+valuel;
      =subop: value-valuel;
      =mulop: value*valuel;
      =divop: value/valuel;
      ENDCASE err($"Illegal Operator");
   RETURN(value);
   RETURN;
   END.
(parsenter) PROCEDURE;
   % this routine parses the current input stream looking for a
   value being typed in -
   It returns: FALSE if no value is typed in
               TRUE and the value if given %
   LOCAL value, field, i, char;
   LOCAL STRING tempstr[10];
   % look at the next source char but do not read it yet %
   char ← nextchar;
      backupchar;
   % process according to what was typed in %
   CASE char OF
      =CA, =C., =CR:  % nothing given %
         RETURN (FALSE);
      IN ['A, 'Z]:  % start of a symbol %
         BEGIN
         rdsym( $tempstr ); % gather symbol %
         % find out if it is a PDP10 instruction mnemonic %
         IF NOT( value← insym($tempstr)) THEN
            % not mnemonic -- return symbol to input buffer %

            FOR i ← 1 UP UNTIL > tempstr.L DO backupchar
         ELSE LOOP
            BEGIN  % parse the PDP10 instruction %
            % get next field - perform range check to trap out
            values for the symbolic register names %
            IF (field ← parsevalue(0,0,0)) IN [$reg1, $areg4] THEN
            field ← field - $reg1 + 1;
            CASE char ← nextchar OF
               =CA, =C., =CR:  % all done %
                  BEGIN
                  value.addr10 ← field;
                  backupchar;
                  RETURN (TRUE, value);
                  END;
               =',:  % an accumulator field was given %
                  value.accum10 ← field;
               ='@:  % indirect address %
                  value.indir10 ← TRUE;
               ='(:  % index coming up -- last field must be addr
               %
                  value.addr10 ← field;
               ='):  % got index %
                  BEGIN
                  value.index10 ← field;
                  field ← value.addr10;
                  REPEAT CASE;
                  END;
            ENDCASE err($"Illegal Instruction Format");
            END; % of parse loop %
         END; % of symbol case %
      =''': %start of a string %
         BEGIN
         *ddtlit* ← NULL; % use ddtlit to collect string %
         nextchar; % read over the initial " %
         LOOP
            CASE char ← nextchar OF
               =BC: ddtlit.L ← MAX(ddtlit.L-1,0);
               ='": EXIT LOOP; % read terminating " %
            ENDCASE *ddtlit* ← *ddtlit*, char;
         RETURN (TRUE, $ddtlit);
         END;
   ENDCASE;
   IF (value ← parsevalue(0,0,0)) IN [$reg1, $areg4] THEN value ←
   value - $reg1 + 1;
   RETURN (TRUE, value);
   END.
   (insym) PROCEDURE( symbol );
      % this routine identifies the given symbol as a PDP10
      instruction and returns the skeleton for the instruction if
      found, FALSE otherwise %
      LOCAL sixbit, i, value;
      REF symbol;
      % range check on length of symbol %
      IF NOT (symbol.L IN [1, 6]) THEN RETURN (FALSE);
      sixbit ← 0;
      value ← 0;
      % convert symbol to sixbit form - left adjusted %
      FOR i ← 1 UP UNTIL > 6 DO
         sixbit ← shl(sixbit,6) +
            (IF i <= symbol.L THEN*symbol*[i] - 32 ELSE 0);
      % try to match sixbit value with optab entries %
      FOR i ← 1 UP UNTIL >= 700B DO % range of opcodes %
         IF optab[i] = sixbit THEN
            BEGIN
            value.opcod10 ← i;
            RETURN (value);
            END;
      % try aux table for long opcodes %
      FOR i ← 0 UP UNTIL >= optab1 DO
         IF optab3[i] = sixbit THEN
            BEGIN
            value.longopcode ← optab2[i];
            RETURN (value);
            END;
      % no can find-- give up %
      RETURN (FALSE);
      END.
(parsevalue)PROC(rd, symtype, symval);
   LOCAL op, value, defvalue;
   REF rd;
   IF symtype = 0 THEN %Get the first symbou%
```

```
      IF defvalue ← NOT getsym(: symtype, symval) THEN
        BEGIN
        value ← IF &rd THEN recordstart ELSE 0;
        GOTO parsewrapup;
        END;
    IF specialsymbol THEN err($"Illegal Address");
    value ← symval;
    LOOP
      BEGIN
      IF (op ← getop()) = 0 THEN EXIT LOOP;
      IF (NOT getsym( :symtype, symval)) OR specialsymbol THEN
      err($"Illegal Address");
      value ← oper(value, symval, op);
      END;
    (parsewrapup):
    IF &rd THEN
      BEGIN
      recordstart ← value;
      initrd(&rd); %Iniialise record%
      fielddesig(&rd);
      END:
    RETURN(value, defvalue);
    END.
  %Replace%
    (replfield)PROC(rd);
      %This procedure accepts an expression, and replaces the
      contents of the field indicated by rd with its value.  It
      expects the expression to be terminated with a CA.%
      LOCAL value, char:
      REF rd;
      IF macromode THEN
        IF numberfields = 1 THEN fieldnumber ← 0
        ELSE err($"Illegal Replace")
      ELSE
        IF fieldnumber >= numberfields THEN err($"Illegal
        Replace");
      IF (char ← nextchar) = CA OR char = EOL THEN RETURN; %Empty%
      backupchar;
      IF parsenter( :value) THEN setfield(&rd, value);
      chkcacr():
      RETURN;
      END.
    (replrecord)PROC(rd);
      %REplace the record indicated by rd with a series of
      values::
        EMPTY CA means don't change it
        Value CA means replace it with value
        %
      %For each field, issue CRLF and type name of field%
      LOCAL value. fld;
      LOCAL STRING tempsr[40];
      REF rd, fld;
      IF NOT initialised THEN err($"Illegal Replace");
      fieldnumber ← 0:
      crlf();
```

```
        LOOP
           BEGIN
           setfieldptr;
           %First get name of field, and type it%
              IF NOT dynamicrecord THEN
                 BEGIN
                 ddgtsymbol($tempsr, (recordname + fieldnumber) .A
                 18M);
                 typeas($tempsr);
                 typeas($" ← ");
                 END;
           %Now get value if thre is one%
              CASE nextchar OF
                 =CA, =CR: NULL; %Ignore this field%
                 ENDCASE
                    BEGIN
                    backupchar;
                    IF parsenter( :value) THEN setfield(&rd, value);

                    chkcacr(); % read over CA or CR %
                    END;
           IF (fieldnumber ← fieldnumber + 1) = numberfields THEN
           EXIT;
           END;
        typeas($"
        End of Record");
        RETURN;
        END.
     (repval)PROC(rd);
        LOCAL calprc, char;
        REF rd, calprc;
        %Parse last part of show command, and replace value as
        required%
        echoff();
        IF (char ← nextchar) # '← THEN
           BEGIN
           IF char = '[ OR char = '/ THEN RETURN ;
           backupchar;
           chkcacr();
           RETURN;
           END;
        typeas($" ← ");
        echon();
        IF replaceaction = 0 THEN err($"Contents of This Entity may
        not be changed");
        IF replaceaction = defaultreplace THEN
           IF macromode THEN replrecord(&rd)
           ELSE replfield(&rd)
        ELSE
           BEGIN
           callreplaceproc(&rd);
           END;
        IF dynamicrecord THEN
           BEGIN
           initialised ← FALSE;
           initrd(&rd);
```

```
          END;
        RETURN;
        END.
      (setfield)PROC(rd, value);
        %Accepts a rdecord descripton, and a value.  Sets the
        defined field to the value%
        LOCAL fld. bp;
        REF rd, fld;
        IF NOT initialised THEN err($"Illegal Replace");
        setfieldptr;
        bp ← (fieldptr + recordstart) .A noindexmask;
        .bp ← value;
        RETURN;
        END.
    (sequence)PROC(rd. symtype, symval);
      %Standard parms...Parse Sequence%
      REF rd;
      copyrd(symval+1, &rd);
      RETURN;
      END.
    (showaddr)PROC(rd);
      %Parms are:
        rd = address of rd
      Returns updated fd%
      LOCAL symtype, symval;
      REF rd;
      echon();
      IF NOT getsym( :symtype, symval) THEN parsevalue(&rd, 0, 0)

      ELSE IF stacksymbol THEN stackref(&rd, symtype, symval)
           ELSE IF sequencename THEN sequence(&rd, symtype, symval)

                ELSE parsevalue(&rd, symtype, symval);
      IF NOT initialised THEN initrd(&rd);
      RETURN;
      END.
  %* BREAKPOINT Routines%
    (bpclall)PROC;
      %Clear all breakpoints%
      LOCAL bpn, bpa;
      bpn ← -1;
      WHILE (bpn ← bpn+1) < nbkpts DO
        IF (bpa ← findbp(bpn)) THEN
          BEGIN
          bpclr(bpa):
          END;
      RETURN;
      END.
    (bpclr)PROC(bpa);
      LOCAL c;
      %Arg is address of bp%
      IF tblsearch($lbptbl. lbptblsz. lbptblentsz, bpa: c) THEN [c] ←
      0; %Clear local table entry%
      [bpa] ← 0;
      RETURN;
      END.
```

```
(bppntall)PROC;
   %Print all breakpoints%
   LOCAL bpn;
   bpn ← -1;
   WHILE (bpn ← bpn+1) < nbkpts DO
      IF findbp(bpn: bpn) THEN typbrk(bpn);
   RETURN;
   END.
(breakpoint)PROC;
   %Get hee to check whether or not  o execute a break...calld
   with a !JSP brkchk%
   (brkchk):
      !HRRZM O,ddtrloc; %Save off return address%
      !JSP ddtsvrg; %Save off regusters%
      IF (ddtrp ← findbp(ddtrloc-1)) = O THEN
         BEGIN
         crlf();
         typeas($"Unrecognised Break Point");
         !JSP ddtresreg;
         !JRST @ddtrloc;
         END;
      CASE [ddtrp].bptest OF
         =O: NULL;
         =1: %Trace%
            BEGIN
            typtrace(ddtrloc-1);
            GOTO bkpret;
            END;
         < 1B7: %Procedure Call..T/F%
            BEGIN
            ddtinst ← [ddtrp].bptest; %Address of proc to be
            called%
            !JSP ddtrlreg; %restore registers except O and s%

            !CALLO @ddtinst; %11 Routine%
            IF NOT al THEN GOTO bkpret;
            END; %Fall Through to break%
         ENDCASE  %Fall through to break%
            BEGIN
            ddtjunk ← [ddtrp].bpval;
            ddtinst ← [ddtrp].bptest; %Instruction%
            !JSP ddtrlreg; %Restore registers except s and O%

            rO ← ddtjunk;
            !XCT ddtinst;
            GOTO bkpret; %Test Failed%
            END; %Fall through to break%
      %By here, we will execute the break point%
         %Resyore registers%
            !JSP ddtresreg;
         %Now call ddt, with breakpoint address on top of stack%

            !PUSH s,ddtrloc; %Return address in case breakpoint
            cleared%
            !JSP breakddt; %Assumes top of stack contains break
            address...returns with break address in 3Z(s)%
```

```
    %Returning from ddt....proceed%
       (breakret): %Returning from breakpon execution%
       %Restore return address at 3(s)%
          !MOVE 3(s);
          !MOVEM ddtrloc;
       %Save registers%
          !JSP ddtsvrg;
       %See if breakpoit is still valid%
          IF (ddtrp ← findbp(ddtrloc-1)) = 0 THEN %No More Brek
          point%
             %No break any more...restore registers and return%

             BEGIN
             ddtrloc ← ddtrloc-1;
             !JSP ddtresreg;
             !JRST @ddtrloc;
             END;
    %Global breakpoint return code%
       (bkpret):
          %Breakpoint is still there...execute instruction and
          proceed%
             ddtinst ← [ddtrp].bpinst;
             !JSP ddtresreg;
             !XCT ddtinst;
             !JRST @ddtrloc;
 %Miscellaneous breakpoint routines%
    (ddtsvrg): %Save registers on stack%
       ddtrg1 ← r0;
       ddtrg2 ← r1;
       !MOVEI 1,2(s); %Location to save registers%
       !JSP ddtsvrl; %Save registers 2-17%
       !PUSH s,ddtrg2; %Place original r1 into proper place%

       !HRRI 1,16B;
       !HRL 1,1;
       !ADD s,1; %Fix up s%
       !JRST @ddtrg1; %RETURN%
    (ddtresreg): %Restore the registers%
       ddtrg1 ← r0;
       r1.LH ← s.RH-15B; %Location of registers%
       r1.RH ← 2; %Where they are to go%
       r0 ← 17B; %Last loc%
       !BLT 1,@0;
       !MOVE 1,1(s); %Restore 1%
       !JRST @ddtrg1; %RETURN%
    (ddtrlreg): %Restore the registers except for s%
       ddtrg1 ← r0;
       r1.LH ← s.RH-15B; %Location of registers%
       r1.RH ← 2; %Where they are to go%
       r0 ← 17B; %Last loc%
       !MOVEM s,ddtrg2 ; %Save off s%
       !BLT 1,@0;
       !MOVE 1,1(s); %Restore 1%
       !MOVE s,ddtrg2;
       !JRST @ddtrg1;
 END.
```

```
(chkbp)PROC(loc);                                                   01411
   %This procedure checks the bkpt num/ addr passed for first a
   stack manipulation instruction, and secondly an already
   defined breakpoint.                                              01412
   Returns addr updated for stack manip and breakpoints as first
   result.                                                          01413
   If previously defined breakpoint, returns breakpoint address +
   number%                                                          01414
   LOCAL bpa, bpn;                                                  01415
   bpa ← bpn ← 0;                                                   01416
   IF (loc > nbkpts) AND [loc] .A smanipumask = smanipulator THEN
                                                                    01417
      BEGIN                                                         01418
      BUMP loc;                                                     01419
      END;                                                          01420
   IF (bpa ← findbp(loc: bpn)) THEN                                 01421
      BEGIN %Brekpoint already there%                               01422
      loc ← [bpa];                                                  01423
      END;                                                          01424
   RETURN(loc, bpa, bpn);                                           01425
   END.                                                             01426
(findbp)PROC(value);                                                01427
   %IF value < nbkpts then assume it is a break point number,
   otherwise an address of an instruction%                          01428
   LOCAL bpa, bpn;                                                  01429
   IF value < nbkpts THEN                                           01430
      BEGIN                                                         01431
      bpa ← $bptbl + value*bptblentsz;                              01432
      IF [bpa] = 0 THEN RETURN(0);                                  01433
      RETURN(bpa, value);                                           01434
      END;                                                          01435
   IF NOT tblsearch($bptbl, bptblsz, bptblentsz, value: bpa, bpn)
   THEN RETURN(0);                                                  01436
   RETURN(bpa, bpn);                                                01437
   END.                                                             01438
(restbp)PROC;                                                       01439
   %Restore the original instructions%                              01440
   LOCAL c, bptinst;                                                01441
   bptinst ← jsp + $brkchk;                                         01442
   c ← -bptblentsz;                                                 01443
   WHILE (c ← c+bptblentsz) < bptblsz DO                            01444
      IF bptbl[c] # 0 AND [bptbl[c]] = bptinst THEN                 01445
         [bptbl[c]] ← bptbl[c].bpinst;                              01446
   RETURN;                                                          01447
   END.                                                             01448
(setbkpt)PROC(addr, test, testval, bpa, bpn);                       01449
   %Create a breakpoint%                                            01450
   %Return brekpoint number%                                        01451
   LOCAL lbpa; %Brealpoint address, number%                         01452
   IF NOT bpa THEN %Get slot in table%                              01453
      BEGIN                                                         01454
      IF NOT tblsearch($lbptbl, lbptblsz, 1, 0: lbpa) THEN
      err($"Breakpoint Table Full");                                01455
      IF NOT tblsearch($bptbl, bptblsz, bptblentsz, 0: bpa, bpn)
      THEN err($"Breakpoint Table Full");                           01456
      [lbpa] ← bpa;                                                 01457
      END;
   IF [addr] .A smanipumask = smanipulator THEN
      BUMP addr;
   [bpa].bpaddr ← addr;
   [bpa].bptest ← test;
   [bpa].bpval ← testval;
   RETURN(bpn);
   END.
(storbp)PROC;
   %Store the breakpoints in the table%
   LOCAL c, bptinst;
   bptinst ← jsp + $brkchk;
   c ← -bptblentsz;
   WHILE (c ← c+bptblentsz) < bptblsz DO
      IF bptbl[c] # 0 AND [bptbl[c]] # bptinst THEN
         BEGIN
         bptbl[c].bpinst ← [bptbl[c]]; %Save off instruction%
         [bptbl[c]] ← bptinst;
         END;
   RETURN;
   END.
(typbrk)PROC(bp);
   LOCAL bpa, bpn;
   LOCAL STRING tempsr[50];
   IF NOT (bpa ← findbp(bp: bpn)) THEN err($"No Such Breakpoint");
   *tempsr* ← EOL, "BP #", STRING(bpn), ", At ";
   typeas($tempsr);
   typval([bpa].bpaddr, symbolicmode);
   IF [bpa].bptest # 0 THEN
      BEGIN
       CASE [bpa].bptest OF
         =1:
            *tempsr* ← "(Trace)";
         <1B7:
            BEGIN
            ddgtsymbol($tempsr, [bpa].bptest);
            *tempsr* ← "Test Procedure is ", *tempsr*;
            END;
         ENDCASE
            BEGIN
            ddgtsymbol($tempsr, [bpa].bptest.RH);
            *tempsr* ← *tempsr*, SP;
            CASE [bpa].bptest .A opcodmask OF
               =came: *tempsr* ← *tempsr*, '=;
               =camn: *tempsr* ← *tempsr*, '#;
               =caml: *tempsr* ← *tempsr*, '>;
               =camg: *tempsr* ← *tempsr*, '<;
               =camle: *tempsr* ← *tempsr*, ">=";
               =camge: *tempsr* ← *tempsr*, "<=";
               ENDCASE err($"Illegal Breakpoint Relation");
            *tempsr* ← "Test: ",*tempsr*, SP,
            *[octnum([bpa].bpval)]*;
            END;
      crlf();
      typeas($tempsr);
      END;
   crlf();
   RETURN;
   END.
(typtrace)PROC(addr);
   %Types out a trace message for address%
   LOCAL STRING tempsr[50];
   ddgtsymbol($tempsr, addr); %Get a symbol for it%
   %Use TENEX output stuff because crlf or typeas may be traced%

      !bout (101B, EOL);
      !sout (101B, chbmty + $tempsr, -tempsr.L);
   RETURN;
   END.
(xbpclear)PROC;
   %Clear a breakpoint%
   LOCAL bpa, bpn;
   echo($"Clear ");
   echon();
   CASE nextchar OF
      ='A:
         BEGIN
         echo($"ll");
         chkcacr();
         bpclall();
         END;
      ENDCASE
         BEGIN
         backupchar;
         bpa ← parsevalue(0, 0, 0);
         chkcacr();
         IF [bpa] .A smanipumask = smanipulator THEN
            BUMP bpa;
         IF NOT (bpa ←findbp(bpa: bpn)) THEN err($"No Such
         Breakpoint");
         bpclr(bpa);
         bpclr(bpn);
         END;
   RETURN;
   END.
(xbpprint)PROC;
   %Print breakpoint%
   LOCAL bpa, bpn;
   echo($"Print ");
   echon();
   CASE nextchar OF
      ='A:
         BEGIN
         echo&[°"ll");
         chkcacr();
         bppntall();
         END;
      =CA:
         bppntall():
      ENDCASE
```

```
         BEGIN
         backupchar;
         bpa ← parsevalue(0, 0, 0);
         chkcacr();
         typbrk(bpa);
         END;
   RETURN;
   END.
(xbpset)PROC;
   %This is the procedure for setting breakpoints%
   LOCAL bpa, bpn, loc, testval, test;
   echo($"Set at ");
   echon();
   loc ← parsevalue(0, 0, 0);
   loc ← chkbp(loc: bpa, bpn);
   test ← testval ← 0;
   LOOP
      BEGIN
      echoff();
      CASE nextchar OF
         =C.: NULL;
         =CA, =CR: EXIT LOOP;
         ='C:
            BEGIN
            echo($"Call ");
            echon();
            IF  NOT (test ← gprcname()) THEN err($"Not a
            Procedure");
            END;
         ='R:
            BEGIN
            echo($"Replaces Breakpoint: ");
            echon();
            IF NOT (bpa ← findbp(parsevalue(0, 0, 0): bpn)) THEN
            err($"No Such Breakpoint");
            END;
         ='T:
            BEGIN
            echo($"Test (Location relation value): ");
            echon();
            test ← parsevalue(0, 0, 0) .A 18M;
            deblnk();
            CASE nextchar OF
               ='=: test ← test .V came;
               ='#: test ← test .V camn;
               ='<:
                  IF nextchar = '= THEN test ← test .V camge
                  ELSE
                     BEGIN
                     backupchar;
                     test ← test .V camg;
                     END;
               ='>:
                  IF nextchar = '= THEN test ← test .V camle
                  ELSE
                     BEGIN
```

```
                        backupchar;
                        test ← test .V caml;
                        END;
                  =CA, =CR: EXIT LOOP;
                  ENDCASE err($"Illegal Test");
               deblnk();
               testval ← parsevalue(0, 0, 0);
               END:
            ENDCASE err($"??");
         END:
      bpn ← setbkpt(loc, test, testval, bpa, bpn);
      RETURN:
      END.
   (xbrkpt)PROC;
      %This the breakpoint control routine%
      echo($"Breakpoint "):
      CASE nextchar OF
         ='C: xbpclear():
         ='P: xbpprint();
         ='S: xbpset();
         ENDCASE
            BEGIN
            echo($"? (Clear/Print/Set) ");
            REPEAT CASE;
            END;
      RETURN;
      END.
%* CONTINUE command %
   (xcontin)PROC;
      echo($"Continue");
      chkcacr();
      crlf();
      storbp();
      SIGNAL(continsig);
      END.
%* DEFINE command %
   (xdddef)PROC;
      % this routine parses and processes the DEFINE symbol command %
      % a symbol is entered into the ddtsymbol table with the
      specified value only if it does not already exist there (NEW)
      or only if it already exists there (OLD) %
      LOCAL value, addrv, char:
      LOCAL STRING tempstr[20]:
      echo($"DEFINE ");
      CASE char ← nextchar OF
         ='N: % NEW symbol %
            BEGIN
            echo($"New Symbol: ");
            IF ddmrkx = 0
               THEN err($"No Block in Mark Stack for Definition");
            echon();
            IF NOT rdsym($tempstr) THEN err($"Illegal Symbol");
               chkcacr();
            IF ddtlookup($tempstr, 0 : value,addrv)
```

```
          THEN err($"Symbol Already Exists");
        echo($" New value = ");
        IF NOT parsenter( :value) THEN err($"Illegal Value");
          chkcacr();
        ddtenter( $tempstr, value, global );
        END;
      ='O: % OLD symbol %
        BEGIN
        echo($"OLD Symbol: ");
        echon();
        IF NOT rdsym($tempstr) THEN err($"Illegal Symbol");
          chkcacr();
        IF NOT ddtlookup($tempstr, O : value,addrv)
          THEN err($"Symbol Does Not Exist");
        echo($" New value = ");
        IF NOT parsenter( :value) THEN err($"Illegal Value");
          chkcacr();
        [addrv] ← value; % stick in new value %
        END;
    ENDCASE
      BEGIN
      echo($"? (New/Old) ");
      REPEAT CASE;
      END;
    RETURN;
    END.
%* FIND %
  (xddfind) PROCEDURE:
    % this routine parses and processes the FIND command %
    % this command finds all occurrences of a particular value
    between two specified address limits and prints out the
    addresses of the hits %
    LOCAL lower,upper, default;
    LOCAL value,mask;
    LOCAL STRING tempstr[75]:
    LOCAL STRING tempstr1[50];
    echon();
    echo($"Find Content ");
    IF NOT parsenter( :value) THEN err($"Invalid Content
    Expression"):
    chkcacr();
    echo($" Masked by ");
    IF NOT parsenter( :mask) THEN mask ← 777777B; % defalut mask %;
    chkcacr();
    echo($" From: ");
      lower ← parsevalue(0,0,0 : default);
      chkcacr();
      IF default THEN lower ← $sysrtn;
    echo($" To: ");
      upper ← parsevalue(0,0,0 : default);
      chkcacr():
      IF default THEN lower ← 554000B;
    crlf();
```

```
      FOR lower UP UNTIL > upper DO
         BEGIN
         IF value = [lower] .A mask THEN
            BEGIN % found a match  -- now print it out %
            ddgtsymbol( $tempstr, lower); % get representation for
            address %
            ddgtsymbol( $tempstr1, [lower]); % get representation for
            value %
            % edit them together %
            *tempstr* ← *tempstr*, "/  ", *tempstr1*, EOL;
            typeas($tempstr);
            END;
         IF inpstp THEN
            BEGIN
            inpstp ← FALSE;
            EXIT LOOP;
            END;
         END;
      RETURN;
      END.
%* GO TO Command %
   (xgo)PROC;
      LOCAL v;
      echo($"Go to Location: ");
      echon();
      IF (v ← parsevalue(0, 0, 0)) < 1000 THEN err($"Illegal
      Address");
      chkcacr();
      storbp();
      SIGNAL(gosig, v); %To be trapped by runddt%
      END.
%* MARK command %
   (xnmark)PROC;
      %This the mark symbol table control routine%
      echo($"Mark Symbol Table: ");
      CASE nextchar OF
         ='C: xddpop();
         ='P: xmkprint();
         ='S: xddmark();
         ENDCASE
            BEGIN
            echo($"? (Clear/Print/Set) ");
            REPEAT CASE;
            END;
      RETURN;
      END.
   (xmkprint) PROCEDURE;
      % this routine parses and processes the MARK SYMBOL TABLE PRINT
      command %
      LOCAL i,j,k;
      LOCAL STRING tempstr[50];
      echo($"Print");
      chkcacr();
      crlf();
      IF ddmrkx THEN *tempstr* ← "Contents of Mark Stack:"
      ELSE *tempstr* ←  "Mark Stack Empty";
   typeas( $tempstr );
   FOR i ← ddmrkx - 1 DOWN UNTIL < 0 DO
      BEGIN
      j ← [ddmrk[i]-1]; % RADIX50 name representation %
      *tempstr* ← NULL;
      crlf();
      LOOP % convert from radix 50 to ascii %
         BEGIN
         DIV j/50B, j, k;
         IF NOT k THEN EXIT LOOP;
         k ← IF (k ← k + 47) > 57 THEN k+7 ELSE k;
         *tempstr* ← k , *tempstr*;
         END;
      typeas($tempstr);
      END;
   RETURN;
   END.
(xddmark) PROCEDURE;
   % this routine parses and processes the MARK SYMBOL TABLE
   command %
   LOCAL i,j,k;
   LOCAL STRING tempstr[50];
   echo($"Set At: ");
   echon();
   IF NOT rdsym($tempstr) THEN err($"Illegal Block Name");
   chkcacr();
   ddtmark($tempstr);
   RETURN;
   END.
(xddpop) PROCEDURE;
   % this routine parses and processes the POP SYMBOL TABLE
   command %
   LOCAL i,j,blockptr,namevalue;
   LOCAL STRING tempstr[50];
   echo($"Clear Block ");
   echon();
   IF NOT rdsym($tempstr) THEN err($"Illegal Block Name");
   chkcacr();
   namevalue ← ddtname( $tempstr, 1) .A 32M;
   % try to find the names symbol in the mark stack %
   FOR i ← ddmrkx - 1 DOWN UNTIL < 0 DO
      BEGIN
      blockptr ← ddmrk[i];
      IF [blockptr-1] = namevalue THEN
         BEGIN % found block, now collapse mark stack %
         FOR j ← i UP UNTIL >= ddmrkx - 1  DO
            ddmrk[j] ← ddmrk[j+1];
         ddmrkx ← ddmrkx - 1;
         % check to see if block being deleted is at the bottom of
         the DDT symbol table %
         ddtchkp( blockptr );
         RETURN;
         END;
      END;
   err($"Block Not In Mark Stack");
   END.
```

```
%* PROCEDURE call, replace, backup%
   (gprcname)PROC;
      LOCAL symval, symtype, c;
      IF NOT getsym(: symtype, symval) THEN RETURN(0);
      IF ([symval].accum10 # $s AND [symval].addr10 # $sysovr) THEN

         IF [symval].opcod10 # 254B %JRST% THEN RETURN(0)
         ELSE
            IF tblsearch($proctbl, proctblsz, proctblentsz, symval:
            c) THEN RETURN(symval, c)
            ELSE RETURN(FALSE);
      RETURN(symval, 0);
      END.
   (xbackp)PROC;
      LOCAL prl, c;
      echo($"Back up to ");
      echon();
      IF NOT (prl ← gprcname( :c)) THEN err($"Illegal Procedure
      Name"):
      chkcacr();
      IF c = 0 THEN err($"Not in Replace List");
      [prl] ← [c+1]; %original inst%
      [c] ← 0; %Clear it%
      RETURN;
      END.
   (xcallp)PROC(prcad, frm, fp);
      %Fake a call to prcad%
      storbp(); %Set up breakpoints%
      ddtjunk ← prcad;
      s ← m;
      IF fp > 0 THEN
         BEGIN
         r2.LH ← frm:
         r2.RH ← m+1;
         r3 ← m.RH + fp;
         !BLT 2,@3:
         END:
      !JSP urstrg;
      GOTO [ddtjunk];
      END.
   (xcalprc)PROC;
      LOCAL prcadr. frm[maxprms], fp. char, rd, c, symtype, symval;

      REF  rd;
      echo($"Call "):
      echon();
      IF NOT prcadr ← gprcname() THEN err($"Not a Procedure");
      fp ← 0:
      IF (char ← nextchar) = '( THEN
         LOOP
            BEGIN
            parsenter(:frm[fp]);
            IF (fp ← fp+1) > maxprms THEN err($"Too Many
            Parameters");
            IF (char ← nextchar) # ', AND char # '/ THEN
               IF char # ') THEN err($"Illegal Paremeter")
            ELSE EXIT;
         END
   ELSE
      IF char # CA THEN
         BEGIN
         IF NOT getsym(: symtype, symval)
         OR NOT stacksymbol THEN err($"Illegal Frame Designator");
         &rd ← symval+1;
         framep ← CASE recordstart OF
            =frametoptype: frametop;
            ENDCASE framep;
         initframe();
         IF nparms > maxprms THEN err($"Too Many Parameters");
         c ← 0;
         WHILE (c←c+1) <= nparms DO
            frm[fp := fp+1] ← [framep + c];
         END
      ELSE backupchar;
   chkcacr();
   %Clear input buffer so we can read characters%
      clrbuf(0);
   crlf();
   xcallp(prcadr, $frm, fp);
   c ← a1; %save off result%
   !JSP usvreg; %Save off user registers%
   restbp(); %Restore break points%
   clrbuf(0); %Get rid of any junk left over%
   crlf();
   typeas( $"Return Value = ");
   typval(c, numericmode); %Type out result%
   RETURN;
   END.
(xprocc)PROC;
   echo($"Procedure ");
   CASE nextchar OF
      ='C:
         xcalprc();
      ='R:
         xreplp();
      ='B:
         xbackp();
      ENDCASE
         BEGIN
         echo($"? (Backup/Call/Replace) ");
         REPEAT CASE;
         END;
   RETURN;
   END.
(xreplp)PROC;
   LOCAL pr1, pr2, c;
   echo($"Replace Procedure ");
   echon();
   IF NOT (pr1 ← gprcname()) THEN err($"Illegal Procedure Name");
```

```
      chkcacr();
      echo($" By ");
      IF NOT (pr2 ← gprcname()) THEN err($"Illegal Procedure Name");
      chkcacr();
      IF NOT tblsearch($proctbl, proctblsz, proctblentsz, 0: c) THEN
         BEGIN
         crlf();
         typeas($"Backup Table Full...Replaced Instruction is:");
         typval([pr1], symbolicmode):
         END
      ELSE
         BEGIN
         [c] ← pr1:
         [ c +1] ← [ pr1];
         END;
      [pr1] ← jrst + pr2;
      RETURN;
      END.
%* RECORD Pointer Set%
   (xrpset)PROC;
      %Set up the record pointer RECP%
      LOCAL symval, symtype;
      echo($"Record Pointer Set to: ");
      echon();
      recprepl(0);
      RETURN;
      END.
%* SHOW%
   (chkmode)PROC( caflag);
      LOCAL char, mode;
      mode ← currenttypemode;
      echoff():
      IF NOT caflag THEN RETURN(mode);
      IF (char ← nextchar) = C. THEN
         BEGIN
         CASE nextchar OF
            ='S:
               BEGIN
               echo($" Symbolic  ");
               mode ← symbolicmode:
               END:
            ='N:
               BEGIN
               echo($" Numeric  ");
               mode ← numericmode;
               END;
            ='T:
               BEGIN
               echo($" Text  ");
               mode ← textmode:
               END:
            ENDCASE err($"Illegal Mode Spec");
         END
   ELSE backupchar;
   IF char = '[ THEN mode ← numericmode;
   IF char = '/ THEN mode ← symbolicmode;
   currenttypemode ← mode;
   RETURN(mode):
   END.
(xnshow)PROC(oldrd, echflg):
   LOCAL rd[ribsize], mode, caflag, char;
   REF oldrd;
   IF echflg THEN echo($"Show ");
   copyrd(&oldrd, $rd);
   caflag ← TRUE;
   CASE (char ← nextchar) OF
      ='R: %Record%
         BEGIN
         echo($"Record ");
         IF recp = 0 THEN err($"Record Pointer Undefined");
         IF NOT recordentity OR recordname # recp THEN
            BEGIN
            entity ← recordtype;
            dynamicrecord ← FALSE:
            recordname ← recp;
            initialised ← FALSE:
            END;
         addressmode ← macro;
         END;
      ='S: %String%
         BEGIN
         echo($"String ");
         entity ← stringtype;
         dynamicrecord ← TRUE;
         defineproc ← $strngrec:
         addressmode ← macro;
         initialised ← FALSE;
         END;
      ='L:
         BEGIN
         echo($"Location "):
         REPEAT(char ← SP);
         END;
      ='↑:
         %Predecessor%
         BEGIN
         IF echflg THEN echo($"Preceeding ");
         chngadr($rd, -1);
         (reltype): %Come here to type relative%
            typloc($rd): %Type out address%
            caflag ← 0;
            GOTO comtype;
         END;
      =LF, ='N:
         %Next%
         BEGIN
         IF echflg THEN echo($"Next ");
         chngadr($rd, 1);
         GOTO reltype:
```

```
         END;
      =CA: % command accept -- may be a bug %
         BEGIN
         backupchar;
         IF displayflag THEN REPEAT CASE (SP) ELSE GOTO comtype;
         END;
      ='←,  % assign a value %
      =CR, =C.: %Same thing over%
         BEGIN
         backupchar;
         GOTO comtype;
         END;
      =TAB: %Move address to last value%
         BEGIN
         recordstart ← lv;
         caflag ← 0;
         GOTO comtype;
         END;
      ENDCASE
         BEGIN
         %Set to location / word%
         IF char # SP THEN
            BEGIN
            typech(char);
            backupchar;
            END;
         entity ← wordtype;
         dynamicrecord ← TRUE;
         defineproc ← $wordrec;
         mother ← 0;
         addressmode ← macro;
         initialised ← FALSE;
         END;
     %Now read and parse the address%
      showaddr($rd);
    %Now type it out%
      (comtype):
      %Check for mode%
         mode ← chkmode(caflag);
      typentity($rd, mode);
    %Now update old address%
      copyrd($rd, &oldrd);
    %Replace if appropriate%
      IF caflag THEN repval(&oldrd);
    RETURN;
    END.
%* TRACE%
  (xtrace)PROC;
    LOCAL taddr, bpn, bpa;
    %Set up trace%
    echo($"Trace Location ");
    echon();
    IF (taddr ← chkbp(parsevalue(0, 0, 0): bpa, bpn)) < 1000 THEN
    err($"??");
    chkcacr();
```

```
         setbkpt(taddr, 1, 0, bpa, bpn);
         RETURN;
         END.
   %* VALUE Command%
      (xtval)PROC:
         LOCAL v;
         echo($"Value of ");
         echon();
         v ← parsevalue(0, 0, 0);
         typeas($" is ");
         typval(v, chkmode(TRUE));
         chkcacr();
         RETURN;
         END.
%* Record Information Block Routines%
   %* RIB Utilities%
      (initrd)PROC(rd);
         %Initialises an RD...Assumes that the followinig fields are valid when called:
            recordstart
            mother
            recordname
            dynamicrecord (and defineproc if applicable)
            entity
            addressmode
         Initialises the following filds to 0 in all cases:
            fieldnumber
            daughter
            suppressloc
         Calls the defineproc if dynamicrecord, or otherwise setsup
            numberfields
            recordsize
            overflowaction
            underflowaction
         %
         LOCAL calprc, value, fld, mode;
         LOCAL STRING tempsr[50];
         REF rd, calprc, fld;
         IF initialised THEN RETURN;
         fieldnumber ← daughter ←  suppressloc ← 0;
         IF dynamicrecord THEN
            BEGIN
            calldefineproc(&rd);
            END
         ELSE
            BEGIN
            overflowaction ← underflowaction ← proceedaction;
            replaceaction ← defaultreplace;
            numberfields ← recordname.LH;
            value ← recordname.RH + recordname.LH-1; %Pointer to last field%
            recordsize ← [value].bpadr + 1; %Number of words in record%
            IF numberfields > maxnumfields THEN err($"Record Too Large");
         value ← 0:
         mode ← IF macromode AND (numberfields > 1) AND rfnmfg THEN
         $trfname ELSE defaultmode;
         &fld ← fieldstart;
         DO
            BEGIN
            fieldptr ← [recordname+value];
            fieldptr.bpindx ← 0;
            typemode ← mode;
            &fld ← &fld + fielddescsz;
            END UNTIL (value ← value + 1) >= numberfields;
         END;
      initialised ← TRUE;
      currenttypemode ← symbolicmode; % default printout type %
      RETURN;
      END.
   (setrd)PROC(rd);
      LOCAL c;
      REF rd:
      c ← -1;
      WHILE (c ← c+1) < rdsize DO rd[c] ← 0;
      RETURN:
      END.
%* Special RIB's [dynamic]%
   %  General Comments%
      %A record definition procedure recieves one parm, RD.
      It is expected to set up the fields:
         Numbrfields
         recordsize,
         overflowaction
         underflowaction
      Plus all of the field descroptors required.
      It may diddle anything else in the RD which it wishes to%
   %Locals%
      (localdef)PROC(rd);
         LOCAL fld. c;
         REF rd, fld;
         fieldnumber ← 0;
         recordsize ← IF framesize <= nparms THEN 0 ELSE  framesize -
         nparms;
         numberfields ←MIN(recordsize, maxnumfields);
         replaceaction ← defaultreplace;
         overflowaction ← $bumpframe;
         underflowaction ← $preframe:
         dynamicrecord ← TRUE;
         recordstart ← framep + nparms;
         entity ← recordtype;
         addressmode ← macro;
         &fld ← fieldstart;
         c ← 0:
         WHILE (c ← c+1) <= numberfields DO
            BEGIN
            fieldptr ← wordbyteptr+nparms+c;
            typemode ← $typlcal;
            &fld ← &fld + fielddescsz;
            END;
         RETURN;
         END.
      (typlcal)PROC(bp, rd);
         LOCAL STRING tempsr[4];
         REF rd;
         crlf();
         typech('L);
         *tempsr* ← STRING(fieldnumber);
         typeas($tempsr):
         typval(?bp, currenttypemode);
         RETURN;
         END.
   %Parms%
      (parmdef)PROC(rd);
         LOCAL fld, c;
         REF rd, fld;
         fieldnumber ← 0:
         recordsize ← numberfields ← nparms;
         replaceaction ← defaultreplace;
         overflowaction ← $bumpframe:
         underflowaction ← $preframe:
         dynamicrecord ← TRUE;
         recordstart ← framep;
         entity ← recordtype;
         addressmode ← macro;
         &fld ← fieldstart:
         c ← 0:
         WHILE (c ← c+1) <= numberfields DO
            BEGIN
            fieldptr ← wordbyteptr + c;
            typemode ← $typparm;
            &fld ← &fld + fielddescsz;
            END;
         RETURN;
         END.
      (typparm)PROC(bp, rd);
         LOCAL STING tempsr[4]:
         REF rd;
         crlf();
         typech('P);
         *tempsr* ← STRING(fieldnumber);
         typeas($tempsr);
         typval(.bp, currenttypemode);
         RETURN;
         END.
   %Recp [Record Pointer%
      (recpdef)PROC(rd);
         LOCAL fld:
         REF rd, fld:
         recordstart ← $recp;
          numberfields ← 2:
         recordsize ← 1;
         overflowaction ← underflowaction ← 0;
         addressmode ← macro;
         replaceaction ← $recprepl:
         &fld ← fieldstart;
      fieldptr ← recprecordaddr;
      typemode ← $recpt1;
      &fld ← &fld + fielddescsz;
      fieldptr ← recprecsize;
      typemode ← $recpt2;
      RETURN;
      END.
   (recprep1)PROC(rd);
      LOCAL symtype, symval;
      REF rd;
      IF NOT getsym( :symtype, symval)
         OR NOT programsymbol
          OR NOT ckbptr([symval.RH])
         OR symval.LH = 0 THEN
            err($"Illegal Record Designator");
      chkcacr();
      regp ← symval;
      RETURN;
      END.
   (recpt1)PROC(bp, rd);
      LOCAL STRING tempsr[40];
      crlf();
      typeas($"First Field in Record is ");
      ddgtsymbol($tempsr, .bp);
      typeas($tempsr);
      RETURN;
      END.
   (recpt2)PROC(bp, rd);
      typech(',);
      typval(.bp, numericmode);
      typeas($" Fields in Record");
      RETURN;
      END.
%Stack Frame%
   (bumpframe)PROC(rd);
      REF rd;
      IF framep = frametop THEN err($"Top Of Stack reached");

      framep ← framep + framesize + 2;
      initframe();
      RETURN;
      END.
   (framedef)PROC(rd);
      LOCAL fld;
      REF rd, fld;
      %When called, recordstart contains:
         1 for frametop
         2 for framebase
         address for current frame
      %
      CASE recordstart OF
         =frametoptype:
            BEGIN
            framep ← frametop;
            initframe();
            END;
      =framebasetype:
         BEGIN
         framep ← framebase;
         initframe();
         END;
      ENDCASE NULL;
   recordstart ← framep;
   overflowaction ← $bumpframe;
   underflowaction ← $preframe;
   numberfields ← 1;
   suppressloc ← TRUE;
   fieldnumber ← 0;
   recordsize ← framesize;
   addressmode ← macro;
   setfieldptr;
   fieldptr ← 0;
   typemode ← $typframe;
   RETURN;
   END.
(framfielddef)PROC(rd);
   LOCAL fld;
   REF rd, fld;
   %When called, recordstart contains:
      1 for mark
      2 fo return
      3 for SIGNAL value
   %
   overflowaction ← $bumpframe;
   underflowaction ← $preframe;
   numberfields ← 1;
   suppressloc ← TRUE;
   fieldnumber ← 0;
   recordsize ← framesize;
   addressmode ← macro;
   setfieldptr;
   fieldptr ← CASE recordstart OF
      =1: marklocfield;
      =2: retlocfield;
      =3: siglocfield;
      ENDCASE fieldptr; %Change addr call%
   typemode ← defaultmode;
   recordstart ← framep;
   RETURN;
   END.
(initframe)PROC;
   %This procedure sets up the relevant globals for the stack
   frame stuff.
      It gets called at initialisation, and ehenever the frame
      poiter (framep) gets chnged%
   LOCAL t, t1;
   %Find out how many parms%
      t ← [returnloc-1];
      IF t.opcod10 = opcall0 THEN nparms ← 0
      ELSE IF t.opcod10 = opcall1 THEN nparms ← 1
      ELSE IF t.opcod10 = opcallm THEN
         nparms ← t.accum10 - 2
         ELSE nparms ← [returnloc-2].addr10-2;
      %Get the framesize%
         t ← stacktop;
         WHILE [t].marklocfield # framep DO t ← [t].marklocfield;
         framesize ← t-framep-2;
      RETURN;
      END.
   (preframe)PROC(rd);
      REF rd;
      IF framep = framebase THEN err($"Bottom of Stack reached");
      framep ← framemark;
      initframe();
      RETURN;
      END.
   (typframe)PROC(bp, rd);
      LOCAL STRING tempsr[50];
      REF rd;
      %First get current procedure name%
          ddgtsymbol($tempsr, [returnloc-1].addr10);
         crlf();
         typeas($"Procedure ");
         typeas($tempsr);
         crlf();
         ddgtsymbol($tempsr, returnloc-1);
         typeas($"Called From ");
         typeas($tempsr);
         *tempsr* ← STRING(nparms), " Parameters", EOL,
         STRING(framesize-nparms), " Locals";
         crlf();
         typeas($tempsr);
      RETURN;
      END.
%Strings%
   (replstring)PROC(rd);
      LOCAL string;
      REF rd, fld, string;
      %First Read the string%
         &string ← getstring(2000, $dspblk);
         txtlit(&string);
      %Now replace it%
         *[recordstart]* ← *string*;
         freestring(&string, $dspblk);
      RETURN;
      END.
   (strngrec)PROC(rd);
      LOCAL fld;
      REF rd, fld;
      %Check validity%
         IF [recordstart] = 0
         OR [recordstart].M > 3000
         OR [recordstart].L > [recordstart].M THEN err($"Illegal
         Format String");
      numberfields ← 3;
      recordsize ← ([recordstart].M + 9) / 5;
      overflowaction ← proceedaction;
      underflowaction ← $strngunderflow;
      addressmode ← macro;
      replaceaction ← $replstring;
      &fld ← fieldstart;
      fieldptr ← stringmax;
      typemode ← $typstmx;
      &fld ← &fld + fielddescsz;
      fieldptr ← stringlength;
      typemode ← $typstsz;
      &fld ← &fld + fielddescsz;
      fieldptr.xptr ← 77B; %Make it an X-pointer%
      xptrsize ← [recordstart].L;
      xptrbytesz ← 7; %Standard A-string alpha%
      fieldptr.xptadd ← 5; %First character we wish to print%
      typemode ← textmode;
      RETURN;
      END.
   (strngunderflow)PROC(rd);
      LOCAL t, count;
      REF rd;
      t ← recordstart;
      count ← 0;
      WHILE (count ← count + 1) <= maxwordsinstring DO
         BEGIN
         BUMP DOWN t;
         IF [t] # 0 THEN
            IF ((([t].M + 9)/5) = count) AND ([t].L <= [t].M) THEN
               BEGIN
               recordstart ← t;
               RETURN;
               END;
         END;
      err($"No Preceeding String");
      END.
   (typstmx)PROC(bp, rd);
      LOCAL STRING tempsr[19];
      REF rd;
       *tempsr* ← "     <", STRING(.bp), ':;
      typeas($tempsr);
      RETURN;
      END.
   (typstsz)PROC(bp, rd);
      LOCAL STRING tempsr[19];
      REF rd;
       *tempsr* ← STRING(.bp), '>, EOL;
      typeas($tempsr);
      RETURN;
      END.
%Words%
   (wordrec)PROC(rd);
      LOCAL fld;
      REF rd, fld;
      numberfields ← 1;
```

```
         recordsize ← 1;
         overflowaction ← underflowaction ← proceedaction;
         addressmode ← macro; %Just to make sure%
         replaceaction ← $replfield;
         dynamicrecord ← TRUE; %For next, provious%
         setfieldptr;
         fieldptr ← wordbyteptr;
         typemode ← defaultmode;
         RETURN;
         END.
%* Internal Hash Table%
   % Calls %
      (ddtsym) PROC( string );
         LOCAL v1,v2,v3;
         LOCAL STRING tempsr[40];
         REF string;
         *tempsr* ← *string*;
         astruc($tempsr);
         v1 ← hashlookup( $tempsr : v2,v3);
         RETURN( v1,v2,v3 );
         END.
   %Utilities%
      (ddthash)PROC(string);
         LOCAL STRING tempsr[5];
         REF string;
         tempsr[1] ← 0;
         IF string.L >= 5 THEN
            tempsr[1] ← string[1]
         ELSE
            WHILE tempsr.L < string.L DO
               *tempsr* ← *tempsr*, *string*[tempsr.L + 1];
         RETURN((((tempsr[1] .A 35M) / 2) +string.L) MOD hashbase);
         END.
      (hashdump)PROC;
         %Type out the contents of te hash table%
         LOCAL nent, stkusd, maxdepth, hash, t, t1, t2;
         LOCAL STRING tempsr[100];
         hash ← nent ← stkusd ← maxdepth ← 0;
         LOOP
            BEGIN
            IF hashstack[hash] # 0 THEN
               BEGIN
               BUMP stkusd;
               *tempsr* ← EOL, STRING(hash);
               %typeas($tempsr);%
               t ← hashstack[hash];
               t2 ← 0;
               DO
                  BEGIN
                  BUMP nent, t2;
                  t1 ← t + [t].LH;
                  *tempsr* ← EOL, "      ", *[t.RH+1]*, "( ",
                  *[octnum([t1])]*, ", ", *[octnum([t1+1])]*, ')';
                  %typeas($tempsr);%
                  END
         UNTIL (t ← [t].RH) = 0;
         maxdepth ← MAX(maxdepth, t2);
         END;
      IF (hash ← hash + 1)>= hashbase THEN EXIT;
      END;
   %Now type out end stuff - commented out - CFD
   *xlit* ← EOL, EOL, "Number Entries = ", *[octnum(nent)]*,
      EOL, "Maximum Depth = ", *[octnum(maxdepth)]*,
      EOL, "Stack Use = ", *[octnum(stkusd)]*, '/,
      *[octnum(hashbase)]*,
      EOL, "Hash Table Storage = ",
      *[octnum(hashptr-$hashtable)]*, '/, *[octnum(hashtblsz)]*;
   typeas($xlit); --->>> %
   RETURN;
   END.
(hashenter)PROC(string, value, record, rcdlngth);
   %enter string if not already entered%
   LOCAL  hash, index;
   REF string;
   hash ← ddthash(&string);
   IF hashstack[hash] # 0 THEN
      IF hashlookup(&string) THEN RETURN(FALSE); %Duplicate%
   [hashptr].RH ← hashstack[hash]; %Link to next entry in stack%
   hashstack[hash] ← hashptr;
   [hashptr+1].M ← (hashtblsz - 1 - (hashptr-$hashtable))*5;
   *[hashptr+1]* ← *string*;
   hashptr ← hashptr + ([hashptr].LH ←(string.L + 14)/5);
   [hashptr] ← value;
   BUMP hashptr;
   IF rcdlngth = 1 THEN [hashptr] ← record
   ELSE mvbfbf(record, hashptr, rcdlngth);
   hashptr ← hashptr + rcdlngth;
   [hashptr] ← 0;
   RETURN;
   END.
(hashlookup)PROC(string);
   LOCAL hash, t, t1;
   REF string;
   t ← $hashstack +ddthash(&string);
   WHILE (t1 ← [t].RH) # 0 DO
      BEGIN
      IF *string* = *[t1+1]* THEN
         BEGIN
         t ← t1 + [t1].LH;
         RETURN(TRUE, [t], t+1);
         END;
      t ← t1;
      END;
   RETURN(FALSE);
   END.
(hashtbinit)PROC;
   LOCAL t;
   IF hashptr # 0 THEN RETURN; %Already Initialised%
   hashptr ← $hashtable;
```

```
      hashtable ← 0;
      t ← -1;
      WHILE (t←t+1) < hashbase DO hashstack[t] ← 0;
      RETURN;
      END.
   (initht)PROC;
      %INitialise DDT internal Hash Table%
      LOCAL rdx, rd[ribsize];
      IF hashptr # 0 THEN RETURN; %Already DOne%
      hashtbinit();
      %Do the internal words%
         %Do the Registers%
            hashenter($"R1", internalsym, 0, 1);
            hashenter($"R2", internalsym, 1, 1);
            hashenter($"R3", internalsym, 2, 1);
            hashenter($"R4", internalsym, 3, 1);
            hashenter($"R5", internalsym, 4, 1);
            hashenter($"R6", internalsym, 5, 1);
            hashenter($"R7", internalsym, 6, 1);
            hashenter($"PP", internalsym, 7B, 1);
            hashenter($"S", internalsym, 10B, 1);
            hashenter($"M", internalsym, 11B, 1);
            hashenter($"RP", internalsym, 12B, 1);
            hashenter($"A1", internalsym, 13B, 1);
            hashenter($"A2", internalsym, 14B, 1);
            hashenter($"A3", internalsym, 15B, 1);
            hashenter($"A4", internalsym, 16B, 1);
         %Now do rest of internal symbols%
            hashenter($"LV", internalsym, $lv + indirectmode-$regl,
            1);
            hashenter($"EC", internalsym, $escchar-$regl, 1);
            hashenter($"SF", internalsym, $symflg-$regl, 1);
            hashenter($"RNAMES", internalsym, $rfnmfg-$regl, 1);
            hashenter($"LC", internalsym, $lc + indirectmode-$regl,
            1);
      %Now do te internal records%
         rdx ← 0;
         %RECP%
            setrd($rd);
            recordname ← $recpdef;
            dynamicrecord ← TRUE;
            entity ← recordtype;
            hashenter($"RECP", seqsymbol, $rdx, rdsize+1);
            hashenter($"R", seqsymbol, $rdx, rdsize+1);
         %Stack Frame Poiter%
            setrd($rd);
            recordstart ← $framep;
            recordname ← $framedef;
            mother ← daughter ← 0;
            dynamicrecord ← TRUE;
            entity ← recordtype;
            hashenter($"FRAME", seqsymbol + stacksymv, $rdx,
            rdsize+1);
            hashenter($"F", seqsymbol + stacksymv, $rdx, rdsize+1);
```

```
%PARMATERS%
   setrd($rd);
   recordstart ← $framep;
   recordname ← $parmdef;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"PARMS", seqsymbol, $rdx, rdsize+1);
   hashenter($"P", seqsymbol, $rdx, rdsize+1);
%LOCALS%
   setrd($rd);
   recordstart ← $framep;
   recordname ← $localdef;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"LOCALS", seqsymbol, $rdx, rdsize+1);
   hashenter($"L", seqsymbol, $rdx, rdsize+1);
%Stack top%
   setrd($rd);
   recordstart ← frametoptype;
   recordname ← $framedef;
   mother ← daughter ← 0;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"TOP", seqsymbol + stacksymv, $rdx, rdsize+1);
%Frame Base%
   setrd($rd);
   recordstart ← framebasetype;
   recordname ← $framedef;
   mother ← daughter ← 0;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"BASE", seqsymbol + stacksymv, $rdx,
   rdsize+1);
%Mark%
   setrd($rd);
   recordstart ← 1;
   recordname ← $framfielddef;
   mother ← daughter ← 0;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"MARK", seqsymbol, $rdx, rdsize+1);
%Re¯urnloc%
   setrd($rd);
   recordstart ← 2;
   recordname ← $framfielddef;
   mother ← daughter ← 0;
   dynamicrecord ← TRUE;
   entity ← recordtype;
   hashenter($"RET", seqsymbol, $rdx, rdsize+1);
%Signal Loc%
   setrd($rd);
   recordstart ← 3;
   recordname ← $framfielddef;
   mother ← daughter ← 0;
   dynamicrecord ← TRUE;
```

```
            entity ← recordtype;
            hashenter($"SIG", seqsymbol, $rdx, rdsize+1);
        hashdump();
        RETURN;
        END.
%* Utilities%
    (ckbptr)PROC(bp);
      %Checks the validity of a byte ponter...returns False if no good%

      IF bp.bpsize > 36
      OR bp.bpbitpos > 36 THEN RETURN(FALSE);
      RETURN(TRUE);
      END.
    (copyrd)PROC(rd1, rd2);
      mvbfbf(rd1, rd2, ribsize);
      RETURN;
      END.
    (sethint)PROC;
      %Set up control H as interrupt%
      levtbl ← levtab.RH;
      chntab[5] ← $ddtint .V 1B6;
      r1 ← 4B5;
      r2 ← $chntab;
      !HRLI 2,levtab;
      !JSYS sir;
      r1 ← 10000005B;
      !JSYS ati;
      r1 ← 4B5;
      r2 ← 1B10;
      !JSYS aic;
      RETURN;
      %DDT Interrupt handling stuff%
        (ddtint):
        !SOS @levtbl; %Point to actual instruction interrupted%
        %Check for stack manipulation instruction%
          !HRR 0,@levtbl;
          !MOVE 0,@0;
          !MOVEM 0,ddtinst; %Save off inst in case we want to execute
          it later%
          !AND 0,smanipumask;
          !CAME 0,smanipulator;
          !JRST ddtntl;
          %By here, interrupted at stack manipulation
          instruction...execute it%
            !XCT ddtinst;
            !AOS @levtbl;
        (ddtntl): %Call ddt %
          !HRLI 0,callddt;
          !HRR 0,@levtbl;
          !HLRM 0,@levtbl;
          !JSYS debrk;
      END.
    (unsethint)PROC;
      %remove control H as interrupt%
      r1 ← 10B;
      !JSYS dti;
      RETURN;
      END.
   (tblsearch)PROC(tbl, tblsize, entsize, value);
      %Generalised table searcher...return TRUE, address of entrym and
      entry number if ok, FALSE otherwise%
      LOCAL c;
      REF tbl;
      c ← -entsize;
      WHILE (c ← c+entsize) < tblsize DO
         IF tbl[c] = value THEN RETURN(TRUE, $tbl[c], c/entsize);
      RETURN(FALSE);
      END.
FINISH
```

NLSSKI

NLSSRT NLSSRT NLSSRT NLSSRT NLSSRT NLSSRT NLSSRT NLSSRT

[illegible]

```
<VICTOR>NLSSRT.NLS;3, 7-MAY-73 20:10 KEV ;  nlssrt.mac

search stenex
title nlssrt

; defined possible entry points
   nls==0       ;main entry point
   tnls= 2      ;tnls entry point
   dex==3       ;dex entry point
   dnls==4      ;dnls entry point
   nic==5       ;query entry point

; make sure entloc is defined
   ifndef entloc,<entloc==0>

start:
   movei 1,101
   gttyp
   move 1,2
   hrroi 2,[asciz /<NETSYS>NLS.NEW/] ;for non IMLACs
   caie 1,5
   cain 1,6
   hrroi 2,[asciz /<netsys>nls.old/] ;for IMLACs
   hrlzi 1,100001
   gtjfn
   haltf
   move 2,[XWD geter,3]
   blt 2,6
   hrli 1,400000
   jrst 3
geter:
   get
   movei 1 400000
   gevec
   jrst entloc(2)

   end start
```